С.Ф. БАРРЕТТ Д.Дж. ПАК

ВСТРАИВАЕМЫЕ СИСТЕМЫ

ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЙ НА МИКРОКОНТРОЛЛЕРАХ СЕМЕЙСТВА 68HC12 / HCS12 С ПРИМЕНЕНИЕМ ЯЗЫКА C

ДМК ПРЕСС

Москва, 2007

Embedded Systems Design and Applications with the 68HC12 and HCS12

Steven F. Barrett
University of Wyoming

Daniel J. Pack
United States Air Force Academy

Upper Saddle River, New Jersey 07458

ВСТРАИВАЕМЫЕ СИСТЕМЫ
Проектирование приложений на микроконтроллерах семейства 68HC12/HCS12 с применением языка C

С. Ф. Баррет
University of Wyoming

Д. Дж. Пак
United States Air Force Academy

Перевод с английского
Т. В. Ремизевич

Научный редактор
Д. И. Панфилов

Издательский дом «ДМК-пресс»
Москва, 2007

УДК **004.318-181.4**
ББК **32.973.26-04**
Б24

Барретт С. Ф., Пак Д. Дж.
Б24 Встраиваемые системы. Проектирование приложений на микроконтроллерах семейства 68HC12 / HCS12 с применением языка С. — М.: Издательский дом «ДМК-пресс», 2007. — 640 с.

ISBN 5-9706-0034-2

В книге последовательно рассматриваются все этапы создания встраиваемых систем на микроконтроллерах с применением современных технологий проектирования. Задумав эту книгу, авторы поставили перед собой задачу научить читателя искусству создания реальных устройств управления на однокристальных микроконтроллерах.

Издание содержит материал, охватывающий все вопросы проектирования, включает множество заданий для самостоятельной работы, примеры программирования, примеры аппаратных решений и эксперименты по исследованию работы различных подсистем микроконтроллеров.

Данная книга является прекрасным учебным пособием для студентов старших курсов технических университетов, которые предполагают связать свою профессиональную деятельность с проектированием и внедрением встраиваемых микропроцессорных систем. Книга также будет полезна разработчикам радиоэлектронной аппаратуры на микроконтроллерах.

ББК 32.872
УДК 621.396.6

ISBN 0-13-140141-6 (анг.)
ISBN 5-9706-0034-2

ПРЕДИСЛОВИЕ

Структура книги 16
Учебные системы 17
Целевая аудитория 18
Благодарности 19

Глава 1. ПЕРВОЕ ЗНАКОМСТВО СО ВСТРАИВАЕМЫМИ СИСТЕМАМИ

1.1. Что такое встраиваемая система? 22
1.2. Особенности встраиваемых систем 25
1.2.1. Работа в реальном времени 25
1.2.2. Миниатюризация размеров и процесс тестирования 26
1.2.3. Минимизация энергии потребления 26
1.2.4. Интерфейс пользователя и интерфейс сопряжения с объектом 27
1.2.5. Многозадачность 27
1.2.6. Минимизация стоимости 27
1.2.7. Ограничение объема памяти 27
1.2.8. Программно–аппаратный дуализм 28
1.3. Введение в микроконтроллеры семейства 68HC12 и HCS12 29
1.4. Микроконтроллеры HCS12 33
1.4.1. Семейство HCS12 35
1.4.2. Обозначения МК 35
1.4.3. Модельный ряд HCS12 35
1.5. Заключение по главе 1 36
1.6. Вопросы и задания 36

Глава 2. ПРОГРАММИРОВАНИЕ ВСТРАИВАЕМЫХ СИСТЕМ И СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

2.1. Почему мы программируем микроконтроллеры на Си? 44
2.2. Преимущества программирования на языке ассемблер.....45
2.3. Преимущества языков высокого уровня 45
2.3.1. Выбираем язык высокого уровня для программирования встраиваемых систем 47
2.3.2. Краткая история языка Си 47
2.4. Оптимальная стратегия - программирование на Си и на ассемблере.....48
2.5. Структурное проектирование 49
2.5.1. Основные положения метода структурного проектирования 49
2.5.2. Документирование программ.....58
2.5.3. Как язык Си соотносится со структурным проектированием 59
2.6. Рабочие тетради 59
2.6.1. Порядок ведения записей 59
2.6.2. Содержание записей.....60
2.7. Блок-схемы алгоритмов.....60
2.8. Пример применения.....62
2.9. Заключение по главе 2 63
2.10. Что еще почитать? 65
2.11. Вопросы и задания 66

Глава 3. ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ НА СИ

3.1. Введение в программирование на Си.....71
3.1.1. Глобальные и локальные переменные 71
3.2. Типы данных в Си 72
3.3. Операторы языка Си.....76
3.4. Функции.....82
3.4.1. Что такое функция? 82
3.4.2. Основная программа 83
3.4.3. Прототипы функций.....84
3.4.4. Описание функций 85
3.4.5. Вызов функций, передача параметров, возврат полученных значений 86
3.5. Файлы заголовков 87
3.6. Директивы компилятора 88

3.6.1. Директива условной компиляции 88
3.7. Конструкции программирования....92
3.8. Операторы для организации программных циклов....93
3.8.1. Оператор FOR 93
3.8.2. Оператор WHILE 94
3.8.3. Оператор DO – WHILE....95
3.9. Операторы принятия решения....95
3.9.1. Оператор IF....96
3.9.2. Оператор IF-ELSE 97
3.9.3. Оператор IF-ELSE IF-ELSE 97
3.9.4. Оператор SWITCH....99
3.10. Массивы....100
3.11. Указатели 102
3.12. Структуры....104
3.13. Процесс программирования и отладки микропроцессорной системы....106
3.13.1. Технология создания программного кода 106
3.13.2. Режим отладки BDM....112
3.13.3. Аппаратные и программные средства отладчика P&E от компании PEMICRO 117
3.13.4. Эмуляторы 118
3.13.5. Логические анализаторы....122
3.14. Особенности компилятора и ассемблера 123
3.15. Заключение по главе 3 134
3.16. Что еще почитать? 134
3.17. Вопросы и задания 134

Глава 4. **МИКРОКОНТРОЛЛЕРЫ 68HC12 И HCS12: АРХИТЕКТУРА И ПРОГРАММИРОВАНИЕ**

4.1. Аппаратные средства МК семейства 68HC12 139
4.2. Аппаратные средства МК семейства HCS12 147
4.3. Режимы работы МК семейства 68HC12/HCS12 149
4.3.1. Рабочие режимы....150
4.3.2. Режимы работы отладочной платы M68EVB912B32....151
4.4. Назначение выводов МК 152
4.5. Регистры специальных функций МК....153
4.5.1. Виртуальный адрес блока регистров....156
4.6. Порты ввода/вывода....157
4.6.1. Спецификация портов ввода/вывода 158
4.7. Подсистема памяти МК B32....164

4.7.1. Карта памяти МК B32 166
4.7.2. Изменение адресов в карте памяти МК 166
4.8. Подсистема памяти МК DP256 167
4.9. Состояния сброса и прерывания МК 168
4.9.1. Реакция МК на внешние события 169
4.10. Состояния сброса и прерывания в МК 68HC12 170
4.10.1. Состояние сброса МК 171
4.10.2. Прерывания 173
4.10.3. Вектора исключений 177
4.10.4. Система приоритетов для исключений 180
4.10.5. Регистры подсистемы прерывания 182
4.11. Процесс перехода к подпрограмме прерывания 183
4.12. Оформление подпрограммы прерывания на Си 187
4.13. Система тактирования 189
4.13.1. Система тактирования отладочной платы MC68HC912B32EVB 189
4.14. Подсистема реального времени – модуль таймера 191
4.14.1. Структура модуля таймера 192
4.14.2. Счетчик временной базы 193
4.14.3. Регистры для управления счетчиком временной базы 199
4.14.4. Каналы захвата/сравнения 202
4.14.5. Счетчик событий 225
4.15. Модуль меток реального времени 230
4.16. Модуль таймера ECT в составе МК MC68HC12BE32 и HCS12 233
4.16.1. Небуферированные каналы входного захвата 235
4.16.2. Буферированные каналы входного захвата 235
4.16.3. Особенности счетчиков событий 236
4.16.4. Регистры управления модуля EST 236
4.17. Обмен информацией в последовательном коде: многофункциональный последовательный интерфейс 240
4.17.1. Термины последовательного обмена 243
4.18. Контроллер асинхронного обмена SCI 245
4.18.1. Передатчик контроллера SCI 248
4.18.2. Приемник контроллера SCI 249
4.18.3. Регистры контроллера SCI 250
4.18.4. Алгоритмы программного обслуживания контроллера SCI 257
4.18.5. Пример программирования контроллера SCI 258
4.19. Синхронный последовательный интерфейс SPI 260
4.19.1. Концепция интерфейса SPI 260

4.19.2. Алгоритмы работы контроллера SPI 261
4.19.3. Регистры контроллера SPI 263
4.19.4. Алгоритмы программного обслуживания контроллера SPI 270
4.19.5. Периферийные ИС с интерфейсом SPI....273
4.20. Введение в теорию аналого-цифрового преобразования....274
4.20.1. Частота дискретизации сигнала 274
4.20.2. Представление аналоговой величины в цифровом коде....275
4.20.3. Квантование по уровню и разрешающая способность....277
4.20.4. Скорость потока данных оцифровки 277
4.21. Принцип действия АЦП....279
4.21.1. АЦП последовательного приближения 279
4.22. Подсистема аналого-цифрового преобразования МК 68HC12....282
4.22.1. Структура и порядок функционирования....282
4.22.2. Регистры управления модуля ATD 284
4.22.3. Пример программирования модуля ATD....293
4.22.4. Обслуживание прерываний от модуля ATD 296
4.23. Особенности модуля ATD в составе МК семейства HCS12 297
4.23.1. Выбор разрядности АЦП 297
4.23.2. Представление результата измерения....297
4.23.3. Запуск измерительной последовательности от внешнего сигнала 299
4.23.4. Программируемое число преобразований в измерительной последовательности....299
4.23.5. Увеличение числа аналоговых входов 300
4.23.6. Регистры модуля ATD HCS12 300
4.24. Подсистема широтно-импульсной модуляции 300
4.24.1. Структура модуля PWM....302
4.24.2. Режимы центрированной и фронтовой ШИМ 305
4.24.3. Система тактирования....306
4.24.4. Регистры модуля PWM....308
4.24.5. Примеры программирования модуля PWM....313
4.25. Ограничение энергии потребления....317
4.25.1. Как остановить МК 68HC12....317
4.25.2. Как вывести МК 68HC12 из состояния пониженного энергопотребления....318
4.26. Советы по использованию платы отладки MC68EVB912B32 320
4.27. Заключение по главе 4 323

4.28. Что еще почитать? 323
4.29. Вопросы и задания 323

Глава 5. **ОСНОВЫ СОПРЯЖЕНИЯ МК С УСТРОЙСТВАМИ ВВОДА/ВЫВОДА**

5.1. Электрические характеристики МК 68HC12 328
5.1.1. Нагрузочные характеристики 330
5.1.2. Что произойдет, если Вы должным образом не учтете электрические характеристики периферийных ИС? 331
5.1.3. Входные и выходные характеристики логических элементов 332
5.2. Устройства дискретного ввода: кнопки, переключатели, клавиатуры 334
5.2.1. Кнопки и переключатели 334
5.2.2. DIP переключатели 335
5.2.3. Клавиатуры 336
5.3. Устройства индикации: светодиоды, семисегментные индикаторы, индикаторы логического выхода с тремя состояниями 342
5.3.1. Светодиоды 342
5.3.2. Семисегментные индикаторы 343
5.3.3. Индикаторы для логического выхода с тремя состояниями 344
5.4. Программное обслуживание дискретных входов и выходов 346
5.5. Подавление механического дребезга контактов переключателей 346
5.5.1. Аппаратная защита от механического дребезга контактов 347
5.5.2. Программная защита от механического дребезга контактов 348
5.5.3. Пример программной защиты 349
5.6. Жидкокристаллические индикаторы 352
5.6.1. Краткие сведения о жидкокристаллических индикаторах 352
5.6.2. Сопряжение МК с символьным ЖК индикатором 353
5.6.3. Сопряжение МК с графическим ЖК дисплеем 359
5.7. Управление электрическим двигателем 367
5.7.1. Силовые полупроводниковые ключи 367
5.7.2. Оптоэлектронная потенциальная развязка 368
5.7.3. Инвертор напряжения 370
5.8. Кодовый замок 371
5.8.1. Схема подключения периферийных устройств 373

5.8.2. Программа управления....373
5.9. Интерфейс МК с аналоговыми датчиками....378
5.10. Интерфейс RS-232....380
5.11. Заключение по главе 6....381
5.12. Что еще почитать ?....382
5.13. Вопросы и задания....382

Глава 6. **ДОБРО ПОЖАЛОВАТЬ В РЕАЛЬНЫЙ МИР!**

6.1 Ужасные истории об ошибках проектирования....386
6.1.1. Случай квадратичного генератора....386
6.1.2. Случай таймера для лазерного излучения....388
6.2. Правила обращения с микросхемой 68HC12 и рекомендации по проектированию....391
6.2.1. Рекомендации по обращению со CMOS....392
6.2.2. Рекомендации по проектированию на CMOS....392
6.3. Исследование помех....393
6.3.1. Что такое помехи....393
6.3.2. Электромагнитная совместимость....395
6.3.3. Спецификации системы помех – не будем крепки задним умом!....395
6.3.4. Методы снижения помех....396
6.4. Защитное программирование....399
6.5. Методики испытаний на наличие помех....401
6.5.1. Обнаружение помех....401
6.5.2. Испытание на чувствительность к помехам....401
6.5.3. Испытания на электомагнитную совместимость....402
6.6. Управление энергопотреблением....403
6.6.1. Параметры потребляемой мощности для процессора 68HC12....403
6.6.2. Типы батарей....404
6.6.3. Емкость батарей....404
6.6.4. Стабилизация напряжения....404
6.6.5. Схемы супервизора для микропроцессора....406
6.6.6. Меры энергосбережения....407
6.7. Заключение по главе 6....408
6.8. Что еще прочитать....408
6.9. Вопросы и задания....409

Глава 7. **ПРИМЕРЫ ВСТРОЕННЫХ СИСТЕМ УПРАВЛЕНИЯ**

7.1. Система привода робота, движущегося вдоль стенок лабиринта....412

7.1.1. Описание проекта 412
7.1.2. Системы 68HC12, используемые в проекте 415
7.1.3. Описание некоторых компонентов системы....415
7.1.4. Структура программы и блок-схема алгоритма 417
7.1.5. Программный код 418
7.2. Лазерный проектор 426
7.2.1. Описание проекта 426
7.2.2. Системы 68HC12 используемые в проекте 426
7.2.3. Описание некоторых компонентов системы....426
7.2.4. Аппаратные средства 432
7.2.5. Структура программы и блок-схема алгоритма 432
7.2.6. Программный код 433
7.2.7. Испытания устройства 437
7.2.8. Заключительные испытания системы управления....438
7.3. Цифровой вольтметр 438
7.3.1. Описание проекта 438
7.3.2. Системы 68HC12 используемые в проекте 438
7.3.3. Основы теории: повторный расчет интерфейса АЦП 438
7.3.4. Структура программы и блок-схема алгоритма 439
7.3.5. Программный код 440
7.3.6. Использование вольтметров для измерения неэлектрических величин 444
7.4. Стабилизация скорости вращения двигателя с использованием оптического тахомера 445
7.4.1. Описание проекта 445
7.4.2. Немного теории....448
7.4.3. Анализ 457
7.4.4. Структура программы и блок-схема алгоритма 458
7.4.5. Код 459
7.4.6. Испытания 463
7.5. Парящий робот....463
7.5.1. Описание проекта 463
7.5.2. Системы HCS12 используемые в проекте 464
7.5.3. Теоретическое обсуждение....464
7.5.4. Структура программы и блок-схема алгоритма 465
7.5.5. Программный код 466
7.5.6. Некоторые комментарии....472
7.6. Система защиты компьютера, основанная на нечеткой логике ...472
7.6.1. Описание проекта 472
7.6.2. Использование системы HCS12 473

7.6.3. Основы теории 473
7.6.4. Структура программы и блок-схема алгоритма 473
7.6.5. Описание системы....474
7.6.6. Обсуждение проекта 480
7.6.7. Программный код 481
7.6.8. Некоторые комментарии....490
7.7. Электронная версия игры в «15» 490
7.7.1. Описания проекта....490
7.7.2. Системы HCS12 используемые в проекте 492
7.7.3. Основы теории 492
7.7.4. Схемное решение, структура программы и блок схема алгоритма 492
7.7.5. О компонентах системы 492
7.7.6. Программный код 493
7.7.7. Некоторые комментарии....512
7.8. Приложение: программирования флеш-памяти B32 EVB 512
7.9. Заключение по главе 7 513
7.10. Что еще прочитать ?....514
7.11. Вопросы и задания 515

Глава 8. **ОПЕРАЦИОННЫЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ**

8.1. Образное сравнение: «живая» операционная система в режиме реального времени 518
8.2. Что является целью ОСРВ? 519
8.3. Обзор комцепций 522
8.3.1. Требования к динамическому распределению RAM 522
8.3.2. Динамическое распределение памяти 525
8.3.3. Структуры данных 526
8.4. Основные понятия....545
8.4.1. Что такое задача? 545
8.4.2. Управление задачами 547
8.4.3. Компоненты многозадачных систем 558
8.5. Типы операционных систем реального времени....560
8.5.1. Системы с циклическим опросом....560
8.5.2. Циклический опрос с прерываниями....561
8.5.3. Круговые системы....562
8.5.4. Смешанные системы 562
8.5.5. Системы с управлением по прерыванию 563
8.5.6. Кооперативная многозадачность 564

8.5.7. Многозадачные систем
с преимущественным приоритетом 564
8.6. Проблемы ОСРВ 565
8.6.1. Конкуренция 565
8.6.2. Повторная входимость 567
8.6.3. Межзадачные связи 567
8.6.4. Безопасность, проверка и безотказная работа 568
8.6.5. Главный вопрос 568
8.7. Выполнение операционной системы
в режиме реального времени 569
8.8. Базовое примечание: контроллер стереоусилителя
и система циклического опроса 569
8.8.1. Краткий обзор проекта 569
8.8.2. Пример кода 570
8.8.3. Испытание контроллера усилителя 583
8.9. Другая прикладная программа: цикл опроса с прерываниями 584
8.10. Сложное прикладное устройство: имитатор ОСРВ 585
8.10.1. Краткий обзор проекта 585
8.10.2. Типовой код 587
8.11. Заключение по главе 8 591
8.12. Что еще почитать? 591
8.13. Вопросы и задания 592

Глава 9. **РАСПРЕДЕЛЕННЫЕ СЕТИ С ИНТЕРФЕЙСОМ MSCAN**

9.1. Компьютерные сети 596
9.2. Промышленные сети 597
9.3. Сети с протоколом CAN 598
9.3.1. Протокол CAN 598
9.3.2. Модуль контроллера последовательного
обмена msCAN12 602
9.3.3. Проблемы синхронизации 620
9.3.4. Конфигурирование модуля msCAN12 для работы в сети 623
9.4. Различия между контроллерами msCAN
в составе 68HC12 и HCS12 626
9.5. Пример программирования контроллера msCAN 627
9.6. Контроллер последовательного обмена BDLC 633
9.7. Заключение по главе 9 633
9.8. Что еще почитать? 634
9.9. Вопросы и задания 634

ПРЕДИСЛОВИЕ

Вначале 2002 года наша первая книга «Микроконтроллеры семейства 68HC12: Теория и применение» была издано издательством Prentice Hall. Нашими целями были: представить основы программирования на ассемблере; проиллюстрировать работу отдельных блоков в составе микроконтроллера и представить методы сопряжения различных внешних устройств с микроконтроллерами. В качестве примера мы использовали автономный мобильный робот для иллюстрации совместной работы микроконтроллера во встраиваемой системе.

Наша вторая книга, посвященная встраиваемым микропроцессорным системам «Разработка и применение встраиваемых систем на основе микроконтроллеров семейства 68HC12 и HCS12», охватывает проблемы не раскрытые в первой книге. Нашей целью в ней было разработать учебное пособие по проектированию встраиваемых систем. Мы постарались провести читателя от основ системного программирования через применения операционных систем реального времени к решению задач распределенного управления. Вместо того чтобы «нырнуть на глубину в бассейн» мы начали с обучения концепциям системного проектирования и программирования на языке С. Затем мы двинулись к обсуждению специфического аппаратного обеспечения, реализованного на кристаллах микроконтроллеров семейств 68HC12/HCS12. В начале этих глав мы придерживались идеологии «ходьбы перед бегом». Мы предполагали, что читатель имеет фундаментальные но базовые знания по организации программно–аппаратного обеспечения микроконтроллеров. Мы считаем это правильным подходом поскольку целевой аидиторией для книги являются студенты учебных заведений вовлеченные во второй цикл обучения разработки встраиваемых систем. Темы обучения в начальных главах книги могут быть пропущены инженерами с опытом. Однако мы получили много пожеланий от таких инженеров на включении этого материала в книгу.

Имея такой задел, мы затем перешли к рассмотрению большого количества примеров втраиваемых систем. Примеры были выбраны таким образом, чтобы читатель познакомился с разнообразными примерами сопряжения различных устройств ввода–вывода с системой. В завершении книги рассматриваются такие разделы как операционные системы реального времени (RTOS) и мультипроцессорные системыю Мы коснулись этих сложных тем только после того как рассмотрели основы микропроцессорных и встраиваемых систем.

Мы имели несколько целей при написании этой книги.

1. Мы хотели, чтобы читатель приобрел навыки программирования как на языке ассемблера так и на языке С при разработке встраиваемых систем управления на основе микроконтроллеров.;

2. Изложить методические аспекты проектирования встраиваемых систем;
3. Представить функциональное аппаратное обеспечение микроконтроллеров;
4. Раскрыть методы сопряжения с микроконтроллерами различных периферийных устройств при создании встраиваемых систем;
5. Рассмотреть технологии по решению узких мест при разработке встраиваемых систем, связанные с применением применением операционных систем реального времени, а также многопроцессорных систем.

Все содержание книги построено с учетом этих целей. Наша мотивация по написанию этой книги исходила из того, что на моменте ее подготовки не существовало всестороннего учебника для студентов по семействам 68HC12/HCS12, рассматривающего процесс разработки и программирования встраиваемых систем на микроконтроллерах.

Мы предприняли попытку практической ориентации книги с сильным упором на обучение и многочисленными практическими примерами. Основанные на реальных применениях, эти примеры сфокусированы на приобретение навыков по разработке встраиваемых систем, методов синхронизации и подавлению шумов, а также способов отыскания неисправностей. Книга представляет обзор языка программирования Си, методов структурного программирования, микроконтроллеров семейств 68HC12/HCS12, детальное обсуждение проблем RTOS, многопроцессорных систем и иллюстрацию концепций разработки встраиваемых систем.

Вначале книги мы представляем читателю концепции структурной разработки систем. Используя подход функционального разбиения системы сверху вниз, студенты будут в состоянии понять любые проблемы связанные со сложностью структуры встраиваемых систем. Мы коснулись некоторых принципов системного подхода к разработке, описанного Meilir Page–Jones в его классической книге «Practical Guide to Structured Systems Design». Эти методы работают в равной степени хорошо при разработке программного, аппаратного и прораммно–аппаратного обеспечения встраиваемых систем. Однажды их представив, мы их активно используем в дальнейшем во всей книге.

Структура книги

В каждой главе мы подробно представляем последовательность и значение описываемых разделов. Каждая глава начинается с описания целей поставленных нами при изложении материала. Это позволяет читателю ясно представлять задачу при чтении главы. После представления основных концепций главы, на конкретном примере рассматриваются применение ключевых понятий и технологий.

В первой главе мы представили понятие встраиваемых систем и специфические проблемы связанные с их разработкой и применением. Глава 2 описывает преимущества программирования на языках высокого уровня – High Level Language (HLL). Мы сбалансировали обсуждение методов программирования на языке ассемблер и HLL и показали, что программы для встраиваемых систем могут содержать оба подхода. Мы обсудили ключевые концепции структурного программирования, позволяющие разбить большие проекты на более понятные и легко реализуемые части. Затем мы применили эти понятия на этапах разработки, реализации и тестирования систем. Мы дали почувствовать читателю некую комфортность использования такого подхода на примерах простых систем прежде чем переходить к более сложным случаям.

В главе 3 мы рассматриваем процесс программной/компиляции/ассемблирования анализируя принципы программирования на языке С. В завершении главы рассматриваются методы и средства программирования и отладки программ. При рассмотрении проблем программирования мы намерено ушли от любых специфических особенностей компиляторов. На сегодняшний день существуют очень много доступных компиляторов для семейств 68HC12/HSC12. В четвертой главе мы описываем структуру семейств 68HC12/HSC12 и их отдельных представителей. В дальнейшем мы иллюстрируем их применение в реальных системах управления.

В главе 5 мы изучаем методы сопряжения внешних устройств с микроконтроллерами .Анализ начинается с простых примеров подключения переключателей и индикаторов и заканчивается такими более сложными устройствами как жидко–кристаллические дисплеи. Шестая глава развивает методы сопряжения микроконтроллеров с устройствами реального мира. В ней разделяются теоретические проблемы построения встраиваемых систем от проблем реально работающих систем. Каждая проблема вначале определяется, а затем подкрепляется методами ее практической реализации.

В главе 7 мы помещаем микроконтроллеры 68HC12/HSC12 в реальные системы. В каждом примере мы обеспечиваем сквозное описание проекта, алгоритм работы и код, необходимый для реализации системы. Мы скрупулезно подошли к подбору примеров так, чтобы все они были реализуемы на микроконтроллерах семейства 68HC12/HSC12. В восьмой главе мы рассмотрели проблемы применения операционных систем реального времени. Мы начали с определений RTOS, а затем перешли к обсуждению возможностей их реализации. В дальнейшем мы рассмотрели проблемы, связанные с реализацией RTOS. Мы предполагали, что читатель не имеет практического опыта работы с системами подобной сложности.

Глава 9 рассматривает распределенные системы. Такие системы содержат более одного микропроцессора в своей структуре. Мы представили методы и подходы, позволяющие сопрягать их в систему, используя встроенный CAN контроллер в семейства 68HC12/HSC12.

В дополнению к содержанию книги мы подготовили и поддерживаем справочный веб сайт www.prenhall.com/pack. Он содержит справочную информацию по семействам 68HC12/HSC12, файлы программ на С, и программно–аппаратные средства поддержки микроконтроллеров семейства 68HC12/HSC12. Для преподавателей этот веб сайт также содержит дополнительный материал включая лекционные слайды в Power Point и рекомендации как заказать информацию по всем решениям задач, представленных во всех домашних заданиях в каждой главе.

Учебные системы

Для иллюстрации системных принципов в главах с 1 по 9 мы рассматривали многочисленные примеры. Примеры были разработаны для двух учебных систем: отладочной платы M68HC912B32EVB (B32EVB) и для MC9S12DP256 или DP256. Мы выбрали отладочную плату B32EVB ввиду ее широкого распространения, разумной цены и что наиболее важно ее многими полезными функциями. EVB имеет интерфейс RS–232, работает от одного источника питания, имеет легкий доступ к основным контрольным точкам через четыре группы разъемов и монтажное поле для размещения дополнительной схемотехники при анализе систем. EVB также

имеет хороший набор функций памяти, включающий в себя 32К байт электрически перепрограммируемой флэш памяти программ (EPROM), 1К байт ОЗУ и 768 байт побайтно стираемой EEPROM для записи данных. Во флэш памяти расположен резидентный монитор/отладчик программ D–Bug12. Мы опишем в деталях все отмеченные свойства в гл. 4. B32 является отличным учебным средством, но оно может быть также успешно использована для реализации прототипов отлаживаемых систем.

Читатели, которые не намерены использовать B32 EVB, могут быть уверены, что большинство из рассмотренных в книге примеров могут быть реализованы на других вариантах отладочных средств семейств 68HC12/HSC12.

Поскольку базисные концепции и функциональные блоки различных микроконтроллеров практически идентичны друг другу, то полученные знания семейств 68HC12/HSC12 могут быть естественным образом применены и для других микроконтроллеров. В гл.7 и 9 мы используем микроконтроллер MC9S12DP256. HCS12 микроконтроллер имеет 256 К байт флэш память, несколько каналов msCAN интерфейсов с соответствующими контроллерами. Он также имеет большой объем ОЗУ. В распоряжении разработчиков имеются различные типы отладочных плат.

Целевая аудитория

Основной аудиторией для книги являются студенты университетов, изучающих курс вычислительные микропроцессорные системы. Поскольку все ABET (Acredit tion Board for Engineering and Tehnology, Inc) требуют наличия такого курса в своих программах, мы надеемся, что преподаватели этих дисциплин будут активно использовать данную книгу в своей практике. Мы ожидаем также, что студенты первого года обучения языков программирования найдут эту книгу для себя также полезной. Владение темой языков программирование позволит студентам легко разобраться с приведенными в книге примерами. В идеале студенты будут имеет полный курс введения в микроконтроллеры. Однако ввиду экспериментальной направленности книги студенты должны будут самостоятельно заполнить пробелы в знаниях там где это будет необходимо.

Основной упор в книге делается на второй семестр курса микроконтроллеры/микропроцессоры программы электротехнического и вычислительного цикла дисциплин. Разные учебные заведения предлагают микропроцессорный цикл на различных этапах обучения студентов. Наши студенты слушают базовый курс по цифровой технике на первом году обучения. После этого они изучают первую часть курса микропроцессорных систем. В завершении, вторую часть микропроцессорного курса они слушают на последних годах обучения. Мы надеемся, что данная книга будет востребована студентами на втором этапе обучения после освоения первого этапа изучения микроконтроллеров.

Мы написали эту книгу для применения ее в качестве учебного пособия для учебных заведений, читающих цикл микропроцессорной техники. В тоже время мы надеемся, что практическая направленность материала будет полезна инженерам для самостоятельного изучения раздела. Мы уверены, что знания о встраиваемых системах будут все больше и больше востребованы для все более широкой аудитории студентов электротехнических и электронных специальностей. Мы живем в обществе

где все больше инженерных проблем решается с помощью встраиваемых систем. Мы предвидим, что скорость внедрения встраиваемых систем будет увеличиваться с ростом требований к интеллектуальности локальных систем.

Благодарности

Эта книга является совокупным трудом многих специалистов. Конечно же ни одна хорошая книга не может появится без выдающегося издателя и его команды. Мы благодарны Tom Robbins и Alice Dworkin из Prentice Hall за их веру в проект. Было большим удовольствием работать с Kevin Bradley и его сотрудниками из Sunflower Publishing Services. Мы благодарны им за их отличную редакторскую работу. Мы высоко оцениваем отзыв и обратную связь от Barry Mullins из Air Force Institute of Technology. В результате всех этих усилий содержание книги значительно улучшилось. Мы также высоко ценим дельные замечания и обратную связь, полученную по окончательной версии книги от Jerry Hamman из University of Wyoming; John Reece из Mercer College из Macom, Georgia и William Stapleton из University of Alabama. Мы также очень благодарны Karen Bosco из Motorola за ее помощь в получении разрешении от Motorola использовать рисунки в книге.

Мы признательны руководству нашего факультета за их поддержку. Colonel Alan Klayton (USAF Academy) постоянно поддерживал нашу работу и и осуществлял многостороннюю помощь. John Steadman (ранее работавший в University of Wyoming, а в настоящее время декан электротехнического факультета в University of South Alabama), активно воодушевлял нас на написание этой книги. Мы благодарны большому количеству студентов вовлеченных в учебный процесс по курсам микропроцессорной технике в отмеченных выше университетах. Их обратная связь аказала нам большую помощь в изложении материала книги. Abbie Wells, Scott Lewis, Joel Perlin, Carrie Hernanddez, Ted Dibble, Tom Schei, Charles Straley, Pamela Beavis и Austin Griffith из университета Wyoming написали ряд программ в примерах и оказали значительную помощь в переходе от семейства от семейства 68HC11 к семейству HC12. В дополнении к этому мы хотели бы поблагодарить многих наших коллег из Air Force Academy и University of Wyoming.

Ряд примеров, представленных в книге базировались на нашем опыте работы на факультетах электротехники в отмеченных ранее университетах. Несмотря на то, что мы старались исправить все допущенные ранее ошибки, книга может содержать еще не выявленные, что может еще сказаться на работе программ.

Я хотел бы еще отметить Glarence Zarn, очень дорогого друга семьи, из–за влияния которого я стал инженером. Когда я был еще ребенком, моя семья проводила много отпусков с семьей Zarns. Меня брали в офис Clarence, полный книг, справочников, схем и плакатов, иллюстрирующих многообразный мир техники. В то время Clarence работал инженером и был вицепрезидентом компании Pentzien Corporation of Omaha, Nebraska в течение долгого времени. Недавно он подарил мне свою логарифмическую линейку (1940), которой он пользовался долгие годы. Я буду хранить ее всегда. Он повторял, что инженерная работа была и всегда останется востребованной. Большое спасибо моим родителям. Спасибо моим бабушкам Eleonore и Jackie, а также дедушке Frank за постоянную веру в меня. Спасибо Young Shin и Rana и Sung Bock и Chong Kon за мотивацию и постоянную поддержку. Осо-

бая благодарность моему отцу за воодушевления меня на этот труд, в то время когда он сам боролся с раком. В заключении хочется отметить, что эта работа не могла бы осуществиться без поддержки членов нашей семьи: Cindy, Heidi, Hearther, Jon R., Christine, Jon B., Andrew и Graham. Примите нашу благодарностью

Steven F. Barrett
Daniel J. Pack

Глава 1

ПЕРВОЕ ЗНАКОМСТВО СО ВСТРАИВАЕМЫМИ СИСТЕМАМИ

ПОСЛЕ ИЗУЧЕНИЯ ГЛАВЫ ВЫ СМОЖЕТЕ:

ЦЕЛИ

- Дать определение термину «вычислитель»;
- Перечислить основные блоки вычислителя. Описать функции этих блоков;
- Объяснить значение терминов: встраиваемая система, микроконтроллер, компьютер общего применения;
- Привести примеры встраиваемых микропроцессорных систем управления;
- Перечислить основные проблемы, которые возникают в процессе разработки встраиваемых систем;
- Пояснить термин «работа в реальном времени» в контексте обсуждения встраиваемых систем;
- Описать основные функциональные блоки микроконтроллеров семейства 68HC12 и семейства HCS12.
- Перечислить основные различия между микроконтроллерами семейства 68HC12 и семейства HCS12.

1.1. Что такое встраиваемая система? 22
1.2. Особенности встраиваемых систем 25
1.3. Введение в микроконтроллеры семейства 68HC12 и HCS12 29
1.4. Микроконтроллеры HCS12 33
1.5. Заключение по главе 1 36
1.6. Вопросы и задания 36

В этой главе мы познакомим Вас с теми проблемами, которые возникают при проектировании, реализации и тестировании встраиваемых микропроцессорных систем. Мы начнем наше рассмотрение с достаточно общих понятий, однако в разделе 1.3 произойдет Ваше первое знакомство ч системами на микроконтроллерах семейств 68HC12 и HCS12.

1.1. Что такое встраиваемая система?

Любая механическая или электрическая система, которая имеет в своем составе устройство управления, выполненное на основе вычислителя, называется встраиваемой системой (Embedded System). Перед тем, как мы продолжим, следует дать определение термину «вычислитель». Все вычислители обязательно состоят из следующих функциональных блоков: центрального процессора (ЦП), запоминающего устройства (ЗУ) устройств ввода/вывода (УВВ) и межмодульных магистралей. Центральный процессор содержит в себе арифметико–логическое устройство (АЛУ), устройство управления и некоторое количество регистров. АЛУ выполняет операции над данными, которые представлены в цифровом коде. Типовыми для АЛУ являются следующие арифметические и логические операции: сложение, вычитание, умножение, деление, логические операции И, ИЛИ. ИСКЛЮЧАЮЩЕЕ ИЛИ, НЕ (инверсия).

Вычислитель, центральный процессор которого выполнен на основе одной большой интегральной схемы (ИС), именуемой микропроцессором (МП), называют микро_ЭВМ. Самый известный пример микро_ЭВМ – это персональный компьютер (ПК). Интегральные схемы, которые объединяют на одном полупроводниковом кристалле все основные функциональные блоки вычислителя, т.е. центральный процессор, запоминающее устройство, устройства для ввода и вывода информации и межмодульные магистрали, называют микроконтроллером (МК).

Блок памяти вычислителя или микроконтроллера хранит коды программы и данные, которые необходимы для выполнения вычислений. По способу организации блока памяти различают вычислительные системы с архитектурой фон Неймана или с Гарвардской архитектурой. Архитектура фон Неймана предоставляет возможность хранения в одних и тех же ячейках памяти, как кодов программы, так и данных. При Гарвардской архитектуре для программы и для данных выделены отдельные области памяти. Гарвардскую архитектуру можно встретить в мощных вычис-

лительных системах, для которых характерно наличие области кэш памяти. В целом архитектура современных компьютеров и микроконтроллеров представляет собой некоторую оригинальную структуру на основе двух этих архитектур.

Устройства ввода/вывода обеспечивают связь центрального процессора с внешним миром. Обычно устройства ввода используются для приема в вычислительную систему информации с датчиков, фиксирующих состояние управляемого объекта, а также для приема команд управления от оператора. Устройства вывода предназначаются для выдачи команд и сигналов управления объектом, а также для отображения информации о текущем состоянии объекта. Система межмодульных магистралей обеспечивает соединение трех перечисленных блоков: центрального процессора, памяти и устройств ввода/вывода, создавая на их основе вычислительную систему. По магистралям передаются коды программы, данные и сигналы управления. Каждому виду сигналов соответствует одноименная магистраль системной шины компьютера: магистраль адреса, магистраль данных и магистраль управления.

Вернемся в встраиваемым системам. Напомним, что встраиваемой системой называется система, управляемая вычислителем, являющимся неотъемлемой составной частью этой системы. Следует различать вычислители общего применения, которые называют компьютерами, и вычислители встраиваемых систем. Компьютер общего применения, например ноутбук, может выполнять множество программ, начиная от текстового редактора, заканчивая сложными расчетными задачами моделирования механических конструкций и электронных схем. Вычислитель встраиваемой системы, который может быть реализован на процессоре с не меньшей вычислительной производительностью, чем ноутбук, выполняет только специальную программу управления. При этом вычислителя встраиваемой системы может иметь дополнительные аппаратные средства, которые будут отличать его от компьютера общего применения.

Вычислители встраиваемых систем не являются некоторым дополнением к знакомому Вам миру персональных компьютеров. Вы удивитесь, если узнаете, что ежегодно объем продаж встраиваемых систем значительно превышает объем продаж персональных компьютеров. Так в 2000 году было продано 150 миллионов компьютеров общего применения, в то время, как объем продаж встраиваемых систем в том же году исчислялся 8000 миллионами. Большинство современных встраиваемых систем выполняется на основе микроконтроллеров (МК) – вычислителей, все функциональные блоки которых объединены на одном полупроводниковом кристалле. Конструктивно МК представляет собой одну интегральную схему (ИС) большой степени интеграции.

В настоящее время многие производители полупроводниковых компонентов, такие как Intel, Microchip, Hitachi, NEC, Atmel, Texas Instruments и др., выпускают микроконтроллеры различной сложности. Относительно простые МК находят применение в бытовой технике и игрушках. Наиболее сложные высокопроизводительные МК используются в коммуникационном оборудовании, для управления самолетами и военной техникой. В этой книге мы предлагаем Вам изучить два типа МК: семейства 68HC12 и HCS12 компании Motorola.

Встраиваемые системы на основе МК окружают Вас со всех сторон. Вы не можете прожить без них и часа. Например, Ваш будильник, телефон и карманный компьютер – все это встраиваемые системы на микроконтроллерах. Ваш дом буквально наводнен встраиваемыми системами: кофеварка, телевизор с дистанционным пультом управления, стиральная машина, кухонный комбайн, электрическая духовка и СВЧ печь, холодильник, система охранной сигнализации, музыкальный центр и DVD проигрыватель...Мы перечислили отнюдь не все домашние устройства.

А Ваш автомобиль? Он ежедневно «возит с собой» от 10 до 50 микроконтроллеров. Встроенные МК делают агрегаты Вашего автомобиля более безопасными, экономичными, обеспечивающими легкость управления и комфортабельность движения. Микроконтроллеры используются в системе впрыска топлива и в системе торможения, для управления трансмиссией и рулевой колонкой, в устройствах приборной панели, маршрутного компьютера, центрального замка и аудио системы. Микроконтроллеры нагревают или охлаждают сиденья Вашего автомобиля, поворачивают зеркала, вращают фары, управляют движением дворников и стекол дверей. В некоторых моделях они могут даже измерить давление в шинах, показать маршрут до цели назначения, определить усталость водителя. Неправда ли, Ваше перемещение на автомобиле не было бы привлекательным без всех этих встроенных систем, ставших уже привычными?

А теперь обратимся к тем областям нашего существования, в которых встраиваемые микропроцессорные системы играют ключевую роль. Технические и общественные системы, перечисленные на рис. 1.1, просто не могли бы существовать без разнообразных встраиваемых систем. ак наша военная безопасность и система коммуникаций для управления государством основываются на множестве высокопроизводительных встраиваемых систем. На борту орбитальных космических станций и спутников считают и управляют встраиваемые системы. Любой современный станок и измерительный прибор – это тоже встраиваемая система. Большинство сложных медицинских диагностических комплексов использует для обработки результа-

Рис. 1.1. *Области применения встраиваемых систем*

тов встраиваемые системы. Совершенствование узлов современного автомобиля и других транспортных средств также немыслимо без встраиваемых систем. И наконец, бытовая техника и устройства домашнего развлечения с мультимедийными технологиями – все это встраиваемые системы.

1.2. Особенности встраиваемых систем

Встраиваемые системы существенно отличаются от компьютеров общего применения. В данном параграфе мы обсудим специфические проблемы, которые должны быть решены разработчиком встраиваемой системы на начальном этапе проектирования.

1.2.1. Работа в реальном времени

Когда мы говорим, что встраиваемая система должна работать в реальном масштабе времени, мы подразумеваем, что система должна производить определенные вычисления за строго определенные временные интервалы. Если система не может произвести необходимые вычисления за отведенный временной интервал, то в лучшем случае объект ее управления будет работать с низкими техническими характеристиками, а в худшем случае будет создана аварийная ситуация. Используя термин «вычисления в реальном времени», мы имеем в виду, что интервал времени, предоставляемый для этих вычислений, ограничен. При этом его численное значение определяется конкретной задачей и может существенно различаться для разных систем. Например, система антиблокировки колес автомобиля должна опросить датчики состояния каждого из четырех колес (колесо скользит или катится) и выработать необходимые сигналы для приводов тормозов в течение нескольких миллисекунд. О такой задаче мы говорим, что она исполняется в реальном времени. Другой пример – система GPS навигации автомобиля, которая должна обновлять карту на дисплее в кабине водителя за несколько секунд. Это тоже будет система реального времени. Однако вычислительную систему, которая рассчитывает оптимальные коэффициенты сложного цифрового фильтра в течение трех часов, мы не называем системой реального времени, поскольку время ее исполнения важно, но не критично для пользователя.

Познакомившись с терминологией, давайте обсудим, какой должна быть встраиваемая система для того, чтобы успешно работать в реальном времени. Во–первых, система должна быть разработана таким образом, чтобы необходимый цикл вычислений укладывался в отведенный временной интервал. Для этого необходимо выбрать соответствующую вычислительную производительность микроконтроллера, разработать эффективный по быстродействию алгоритм, разработать схемы интерфейсов с минимально возможными задержками в передаче сигналов. Во–вторых, встраиваемая система должна обладать устойчивостью по отношению к внешним данным. Допустим, для формирования результата система должна получать данные извне. А эти данные не пришли вовремя. Тогда система не может выдать необходимый результат в требуемый момент времени, однако она не должна «зависнуть». Она должна продолжить поставлять результаты в реальном времени, но в ином, возможно сокращенном виде.

В противоположность системам реального времени компьютеры общего назначения не имеют жесткого ограничения по времени выполнения программы. Долгое ожидание завершения расчетов может расстроить пользователя, но не приведет к заметным негативным последствиям. А вот если встроенная в медицинское оборудование система не выполнит задачу за отведенный для нее срок, то это может закончиться в некоторых случаях и смертельным исходом. Поэтому организация работы встраиваемых систем в реальном времени является одной из основных проблем проектирования.

1.2.2. Миниатюризация размеров и процесс тестирования

Многие современные системы должны встраиваться в достаточно миниатюрные устройства, такие как мобильный телефон, пульт управления телевизором, датчик расхода воды и т. д. Очень часто геометрия печатной платы системы определяется корпусом того устройства, для которого она предназначается. Поэтому миниатюризация исполнения – одна из проблем разработчика современных встраиваемых систем.

Другая важная проблема – учет на начальной стадии разработки способов тестирования готового изделия, как на этапе разработки, так и на этапе производства. Большинство встраиваемых систем должны иметь внутренние тестовые программы, которые позволяют быстро и с большой степенью достоверности убедиться в работоспособности программы управления.

1.2.3. Минимизация энергии потребления

Разработчики компьютеров общего назначения (за исключением ноутбуков) уделяют значительно меньше внимания вопросам энергопотребления устройства, нежели разработчики встраиваемых систем. Дело в том, что, во–первых, персональные компьютеры питаются от централизованной сети, которая не накладывает существенных ограничений на энергию потребления, и, во–вторых, объем корпуса персонального компьютера достаточно велик для размещения в нем устройства принудительного охлаждения. В противоположность компьютерам общего назначения, современные встраиваемые системы должны работать в условиях резкого ограничения потребляемой энергии, поскольку число встраиваемых систем с автономным питанием непрерывно возрастает. К тому же пользователи предъявляют все большие требования к миниатюризации систем. Вспомните современный мобильный телефон, карманный электронный органайзер, CD–плеер.

Для ограничения энергии потребления разработчики используют разные решения. Одним из них является снижение частоты тактирования МК. Однако такая мера имеет ограничение, поскольку для любой задачи реального времени имеется ограничение снизу по вычислительной производительности. Другим решением (или дополнительным к первому) является временное отключение питания тех периферийных модулей МК, которые в данный момент исполнения программы не используются. Аппаратные средства современных МК предоставляют такую возможность. Последний способ требует особого внимания разработчика, поскольку отключение какого–либо модуля в составе системы может привести к изменению электрических характеристик ее входов и выходов, которое не должно сказаться на работоспособности системы в целом.

1.2.4. Интерфейс пользователя и интерфейс сопряжения с объектом

Любая встраиваемая система должна взаимодействовать с пользователем или с окружающей средой. Например, перемещающийся в пространстве робот (рис. 1.2) должен с помощью инфракрасных датчиков обнаруживать препятствия и обходить их. Микроволновая печь должна взаимодействовать с человеком посредством кнопок режимов, установленных на передней панели прибора. А система охранной сигнализации должна взаимодействовать как сдатчиками сохранности помещения, так и с органами управления человеком. Подобные примеры могут быть продолжены. И на их основе можно сделать вывод, что для разработчика встраиваемых систем вопросы выработки решений по взаимодействию с человеком и с объектом управления являются чрезвычайно важной задачей. Причем возможные решения лежат на стыке выбора типа датчиков (включая принцип действия датчика), дизайн–проекта, конструктивного исполнения, аппаратного решения электронных блоков и, наконец, алгоритмов обработки информации.

1.2.5. Многозадачность

Большинство встраиваемых систем должно обслуживать в реальном времени сразу несколько внешних устройств. Причем периоды повторения алгоритмов вычисления в реальном времени для каждого из устройств различаются. При разработке таких систем разработчик стоит перед дилемой, использовать для решения задачи один высокоскоростной МК, или сделать мультипроцессорную систему, в которой для каждой задачи будет использован собственный микропроцессор или микроконтроллер.

1.2.6. Минимизация стоимости

Большое количество встраиваемых систем предназначено для управления недорогими устройствами массового спроса, такими как СВЧ печь, мобильный телефон и т.п. Успех реализации таких устройств будет определяться их конечной стоимостью, что накладывает жесткие ограничения на стоимость встраиваемой системы. Каждая встраиваемая система имеет множество возможных решений, как на уровне способа реализации (микроконтроллер или программируемая логическая матрица, вариации интерфейсных схем к тому и другому решениям), так и на уровне выбора конкретной элементной базы. Поэтому выбор правильной стратегии проектирования с целью минимизации стоимости – одна из основных проблем проектирования встраиваемой системы.

1.2.7. Ограничение объема памяти

Если Вы достаточно грамотный пользователь персонального компьютера, то хорошо знакомы с постоянным увеличением объема памяти ПК, которое не сопровождается пропорциональным увеличением ее стоимости. Поэтому программисты для ПК со-

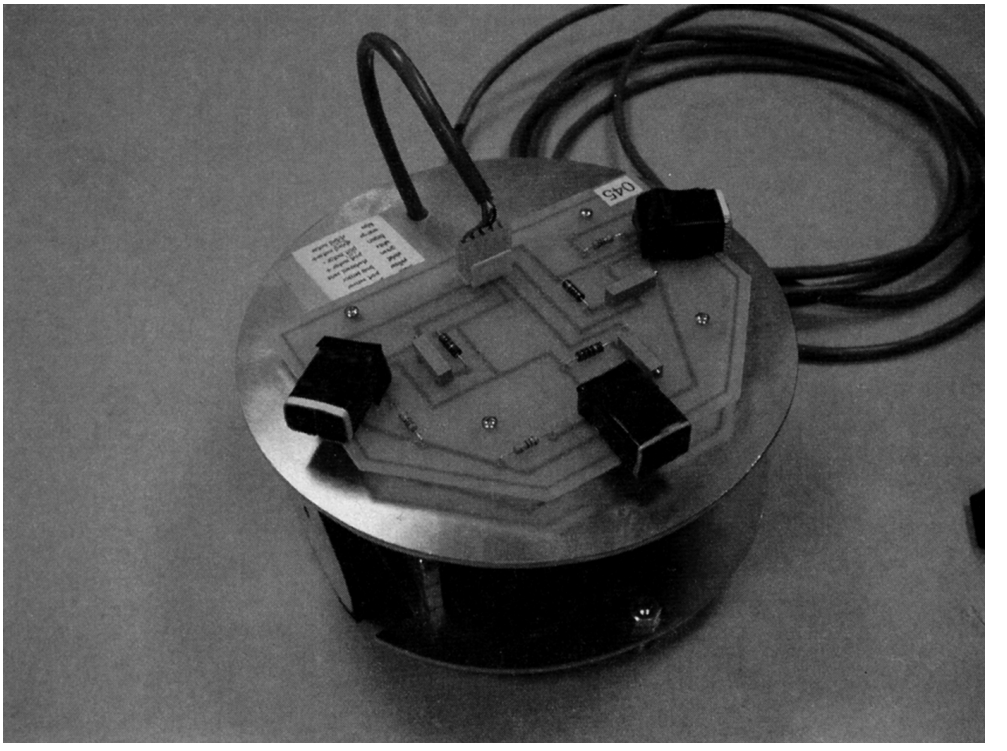


Рис. 1.2. *Робот, способный двигаться сквозь лабиринт*

вершенствуют свои продукты, в том числе, используя без ограничения увеличение объема памяти программ. Встраиваемые системы не предоставляют разработчику такой возможности, поскольку объем резидентной памяти МК оказывает существенное влияние на его стоимость. Современная элементная база позволяет выполнить мобильный телефон с несколькими Гб внутренней памяти, однако какое количество покупателей пожелает купить достаточно дорогое устройство? Поэтому разработка решений с минимизацией затрат памяти – одно из направлений совершенствования встраиваемых систем.

1.2.8. Программно-аппаратный дуализм

Большое количество встраиваемых систем могут быть реализованы как на МК с соответствующей управляющей программой, так и на основе высокоинтегрированной жесткой логики, например, на программируемых логических ИС. Первое решение обладает большей гибкостью, поскольку управляющая программа может быть многократно доработана без изменения аппаратного решения устройства. Второе решение обязательно будет более быстродействующим по сравнению с первым. Возможны и комбинированные варианты решения, при которых часть функций будет возложена на МК, а часть – на устройства жесткой логики. Выбор способа реализации остается за разработчиком.

1.3. Введение в микроконтроллеры семейства 68HC12 и HCS12

В предыдущем параграфе мы обсудили общие проблемы, связанные с разработкой и эксплуатацией встраиваемых микропроцессорных систем. В своем рассмотрении мы пока не касались той элементной базы, на основе которой выполняются встраиваемые системы, т.е. микроконтроллеров. Поскольку наша книга посвящена встраиваемым системам на микроконтроллерах семейства 68HC12/HCS12 компании Motorola/Freescale Semiconductor, то перейдем к знакомству с этой элементной базой.

Семейство микроконтроллеров 68HC12/HCS12 относится к группе 16–разрядных МК. Процессорное ядро 68HC12 унаследовало свою программно–логическую модель и систему команд от широко известного 8–разрядного процессорного ядра HC11. Начало семейству 68HC12 было положено в 1996 году выпуском двух базовых моделей: MC68HC12A4 и MC68HC912B32. Микроконтроллер MC68HC12A4 был предназначен для работы в расширенном режиме, т.е. с внешней памятью программ. Микроконтроллер MC68HC912B32 уже имел на кристалле многократно программируемое пользователем энергонезависимое запоминающее устройство, выполненное по технологии Flash. В 2002 году компанией Motorola/Freescale Semiconductor было предложено новое семейство HCS12, которое предназначалось для замены МК семейства 68HC12 на более высокопроизводительные, но полностью программно совместимые модели. На протяжении этой книги мы будем использовать в качестве базового микроконтроллер MC68HC912B32. Это простой МК низкой стоимости, доступный как для обучения, так и для относительно несложных разработок. Однако большая часть сведений, которые Вы почерпнете из данной книги, может быть легко распространена и на МК HCS12.

На основе базового МК MC68HC912B32 производителем был создан целый ряд моделей: MC68HC12BE32, MC68HC912BC32 и MC68HC12BC32. Основное отличие этих моделей друг от друга состоит в объеме размещенной на кристалле памяти программ и в наличии или отсутствии контроллера CAN сети.

На рис. 1.3 представлена структурная схема сразу двух МК: MC68HC912B32 и MC68HC12BE32. На рис. 1.4 дана цоколевка корпуса для этих микроконтроллеров. Далее на рис. 1.5 показана структура микроконтроллера MC68HC912BC32, на рис. 1.6 – цоколевка корпуса для него. В таблице рис. 1.7 перечислены основные функциональные блоки каждой из перечисленных моделей МК. Эта же таблица позволяет легко определить функциональные различия между рассматриваемыми МК.

Микроконтроллеры семейства 68HC12 – это 16–разрядные МК, что означает, что центральный процессор может выполнять операции над 16–разрядными данными, а также то, что данные передаются внутри МК по 16–разрядной магистрали данных. МК 68HC12 имеют также 16–разрядную магистраль адреса, что позволяет им адресовать 65 536 ячеек памяти. Максимальная частота системной шины МК семейства 68HC12 равна 8 МГц, что обеспечивать значительное возрастание вычислительной производительности по отношению к предшественнику – МК HC11.

Система команд 68HC12 основана на системе команд HC11, однако число способов адресации и число выполняемых действий значительно расширены. Система команд 68HC12 включает 208 инструкций, в том числе пять команд деления с разряд-

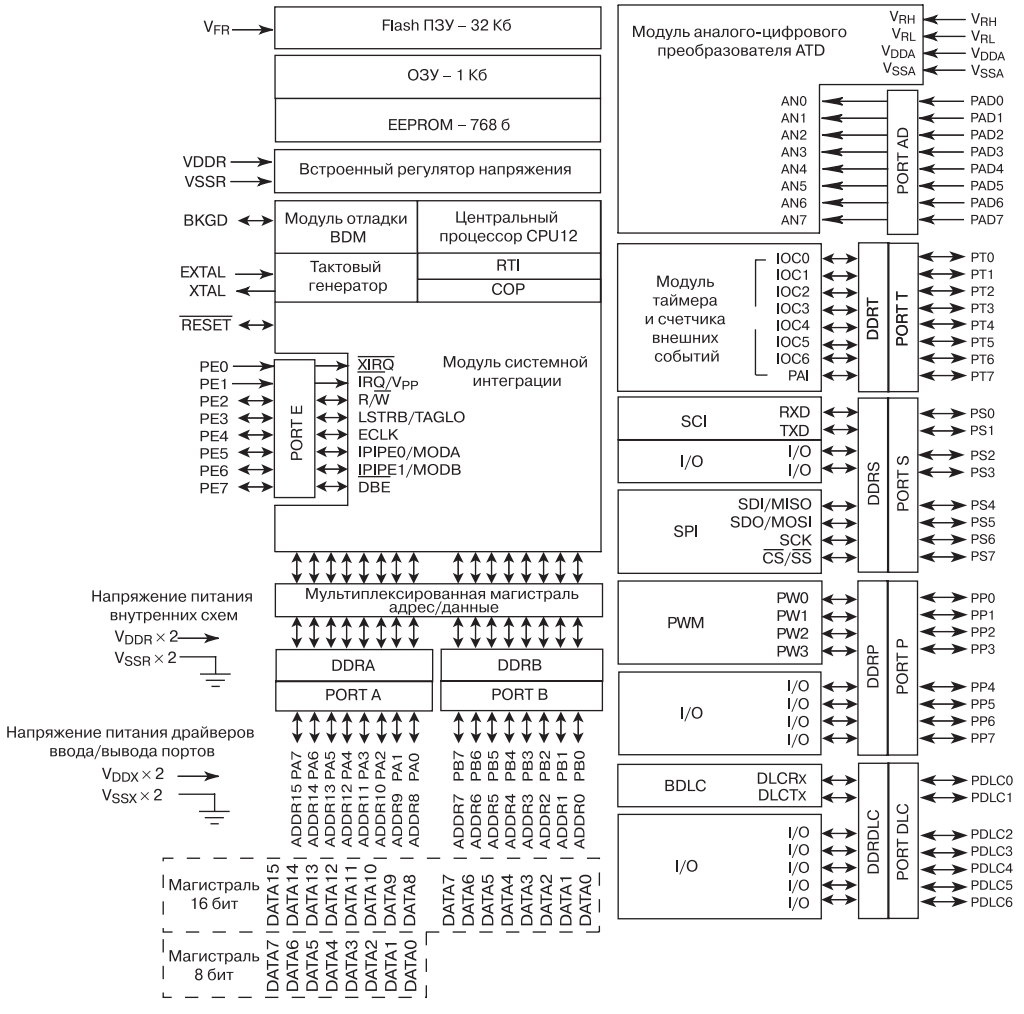


Рис. 1.3. *Структура микроконтроллеров MC68HC912B32 и MC68HC12BE32*

ностью данных 16/16 и 32/16 в целочисленном и дробном форматах, команды выбора максимального и минимального числа, команды нечеткой логики. Операции сложения или вычитания двухбайтовых чисел выполняются за 2 или 3 такта системной шины МК. Центральный процессор 68HC12 поддерживает 16 способов адресации, при этом исполнение каждой команды из группы арифметических или логических команд возможно с использованием по крайней мере 12 способов адресации. Центральный процессор 68HC12 имеет двухадресные команды, позволяющие выполнять пересылки 8– и 16–разрядных данных между двумя ячейками памяти или регистрами специальных функций минуя регистры центрального процессора. Четыре коман-

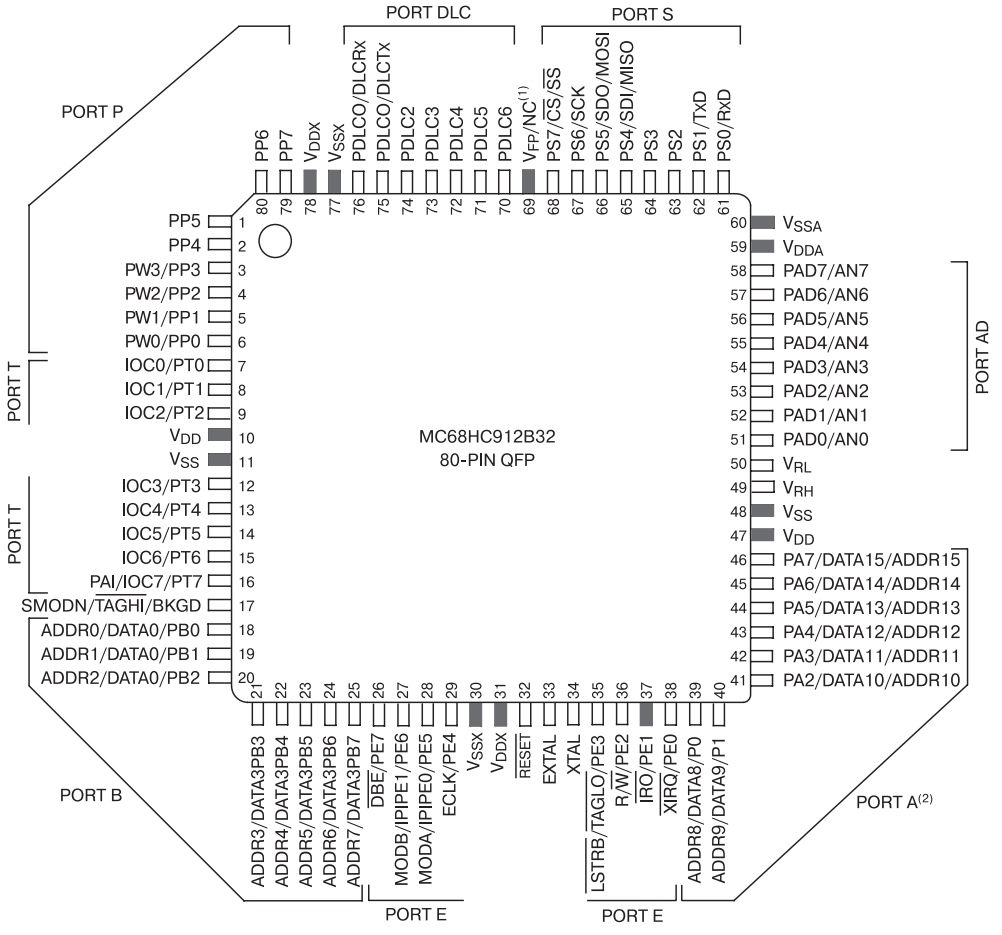


Рис. 1.4. *Цоколевка корпуса микроконтроллеров MC68HC912B32 и MC68HC12BE32*

ды предназначены для реализации алгоритмов нечеткой логики (fuzzy logic): команда фазификации MEM, команда обработки нечетких переменных REV, команда обработки нечетких переменных REVW и команда дефазификации WAW.

Микроконтроллеры семейства 68HC12 обладают резидентной Flash памятью программ, объемом до 32 Кб, оперативной памятью данных до 1 Кб, энергонезависимой памятью данных типа EEPROM объемом до 768 байт. Они имеют встроенный модуль отладки, который позволяет отлаживать программы, а также выполнять операции стирания/программирования Flash и EEPROM, взаимодействуя с персональным компьютером по однопроводному последовательному интерфейсу.

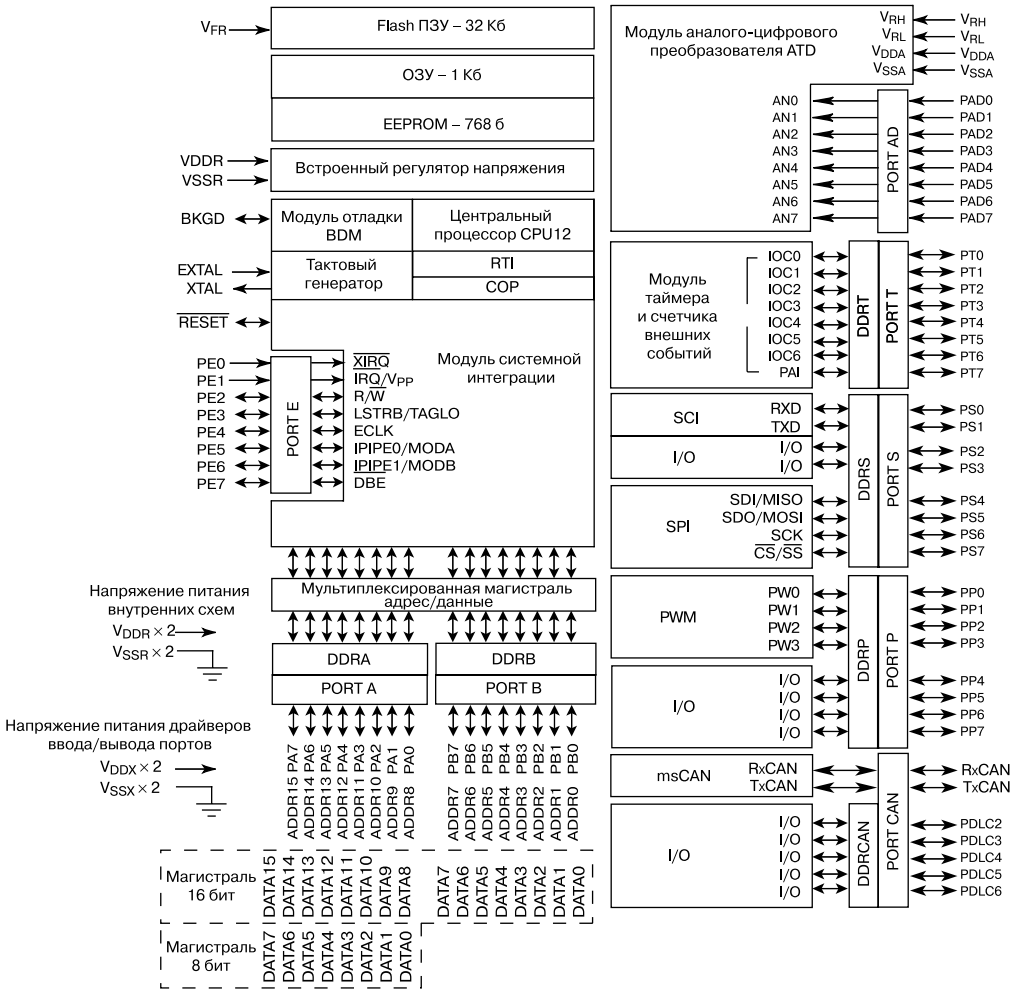


Рис. 1.5. *Структура микроконтроллера MC68HC912BC32*

Микроконтроллеры семейства 68HC12 имеют до семи многофункциональных двунаправленных портов ввода/вывода, модуль аналого-цифрового преобразователя, модуль таймера с функциями входного захвата и выходного сравнения, 16–разрядный счетчик внешних событий, модуль широтно–импульсного модулятора и несколько контроллеров последовательных интерфейсов. Полный перечень возможных для МК семейства 68HC12 периферийных модулей приведен в таблице рис. 1.7.

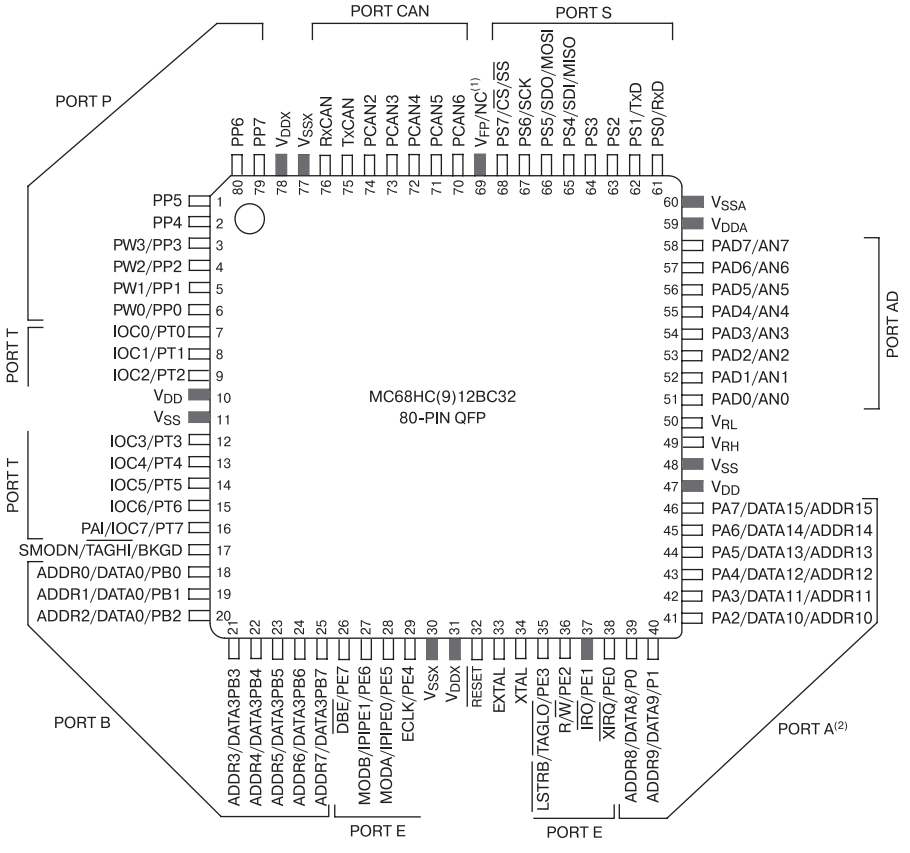


Рис. 1.6. *Цоколевка корпуса микроконтроллера MC68HC912BC32*

1.4. Микроконтроллеры HCS12

Подобно семейству 68HC12, семейство HCS12 объединяет ряд микроконтроллеров с одинаковым процессорным ядром CPU HCS12, различающихся объемом резидентной памяти и набором периферийных модулей, интегрированных на кристалл МК. Различные модели МК в составе семейства имеют Flash память программ объемом до 512 Кб, оперативную память объемом до 12 Кб. Напряжение питания большинства моделей семейства – 5,0 В, что позволяет обеспечить электромагнитную совместимость в автомобильных применениях. Частота внутренней системной шины МК семейства HCS12 равна 25 МГц, что существенно увеличивает их производительность по сравнению с МК семейства 68HC12.

Все модели МК семейства HCS12 имеют в своем составе следующие функциональные блоки:

- Оперативное запоминающее устройство и постоянное запоминающее устройство трех типов: Flash, EEPROM, масочного типа;
- Порты с двунаправленными линиями ввода/вывода;

Функциональные модули в составе МК	**MC68HC912B32**	**MC68HC12BE32**	**MC68HC912BC32**	**MC68HC12BC32**
Центральный процессор CPU12	+	+	+	+
Системная магистраль	+	+	+	+
Память программ Flash 32Кб	+		+	
Память программ однократно программируемая 32Кб		+		+
EEPROM 768байт	+	+	+	+
ОЗУ 1 Кб	+	+	+	+
Модуль таймера TIM	+	+	+	+
Модуль аналого–цифрового преобразования ATD	+	+	+	+
Усовершенствованный модуль таймера ECT	+	+	+	+
Модуль широтно–импульсного модулятора PWM	+	+	+	+
Модуль контроллера асинхронного последовательного обмена SCI	+	+	+	+
Модуль контроллера синхронного последовательного обмена SPI	+	+	+	+
Модуль контроллера последовательно обмена CAN			+	+
Модуль контроллера последовательного обмена BDLC	+	+	+	+
Сторожевой таймер COP	+	+	+	+
Модуль отладки BDM	+	+	+	+
Модуль делителя для низкочастотной синхронизации	+	+	+	+

Рис. 1.7. *Сравнительные характеристики микроконтроллеров семейства 68HC12B*

- Модуль таймера с 16–разрядным счетчиком временной базы и 8 каналами захвата/сравнения;
- Подсистему последовательного обмена с несколькими контроллерами ввода/вывода различных стандартов (SCI, SPI, CAN и др.);
- Модуль АЦП с 8–и или 10–разрядным представлением результата;
- Модуль ШИМ с разрешением 8 или 16 разрядов.

Структура МК MC9S12DP256B представлена на рис. 1.8. Обратите внимание, что большая часть периферийных модулей этого МК аналогична модулям микроконтроллеров семейства 68HC12. От ранее рассмотренного МК MC68HC912B32 микроконтроллер DP 256 отличает увеличенный до 256 кб объем Flash памяти программ, наличие в его составе модуля усовершенствованного таймера ECT, двух 8–канальных модулей аналого–цифрового преобразования ATD, пяти контроллеров интерфейса информационной сети в стандарте CAN.

1.4.1. Семейство HCS12

Семейство HCS12 объединяет более 30 моделей МК. Однако мы не хотим концентрировать внимание читателя на изучении модельного ряда HCS12, поскольку детальное знание различных представителей семейства необходимо при профессиональной деятельности. А в процессе обучения мы наоборот, хотим использовать общность структуры и режимов работы функциональных модулей МК 68HC12 и HCS12. Поэтому в рамках этого первого знакомства с семейством HCS12 ограничимся рассмотрением системы условных обозначений МК и кратким обзором структуры некоторых МК семейства.

1.4.2. Обозначения МК

Каждая модель МК в составе семейства 68HC12/HCS12 имеет собственное сокращенное обозначение. Это обозначение используется для маркировки корпуса МК и при заказе ИС МК у производителя. Система сокращенных обозначений для МК семейства 68HC12 и HCS12 представлена на рис. 1.9. Обратите внимание, что каждое поле в сокращенной записи отражает определенную техническую характеристику изделия. В перечень технических характеристик входят не только структура МК и частота тактирования (функциональные характеристики), но и тип корпуса, диапазон рабочих температур, т.е. характеристики, связанные с конструктивным исполнением и условиями эксплуатации конечного изделия.

1.4.3. Модельный ряд HCS12

В настоящее время компания Motorola/Freescale Semiconductor выпускает около 40 МК с процессорным ядром HCS12 (рис. 1.10*). Традиционно для Motorola/Freescale Semiconductor все МК одного семейства группируются в се-

*) – таблица рис. 1.10 была дополнена авторами перевода с учетом выпущенных в 2005–2006 г.г. новых моделей МК семейства HCS12.

рии по схожести периферийных устройств. Внутри серии МК различаются объемом резидентной памяти и числом линий портов ввода/вывода. Все МК семейства HCS12 внутри одной серии совместимы по выводам корпусов, благодаря чему на печатную плату можно установить МК с большей памятью без изменения платы.

Сегодня в состав семейства HCS12 входят 6 серий. Серия A – МК общего применения с тремя типами относительно простых контроллеров последовательных интерфейсов. Серии C и CG – недорогие модели без EEPROM способные работать при пониженном напряжении питания. Серия D, объединяющая наибольшее число МК, ориентирована на использование в CAN–приложениях. Отдельные модели содержат до 5 CAN–контроллеров на кристалле! Серия E – МК с встроенным ШИМ–генератором для управления электроприводом. Серия H – специализированные МК для управления приборными панелями автомобилей, содержат драйверы шаговых двигателей стрелочных индикаторов и контроллер управления ЖК–индикатором. Указанная производителем специализация не препятствует использованию этих МК в устройствах другого типа с многофункциональными приборными панелями. Три последних МК в таблице рис. 1.10 – родоначальники новых серий. Среди них особенно интересен МК HC9S12NE64 с контроллером 10/100 Ethernet на кристалле.

1.5. Заключение по главе 1

В этой главе мы дали определение встраиваемым системам и привели примеры таких систем. Мы также обсудили проблемы, связанные с разработкой встраиваемых систем. В заключении мы провели обзор основных технических характеристик микроконтроллеров семейства 68HC12/HCS12, тех МК, с которыми Вы будете иметь дело на протяжении всей этой книги.

1.6. Вопросы и задания

Основные

1. Перечислите основные блоки вычислителя.
2. Какие функции выполняет центральный процессор в составе вычислителя?
3. Дайте определение термину компьютер.
4. Дайте определение термину микропроцессор.
5. Дайте определение термину микроконтроллер.
6. Как называется магистраль микропроцессорной системы, по которой передаются сигналы управления от центрального процессора к блоку памяти?
7. Какие функции может исполнять модуль ШИМ микроконтроллера 68HC12 в системе управления?
8. Перечислите, какие домашние встроенные системы не были упомянуты в этой главе?

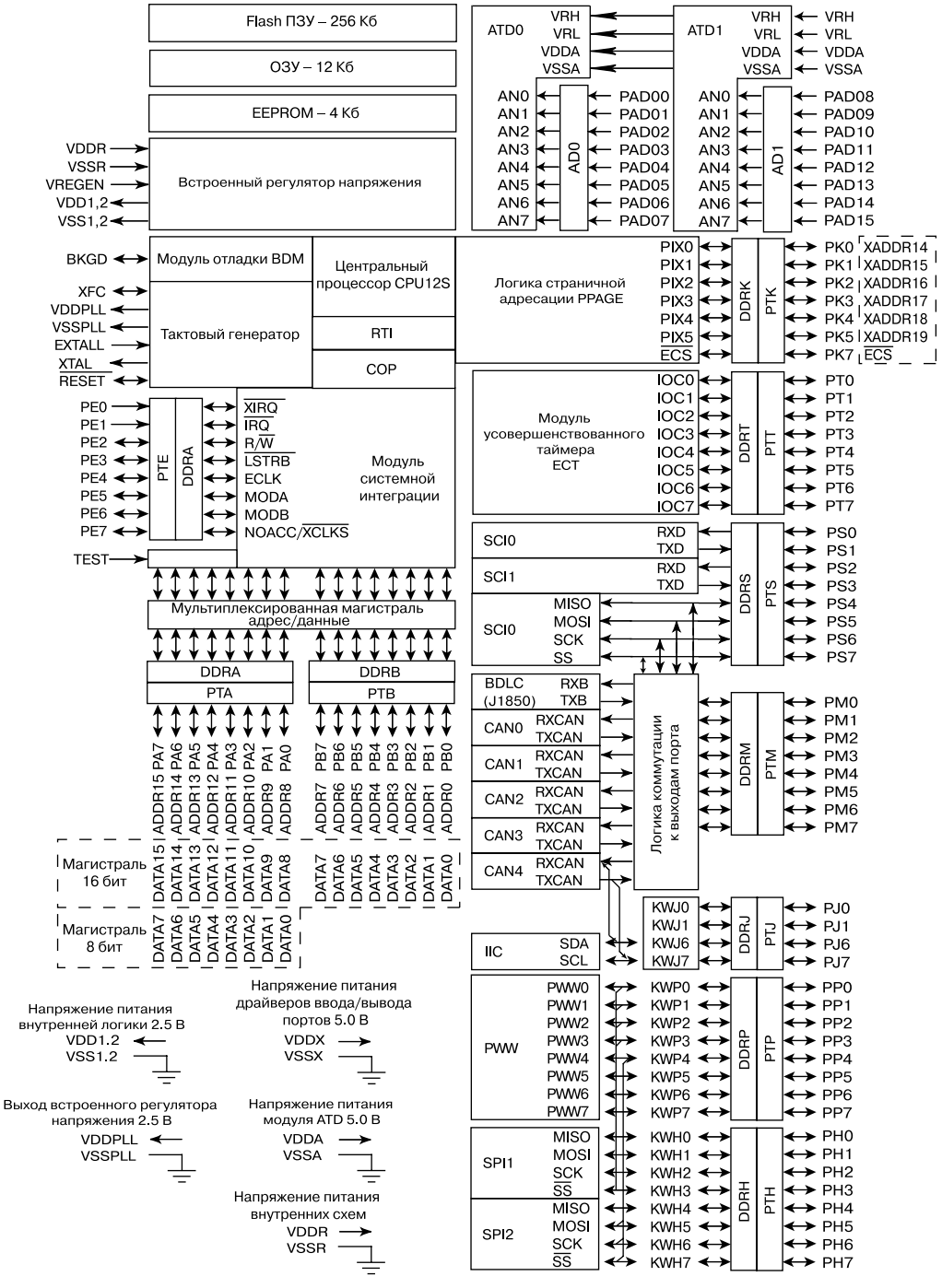


Рис. 1.8. *Структура микроконтроллера MC9S12DP256B*

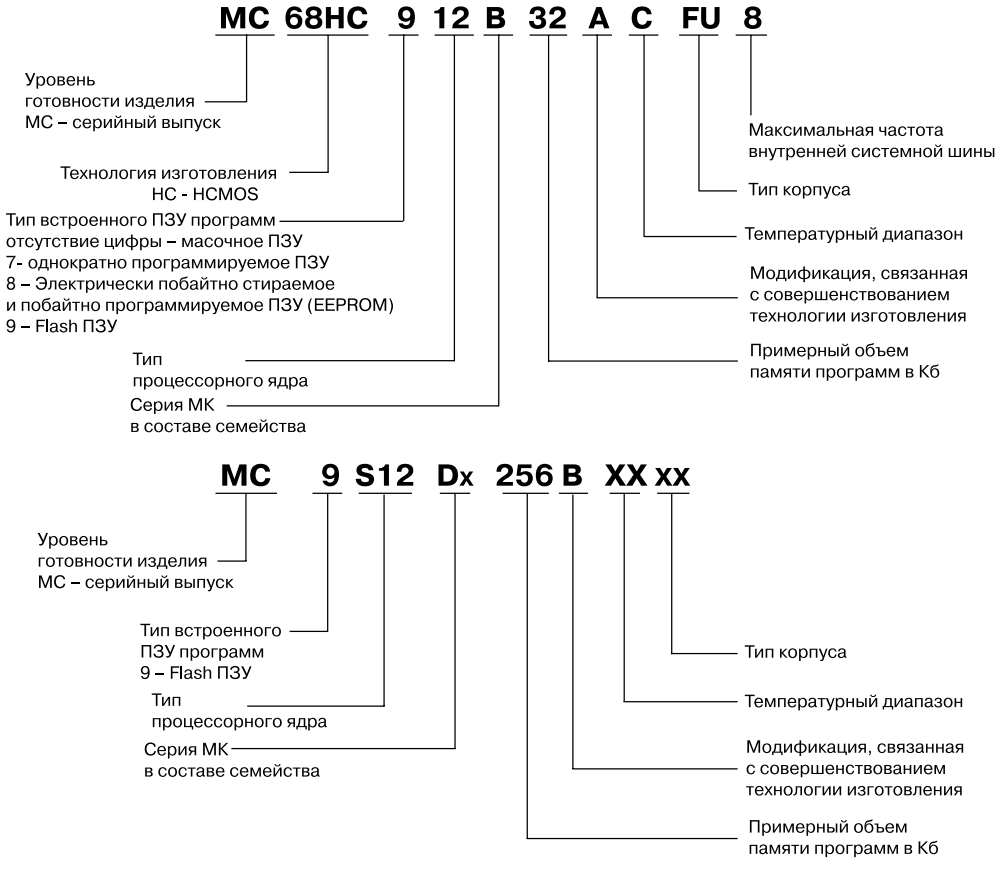


Рис. 1.9. *Система обозначений МК семейства 68HC12/HCS12*

Более сложные

1. Поясните, чем отличаются микроконтроллер и персональный компьютер.
2. В тексте изученной Вами главы утверждается, что разработчик персональных компьютером может не уделять значительного внимания мощности потребления и размерам блока памяти своего изделия. Почему так? Каковы ограничения? В каких изделиях этого класса ограничения на мощность потребления умеренные, в каких более жесткие?
3. Приведите примеры работы встроенных систем в реальном масштабе времени?
4. В каких случаях Вы, как разработчик встроенной системы, можете выбрать однокристальный режим работы МК, а в каких расширенный режим работы?

Тип МК	ПЗУ FLASH, байты	ОЗУ, байты	EEP-ROM, байты	Число линий ввода/вывода	Контроллеры последовательных интерфейсов	Таймер Число каналов/ разрядность	АЦП Число каналов/ разрядность	Модуль ШИМ Число каналов/ разрядность	Специальные модули *)	Частота шины CPU, МГц	Напряжение Питания, В
Серия A											
MC9S12A32	32000	4096	1024	91	IIC 2 SCI SPI	8/16	8/10	4/16 8/8		25	5
MC9S12A64	65536	4096	1024	59 91	IIC 2 SCI SPI	8/16	8/10	4/16 7/8 8/8		25	5
MC9S12A128 **MC9S12A128B**	131072	8192	2048	59 91	IIC 2 SCI SPI	8/16	8/10	4/16 8/8	EBUS	25	5
MC9S12A256B	26144	12288	4096	59 91	IIC 2 SCI SPI	8/16	2/10 3/10 8/10	4/16 8/8	EBUS	25	5
MC9S12A512	512000	4096	1024	59	IIC 2 SCI SPI	8/16	16/10	7/8		25	5
Серия C											
MC9S12C32	32000	2000	нет	60	CAN SCI SPI	8/16	8/10	6/8	LVI	16 25	3,3 5,0
MC9S12C64	64000	4000	нет	60	CAN SCI SPI	8/16	8	6/8	LVI	25	3,3 5,0
MC9S12C96	96000	4000	нет	60	CAN SCI SPI	8/16	8	6/8	LVI	25	3,3 5,0
MC9S12C128	128000	4000	нет	60	CAN SCI SPI	8/16	8	6/8	LVI	25	3,3 5,0

Рис. 1.10. *Технические характеристики МК семейства HCS12*

Серия D											
MC9S12D32	32000	4096	1024	91	CAN IIC 2 SCI SPI	8/16	8/10	4/16 8/8		25	5,0
MC9S12D64	65536	4096	1024	59 91	CAN IIC 2 SCI SPI	8/16	8/10	4/16 7/8 8/8		25	5,0
MC9S12DJ64	65536	4096	1024	59 91	CAN IIC J1850 2 SCI SPI	8/16	8/10	4/16 7/8 8/8		25	5,0
MC9S12DB128	131072	8192	2048	91	BYTE- FLIGHT 2 CAN 2 SCI 2 SPI	8/16	16/10	8/8	EBUS	25	5,0
MC9S12DB128B	131072	8192	2048	91	BYTE- FLIGHT CAN 2 SCI 2 SPI	8/16	16/10	8/8	EBUS	25	5,0
MC9S12DG128 **MC9S12DG128B**	131072	8192	2048	59 91	2 CAN IIC 2 SCI SCP 2 SPI	7/16 8/16	16/10	8/8	EBUS	25	5,0
MC9S12DG256B	26144	12288	4096	91	2 CAN IIC 2 SCI 2 SPI	8/16	16/10	4/16 8/8	EBUS LVI	25	5,0
MC9S12DJ128 **MC9S12DJ128B**	131072	8192	2048	59 91	2 CAN IIC J1850 2 SCI 2 SPI	7/16 8/16	8/10	4/16 8/8	EBUS	25	5,0
MC9S12DJ256B	262144	12288	4096	59 91	2 CAN IIC J1850 2 SCI 3 SPI	7/16 8/16	8/10	4/16 8/8	EBUS	25	5,0
MC9S12DP256B	262144	12288	4096	91	5 CAN IIC J1850 2 SCI 3 SPI	8/16	16/10	8/8	EBUS	25	5,0
MC9S12DP512	512000	12288	4096	91	5 CAN IIC J1850 2 SCI 3 SPI	8/16	16/10	8/8	EBUS	25	5,0

Рис. 1.10. *Технические характеристики МК семейства HCS12 (продолжение)*

MC9S12DT128 MC9S12DT128B	131072	8192	2048	91	3 CAN IIC 2 SCI 2 SPI	8/16	8/10	4/16 8/8	EBUS	25	5,0
MC9S12DT256B	262144	12288	4096	91	3 CAN IIC 2 SCI 2 SPI	8/16	8/10	4/16 8/8	EBUS LVI	25	5,0
Серия E											
MC9S12E64	65536	4096 8192	нет	59 91	IIC 3 SCI SPI	4/16	16/10	6/8	PWMF 2 DAC	25	5,0
Серия G											
MC9S12GC16	16000	2000	нет	60	SCI SPI	8/16	8/10	6/8	LVI	16 25	3,3 5,0
MC9S12GC32	32000	2000	нет	60	SCI SPI	8/16	8/10	6/8	LVI	16 25	3,3 5,0
MC9S12GC64	64000	4000	нет	60	SCI SPI	8/16	8/10	6/8	LVI	25	3,3 5,0
MC9S12GC96	96000	4000	нет	60	SCI SPI	8/16	8/10	6/8	LVI	25	3,3 5,0
MC9S12GC128	128000	4000	нет	60	SCI SPI	8/16	8/10	6/8	LVI	25	3,3 5,0
Серия H											
MC9S12H128	131072	6000	4096	99	2 CAN IIC 2 SCI SPI	8/16	16/10	4/16 8/8	EBUS LVI LCD 32×4 MC 24	16	5,0
MC9S12H256	262144	12288	4096	99	2 CAN IIC 2 SCI SPI	8/16	16/10	4/16 8/8	EBUS LVI	16	5,0
Разные МК											
MC9S12NE64	64000	8000	нет	48 80	Ethernet IIC 2 SCI SPI	4/16	8/10	нет	RTI	25	3,3 5,0
MC9S12T64	65536	2048	2048		SCI SPI	8/16	8/10	4/16 8/8			5,0
MC9S12UF32	32768	3584	нет	75	SCI USB 2.0	8/16				30	5,0

Примечание:
EBUS – модуль интерфейса внешней магистрали;
LVI – модуль контроля за пониженным напряжением питания;
RTI – модуль меток реального времени;
PWMF – модуль специализированного генератора для управления силовыми коммутаторами в электроприводе;
DAC – модуль одноканального ЦАП;
LCD 32 4 – контроллер управления ЖКИ дисплеем (4 группы по 32 сегмента);
MC 24 – 24 выхода с повышенной токовой нагрузкой для управления маломощными шаговыми электродвигателями.

Рис. 1.10. *Технические характеристики МК семейства HCS12 (продолжение)*

5. В тексте главы утверждается, что тестирование встраиваемой микропроцессорной системы является достаточно сложной задачей, решение которой должно быть продумано на стадии проектирования изделия. Почему так?

Исследовательские

1. Программно–аппаратный дуализм встраиваемой микропроцессорной системы?
2. В настоящее время встраиваемые системы перестают быть автономными устройствами. Они связываются между собой подобно объединению компьютеров в сеть Internet. Поэтому в скором времени пользователь столкнется с необходимостью понимания не только своей собственной системы, но и понимания абстрактного взаимодействия систем. Как Вы представляете себе проблемы создания информационных сетей на основе встраиваемых систем? Как изменится инфраструктура нашего общества при реализации этих идей?

Глава 2

ПРОГРАММИРОВАНИЕ ВСТРАИВАЕМЫХ СИСТЕМ И СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

ПОСЛЕ ИЗУЧЕНИЯ ГЛАВЫ ВЫ СМОЖЕТЕ:

ЦЕЛИ

- Провести сравнительный анализ языка ассемблер и языков программирования высокого уровня для разработки программного обеспечения встраиваемых систем.
- Рассказать об особенностях языка Си, которые позволили выбрать его в качестве основного языка высокого уровня для кодирования управляющих программ встраиваемых систем.
- Рассказать об основных положениях метода структурного проектирования и применить этот метод на практике в области встраиваемых систем.
- Понять важность документирования в процессе разработки.
- Перечислить правила ведения инженером рабочей тетради.
- Применить методы универсального языка моделирования к описанию функционирования встраиваемых систем.

2.1. Почему мы программируем микроконтроллеры на Си? 44
2.2. Преимущества программирования на языке ассемблер.....45
2.3. Преимущества языков высокого уровня 45
2.4. Оптимальная стратегия - программирование на Си и на ассемблере.....48
2.5. Структурное проектирование 49
2.6. Рабочие тетради 59
2.7. Блок-схемы алгоритмов.....60
2.8. Пример применения.....62
2.9. Заключение по главе 2 63
2.10. Что еще почитать? 65
2.11. Вопросы и задания 66

Предлагаемая Вашему вниманию глава, быть может, самая главная в этой книге. Она предлагает Вам несколько шире взглянуть на процесс проектирования встраиваемых систем, нежели Вы делали это до настоящего времени. Мы покажем в этой главе, почему язык Си стал основным языком высокого уровня для проектирования управляющих программ встраиваемых систем. Далее мы сравним технологии программирования встраиваемых систем на Си и на ассемблере и покажем, что оптимальным является сочетание этих двух языков. Следующим предметом нашего изучения будет метод структурного проектирования в приложении к встраиваемым системам. Мы рассмотрим основные положения этого метода и приведем примеры его использования. Мы также рассмотрим способы описания программного продукта с использованием универсального языка моделирования.

2.1. Почему мы программируем микроконтроллеры на Си?

На протяжении всей этой книги для написания фрагментов программ управления мы используем язык Си. Мы выбрали Си для программирования встраиваемых микропроцессорных систем по многим причинам, которые, мы надеемся, станут Вам ясны по мере знакомства с материалом этой главы.

В предисловии мы адресовали данную книгу подготовленным читателям, которые знакомы с основами цифровой и микропроцессорной техники, имеют опыт программирования на языке ассемблера для какого-либо типа МК. Не пугайтесь, если Вы чувствуете себя недостаточно образованным в перечисленных областях знаний. Мы постарались преподнести материал данной книги так, чтобы он легко усваивался учащимися. Для того чтобы восполнить недостающие знания в области цифровой техники, рекомендуем обратиться к книге [9] или к иным подобным изданиям. Для предварительного знакомства с архитектурой и системой команд микроконтроллеров 68HC12 Вы можете использовать книгу [6].

В следующих параграфах данной главы мы познакомим Вас с технологиями создания программного обеспечения для встраиваемых систем. Мы проведем сравнительный анализ преимуществ и недостатков технологий программирования встроенных систем на Си и на ассемблере. Мы также постараемся разъяснить Вам, почему язык Си стал стандартом программирования для встроенных систем.

2.2. Преимущества программирования на языке ассемблер

Многие разработчики встраиваемых систем используют для программирования микроконтроллеров только язык ассемблера. И существует много убедительных доводов в пользу такого решения. В общем, хорошо написанная на языке ассемблера программа исполняется за меньшее время и занимает в памяти меньший объем, нежели та же программа, написанная на языке высокого уровня. Именно эти характеристики: время выполнения и размер программного кода, – являются критическими для приложения, где элементная база обладает относительно невысоким быстродействием, а память программ ограниченна в объеме.

В одном из курсов по микропроцессорной технике, мы попросили студентов во время лабораторных работ запрограммировать одну и ту же задачу на ассемблере и на языке Си. Пример был следующий: в массиве 16-разрядных чисел без знака необходимо было подсчитать число чисел, равных заданному значению. Поскольку компилятор Си сначала преобразует исходный текст программы в программу на языке ассемблер и только потом превращает ассемблерный текст в коды инструкций микроконтроллера, студенты смогли сравнить собственные ассемблерные программы с аналогичными программами, сгенерированными компилятором. И студенты могли убедиться, что их программы оказались компактнее и более логично написанными, чем ассемблерная программа на выходе компилятора.

В дополнение к уже отмеченным преимуществам, язык ассемблера предоставляет разработчику прямой доступ ко всем без исключения аппаратным средствам МК. Такой доступ возможен и в Си, но с меньшими возможностями, т.е. в ограниченном объеме. Однако это преимущество в полной мере может быть реализовано только высококлассным специалистом. Программист, успешно решающий задачи на ассемблере, должен детально разбираться в алгоритмах преобразования кодов и очень хорошо знать аппаратные возможности МК.

2.3. Преимущества языков высокого уровня

Если Вы программировали на ассемблере, то Вы должны были изучить массу приемов, которые и позволят Вам стать профессионалом высокого уровня. Вы должны были познакомиться не только с деталями архитектуры МК, но и с особенностями его системы команд. Однако при переходе к другому типу МК, Вам придется потратить не так мало времени, чтобы адаптировать свои программы к МК с другой системой команд ассемблера. Такую ситуацию называют несовместимостью кодов.

Недавно один из авторов этой книги должен был выполнить разработку, в которой заказчик определил использование МК компании Atmel. Он никогда ранее не использовал 8-разрядные МК от компании Atmel, но был не прочь их освоить. Поэтому автор решил выполнять проект на Си. Это позволило ему быстрее завершить исполнение заказа. Если бы он решил выполнять проект на ассемблере, то ему потребовалось значительно больше времени не только для изучения архитектуры и алгоритмов работы периферийных модулей, но и системы команд, поскольку МК Atmel существенно отличаются от знакомых автору МК Motorola/Freescale

Semiconductor. Кроме того, область применения устройства была такой, что он мог не беспокоиться о размере программы и времени ее выполнения. Поэтому в приведенных условиях выбор языка Си для проекта был логичным и естественным.

В общем, языки высокого уровня позволяют создать такой исходный текст программы, который будет обладать свойством переносимости, его сможет прочесть и понять не только разработчик программы (свойство читабельности), и, наконец, на его основе будет сгенерирован компактный исполняемый машинный код,. Кроме того, языки высокого уровня располагают библиотеками математических действий над числами, представленными в различных форматах, в том числе и в формате с плавающей запятой. Структура программы на языке высокого уровня хорошо соотносится с методами структурного проектирования программного обеспечения. Виртуозный программист на языке ассемблера может опровергнуть наши обоснования преимуществ программирования на языках высокого уровня, однако наши выводы основываются на опыте коллег разного возраста, и, соответственно, разной квалификации, а также на собственном опыте.

Поскольку программы на языке высокого уровня обладают более высокой степенью абстракции, эффективность программирования возрастает. Поэтому программист может завершить проект за меньшее время, чем он выполнял бы этот проект на ассемблере. Кроме того, если один и тот же ранее отлаженный код используется в следующих проектах, то эффективность программирования значительно повышается (концепция многократного использования программного кода для типовых функций управления).

Языки высокого уровня обладают важным свойством переносимости программного кода. Это означает, что программа, написанная на языке высокого уровня для одного МК, затем может быть скомпилирована другим компилятором для МК с другим процессорным ядром. И эта программа тоже окажется работоспособной. Для того, чтобы программа обладала свойством переносимости, синтаксис языка высокого уровня для разных компиляторов должен быть абсолютно одинаков. В частности, таким свойством обладает язык Си стандарта ANSI (American National Standards Institute). Разработчики называют его просто «ANSI C». Основная цель стандартизации состоит в том, чтобы обеспечить разработчику возможность написания типовых функций управления один раз с последующим их многократным использованием в разных проектах и для разных микроконтроллеров.

Языки высокого уровня также обеспечивают очень хорошую читабельность кода. Если программа хорошо написана, другой программист, не ее разработчик, может прочесть исходный текст и понять, какой алгоритм реализован и как программа работает. Мы специально не определили, что означает термин «хорошо написана». Мы оставим этот весьма важный аспект до параграфа 2.5, в котором обсудим структурное проектирование.

Еще одним преимуществом языков высокого уровня является простота реализации различных математических вычислений. Например, операции умножения и деления чисел в формате представления с плавающей точкой достаточно сложно реализуются на ассемблере. Строго говоря, имеются специальные библиотеки функций на ассемблере, которые, впрочем, сейчас уже достаточно трудно достать. В тоже время с математическими вычислениями прекрасно справляются языки высокого уровня. Так в главе 4 мы покажем, как на Си записать выражение для вычисления длительности импульса, используя значения моментов времени, полученные с таймера:

$$T_{IMP} = (2^{16} \times n) + (Stop\_count - Start\_count)$$

Вычисление этого уравнения достаточно сложно выполнить на ассемблере, однако для языка высокого уровня это рутинная задача. Однако не следует забывать, что мы должны включить эти математические операции в код программы, а это увеличит затраты памяти МК на проект.

Подведем итоги нашему короткому сравнению языка ассемблер с языками высокого уровня. Мы можем сделать вывод, что языки высокого уровня пригодны и весьма полезны для программирования встраиваемых систем.

2.3.1. Выбираем язык высокого уровня для программирования встраиваемых систем

Проведя обзор в Internet, Вы обнаружите достаточно большое число разнообразных компиляторов различных языков высокого уровня для встраиваемых систем. Вы без труда найдете компиляторы Си, C++, Java. Ada, Fortran и некоторые другие. Каждый из этих языков имеет свои преимущества и недостатки. Часто выбор языка программирования может определяться специфическими особенностями задачи или просто пожеланиями заказчика. Детальное сравнение всех перечисленных языков выходит за рамки этой книги и вряд ли возможно.

Для этой книги мы выбрали язык Си, потому что именно он сейчас используется в организациях разработчиков, и он обеспечивает хороший доступ к аппаратным ресурсам микроконтроллеров. Язык Си известен как некоторый промежуточный язык, который объединяет в себе свойства языка высокого уровня и одновременно обеспечивает легкий доступ к регистрам и ячейкам памяти МК. Именно это свойство отмечал один из разработчиков Си господин Ричи (Ritchie).

Язык Си был изобретен в середине 60-ых годов прошлого столетия. Несмотря на почтенный возраст, он так и остался одним из простых и компактных языков высокого уровня. Изначально язык Си был разработан для создания операционной системы Unix, поэтому его характеризуют как «инструмент для создания более сложных инструментов». Язык Си покрывает основные потребности программистов встраиваемых систем без отягощения редко используемыми конструкциями. Более того, программист может достаточно быстро освоить навыки программирования на Си и создавать приложения, которые по быстродействию и затратам памяти близки к ассемблерным. В завершение отметим, что основные конструкции языка Си делают их крайне удобными для реализации принципов структурного программирования.

2.3.2. Краткая история языка Си

Наш разговор о Си был бы неполным, если бы мы кратко не остановились на происхождении этого языка. Для более полного погружения в эту тему советуем обратиться к книге [8]. Ниже по тексту параграфа мы даем краткое изложение одного из разделов этой книги.

Первая версия языка Си была разработана в середине 60-ых годов для разработки операционной системы Unix в лаборатории Bell. Один из самых первых разработчиков этого языка, Кен Томсон (Ken Thompson), решил, что требуется язык для создания более сложных языков программирования. Он создал такой

язык и назвал его «B». В процессе развития своего творения Томсон постоянно боролся с ограничением ресурсов памяти, что теперь очень похоже на встраиваемые системы. Деннис Ричи (Dennis Ritchie) решил расширить язык «B» свойством генерировать малый по объему код, который сможет соперничать с кодом, написанным на ассемблере. В 1973 году важнейшие свойства этого нового языка «C» были получены.

Возрастающая популярность Си заложена в его переносимости. Компиляторы Си были созданы для многих платформ (так в сообществе разработчиков называют процессорное ядро МК), отчего его популярность еще больше выросла. Наиболее бурно Си стал использоваться в 80 годах, когда стал основным языком для создания программ персональных компьютеров.

Американский национальный институт стандартизации (American Nationa1 Standards Institute – ANSI) в 1982 году учредил комитет X3J11 для разработки стандарта языка Си. В 1989 доклад комитета был передан в Международную организации стандартизации (Internationa1 Organization for Standardization – ISO) и международную электротехническую комиссию (Internationa1 E1ectrotechnica1 Commission – IEC) и был утвержден в качестве стандарта ISO/IEC 9899-1990. За этим стандартом последовало неизбежное развитие языка, которое было узаконено в 1999 стандартом ISO/IEC 9899. И Си стал языком, который наиболее часто используется в компьютерной индустрии.

2.4. Оптимальная стратегия – программирование на Си и на ассемблере

Итак, мы установили, что предпочтительно программировать встраиваемые системы на языке высокого уровня. И в качестве такого языка мы обоснованно выбрали язык Си. Но и ассемблер имеет свои преимущества. Так на каком языке мы все-таки будем программировать встраиваемые системы?

Практика применения языка Си и ассемблера показывает, что оптимальный результат, как с точки зрения экономии времени на разработку, так и по времени исполнения программы, можно получить, используя в одном проекте сразу два языка программировании: и Си, и ассемблер. Основная часть прикладной программы, в которой производятся преобразования данных, будет написана на СИ, в то время, как критичные по времени реализации фрагменты алгоритма, следует оформить на ассемблере. Кроме того, ассемблер иногда используют для программной поддержки некоторых внешних по отношению к МК периферийных ИС. Драйверы обмена с такими ИС обычно требуют многократного переключения отдельных линий портов МК, что в компиляторах Си для некоторых МК удобнее выполнить на ассемблере. Еще один случай обязательного включения ассемблерного фрагмента в текст основной программы на Си мы продемонстрируем на примере использования команд ассемблера 68HC12/HCS12 для преобразования данных по правилам нечеткой логики (см. гл. 7).

Ранее мы упомянули, что конструкции языка Си как нельзя лучше сочетаются с методом структурного проектирования. Настало время познакомиться с этим методом.

2.5. Структурное проектирование

Несколько следующих параграфов мы посвятим изложению основных идей метода структурного проектирования. Для создания этого раздела мы использовали материалы, изложенные в [1, 7], а также опыт собственных разработок.

Метод структурного проектирования не гарантирует обязательного успешного завершения проекта. Однако этот метод значительно увеличивает вероятность создания за ограниченное время качественной системы, полностью совместимой с управляемым объектом.

2.5.1. Основные положения метода структурного проектирования

Теория. Метод структурного проектирования – это регламентированная последовательность действий, которая позволяет разработать структуру аппаратных и программных средств встраиваемой системы, удовлетворяющих техническим требованиям к проектируемому устройству.

Первым шагом этой последовательности действия является как можно более полное описание технических требований к будущей системе. В подавляющем большинстве случаев технические требования формулирует не тот, кто потом реализует систему. Поэтому технические требования должны быть как можно более точно доведены до исполнителя. Исполнитель обязан подробно исследовать предложенные ему технические требования, понять их обоснованность и согласованность. Представьте себе трагедию разработчика, который выполнил систему, работающую правильно, но по неправильному техническому заданию! Разработчик потратил время и деньги на проект, но он не нужен заказчику! Кто виноват? Вывод: структурное проектирование использует определение проблемы как путь к определению решения этой проблемы.

Применение. На протяжении всего этого параграфа мы будем иллюстрировать применение выдвинутых концепций на примере разработки системы управления стереоусилителем. Прототип нашего примера - контроллер стеренусилителя, был разработан доктором Паррисом Нилом (Parris Neal) из Аэрокосмической Академии в Колорадо. Парис – превосходный инженер, разрабатывает стереоусилители мирового уровня. Любой из его проектов – это произведение искусства. Паррис разработал стереоусилитель, который может принимать звуковые сигналы от шести различных источников. Пользователь должен выбрать сигнал либо с помощью переключателей на передней панели корпуса усилителя (рис. 2.1), либо с дистанционного пульта управления, связанного со стереоусилителем по инфракрасному каналу. Паррис попросил первого автора этой книги разработать микропроцессорный контроллер для этого усилителя, используя 8-разрядный RISC МК компании Atmel. Ему было интересно на практике исследовать возможности МК Atmel в качестве низкостоимостного МК для следующих проектов.

После первого обсуждения были выявлены следующие технические требования к проекту:

- Необходимо разработать микропроцессорный контроллер для управления стереоусилителем;

- Контроллер должен обеспечить подключение ко входу срereоусилителя одного их шести источников звукового сигнала. Выбор источника сигнала может осуществляться оператором с пульта управления или с дистанционного пульта управления, связанного с основным устройством по инфракрасному каналу.
- При разработке устройства управления необходимо использовать МК компании Atmel.

После знакомства с этими первыми техническими требованиями был создан список вопросов, который должен был бы помочь разработчику более полно понять все особенности данного проекта:

- Что конкретно должен делать контроллер в ответ на выбор номера канала воспроизведения?
- Кто несет ответственность за интерфейс сопряжения между контроллером и стереоусилителем?
- Каковы электрические характеристики сигналов с передней панели усилителя и с удаленного пульта управления?
- Какие сигналы должен формировать контроллер для управления переключением каналов усилителя?

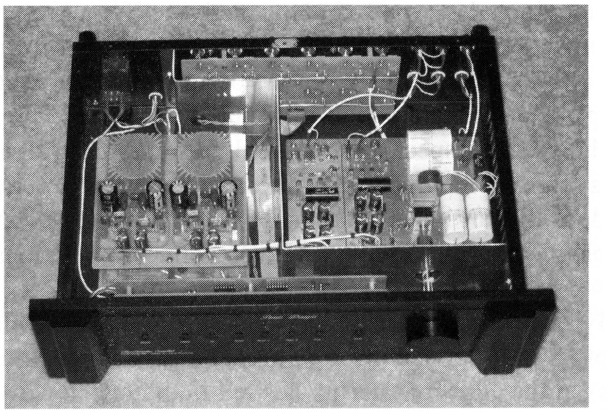
В ответ на эти вопросы заказчик (доктор Паррис Нил) выдал детальное словесное описание желаемых режимов работы своего устройства на четырех страницах. Он также нарисовал обобщенную блок-схему алгоритма процесса управления. На основе этого описания нами совместно в процессе непрерывного обсуждения были созданы структурная схема контроллера и более подробная блок-схема алгоритма. Далее мы два месяца переписывались по электронной почте, чтобы уточнить все детали устройства. Заметьте, в течение этих двух месяцев ни одной строки кодов программы не было написано!

Теория. Тщательный детальный анализ требований заказчика должен предшествовать собственно процессу проектирования. Эти требования затем превращаются в детальное описание технических характеристик будущего устройства. Технические характеристики содержат полное описание режимов функционирования устройства, включая реакцию устройства на каждое входное воздействие, а также алгоритмы обработки данных и обработки ошибок. Все пункты описания технических характеристик устройства, которые могут толковаться двояко, должны быть совместно обсуждены заказчиком и исполнителем с целью устранения разногласия по каждому пункту. Когда формулирование порядка работы устройства закончено, можно задуматься о способах воплощения этого устройства.

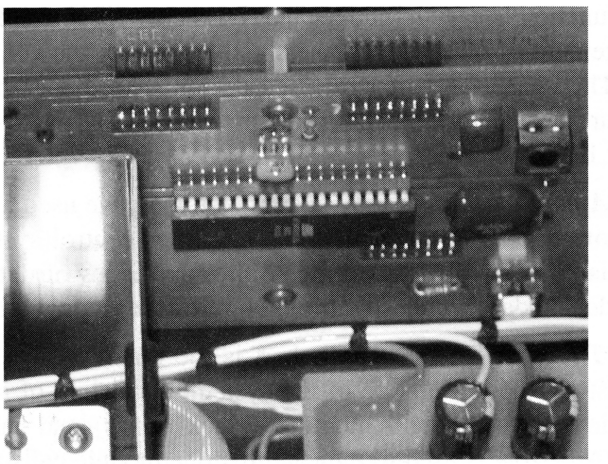
Вторым шагом структурного проектирования является разбиение поставленной задачи (системы) на несколько иерархически взаимосвязанных более мелких задач (подсистем). Что это все означает? Мы разобьем систему на несколько функционально законченных подсистем, и определим взаимосвязи между ними. Поскольку для каждой такой подсистемы определены функции, но техническая реализация этих функций пока еще не ясна, разработчики именуют их «черными ящиками».

Каждый такой «черный ящик» – это хорошо описанный блок, который выполняет некоторую законченную функцию Мы знаем его входы и выходы, но пока не знаем все детали функционирования. Мы продолжает делить систему на такие «черные ящики» до тех пор, пока функция каждого выделенного блока не станет полностью ясной для понимания. Кроме того, в процессе разделения на блоки и создания из них иерархической структуры проектируемой системы мы показываем связи между блоками.

а) Вид спереди



б) Плата с микроконтроллером



в) Вид сзади

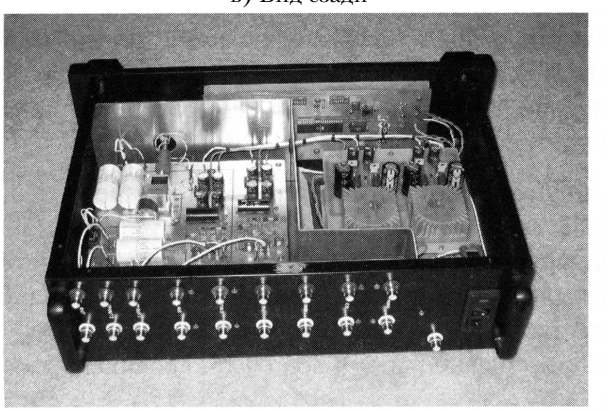


Рис. 2.1. *Внешний вид стереоусилителя с дистанционным управлением*

Третий шаг структурного проектирования – это создание графического образа работы системы, который способствует более полному пониманию внутренних связей между блоками в процессе функционирования устройства. Для создания графического образа системы применяются структурные схемы и диаграммы работы.

Применение. Как было ранее упомянуто, мы использовали детальные структурные схемы и блок-схемы алгоритмов для обсуждения с заказчиком технических характеристик проектируемой системы. Было бы чрезвычайно трудно превратить 4 страницы текста с описанием требований к устройству во взаимосвязанную картину без использования этих графических образов.

Теория. Структурная схема алгоритма функционирования системы (или ее аппаратного обеспечения, или программы управления) – это основной результат структурного проектирования, который иллюстрирует, каким образом большая система (или электрическая схема, или программа управления) состоит из модулей, которые изображены блокам на рисунке. Стрелки используются для того, чтобы показать, что один из модулей системы формирует сигналы для другого модуля. В случае структурной схемы программного обеспечения один модуль вызывает другой, к которому направлена стрелка. Стрелка с кружком показывает, что один программный модуль передает данные другому модулю. Блок-схема алгоритма тоже очень полезный инструмент, который позволяет проследить процесс преобразования данных на протяжении исполнения всей программы. Однако блок-схемы алгоритма читабельны только для относительно небольших алгоритмов, когда эти схемы умещаются на одной или двух страницах. В универсальном языке моделировании блок-схемы алгоритмов заменены на диаграммы работы. Мы вернемся к этому вопросу в 2.7.

Применение. Структура программного обеспечения контроллера управления стереоусилителем показана на рис. 2.2. Блок-схема алгоритма – на рис. 2.3. Проанализировав эти рисунки, можно увидеть, что структура программы управления показывает взаимосвязь отдельных модулей программы, в то время, как блок-схема алгоритма является графическим описанием порядка функционирования этой программы. В этот момент, Вы можете подумать: «Наконец то! Теперь можно кодировать программу!». Но нет, существует еще несколько обязательных этапов структурного проектирования, которые должны предшествовать написанию текста программы.

Теория. Следующий шаг структурного проектирования – детальное определение функций каждого блока. Для каждого выделенного на предыдущих этапах «черного ящика» определяются функции входов и выходов. В случае программного обеспечения определяются данные, которые передаются программным функциям и возвращаются ими. Для описания связей между входами и выходами блоков используют псевдокодирование.

Этап псевдокодирования определить принципы программной реализации отдельных блоков структурной схемы. После завершения этого этапа разработчик будет иметь полную структурную схему, которая описывает поставленную задачу в целом, и множество детализированных блоков, которые описывают отдельные функции проектируемого устройства.

Применение. В проекте контроллера стереусилителя мы разместили псевдокод непосредственно в графические образы блок-схемы алгоритма (рис. 2.4).Вы можете убедиться, что теперь для каждого блока мы имеем полное перечисление его функций, после которого можно приступить к непосредственному написанию текста программы.

Рис. 2.2. *Структура программы управления стереоусилителем*

Теория. Следующий шаг структурного проектирования – написание исходного текста программы на Си, в редких случаях на ассемблере. Причем под написанием программы понимается не только создание исходного текста программы, но и его поэтапная отладка. Отладка может производиться с использованием трех стратегий.

Первая стратегия соответствует методу проектирования сверху-вниз. Сначала пишется и отлаживается основная функция (main.c) программы управления. При этом все вызываемые функции симулируются пустыми программными модулями. По мере продвижения сверху вниз, все большее число функций наполняется содержанием, и для них записывается исходный текст программы.

Вторая стратегия соответствует методу проектирования снизу-вверх. В соответствие с этим методом сначала пишутся и отлаживаются пока несвязанные функции нижнего уровня. Постепенно разработчик переходит к функциям более высокого уровня, выстраивая иерархические связи в соответствие со структурной схемой.

Третья стратегия сочетает в себе две предыдущие. Попеременно пишутся и отлаживаются функции сверху и снизу структурной схемы. Объединение проекта в целом осуществляется на каком-то из средних уровней.

До настоящего времени мы рассматривали все шаги метода структурного проектирования преимущественно в приложении к разработке программного обеспечения встраиваемой системы. Однако эти же этапы в полной мере применимы также к проектированию аппаратного обеспечения системы.

Применение. В проекте контроллера стереусилителя мы специально разработали аппаратный симулятор, который использовался на этапе отладки программного обеспечения. Этот симулятор состоял из некоторого количества переключателей для имитации органов управления на передней панели стереоусилителя, и светодиодов, которые отображали состояние сгенерированных контроллером сигналов уп-

равления. Функциональная схема симулятора приведена на рис. 2.5. С помощью этого симулятора мы провели отладку, а затем поэтапно проверили функционирование разработанной программы. Доктор Парис Нил настоятельно попросил нас проверить программное обеспечение в автоматическом режиме, т.е. без применения программных средств отладки, по каждому из возможных сценариев работы. И только затем мы разместили контроллер в корпусе стереоусилителя и приступили к комплексным испытаниям законченного изделия.

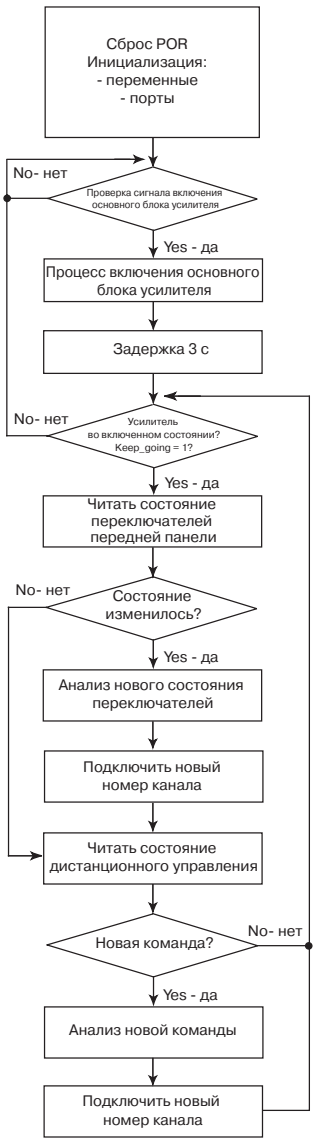


Рис. 2.3. *Блок-схема алгоритма управления стереоусилителем*

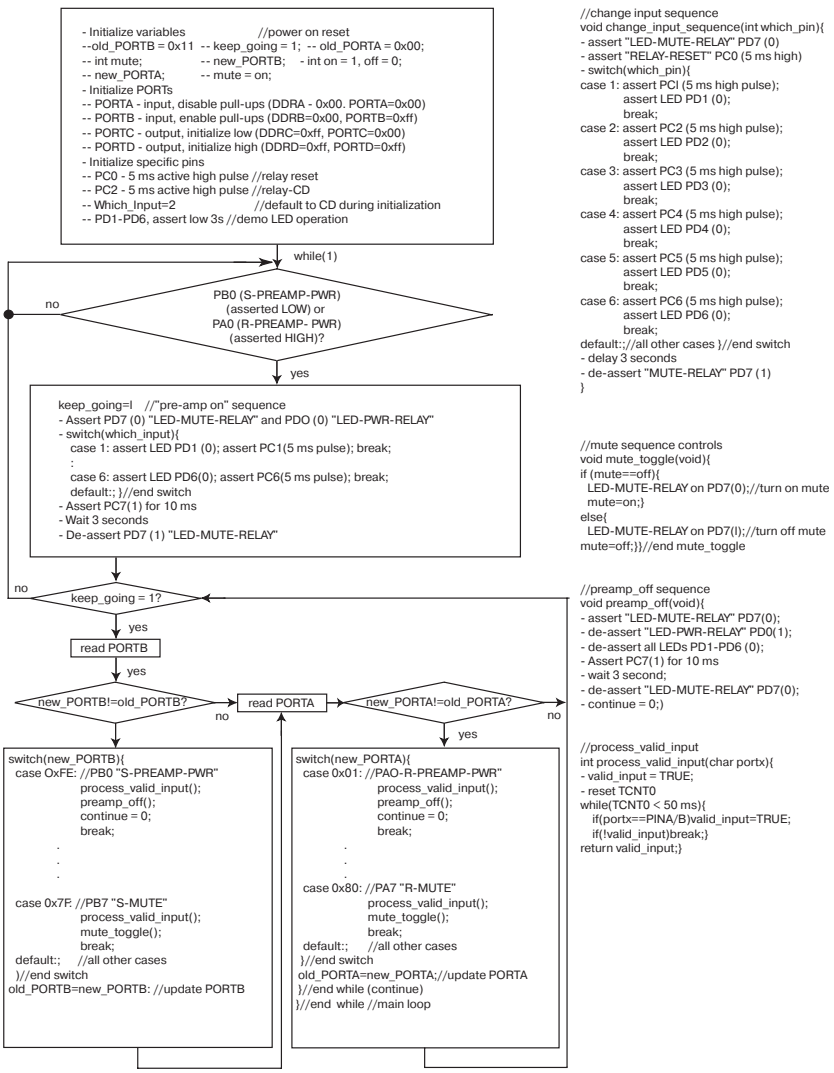


Рис. 2.4. *Псевдокод программы управления стереоусилителем*

Теория. Заключительным этапом метода структурного проектирования является определение критериев, по которым можно будет сделать вывод, что устройство удовлетворяет поставленным техническим требованиям. Это предполагает разработку стратегии верификации, отладки и тестирования разработанного устройства. Программа верификации – это не одно и тоже, что программа тестирования. Тем не менее, тестирование – это значительная часть процесса верификации. Для того, чтобы правильно испытать систему, разрабатывается специальная программа тестирования, которая позволяет полностью проверить режимы работы прибора и их соответствие техническим требованиям. Основной задачей является выявить и зафиксировать имеющиеся ошибки, и очень важно убедиться, что проект соответствует планируемому поведению.

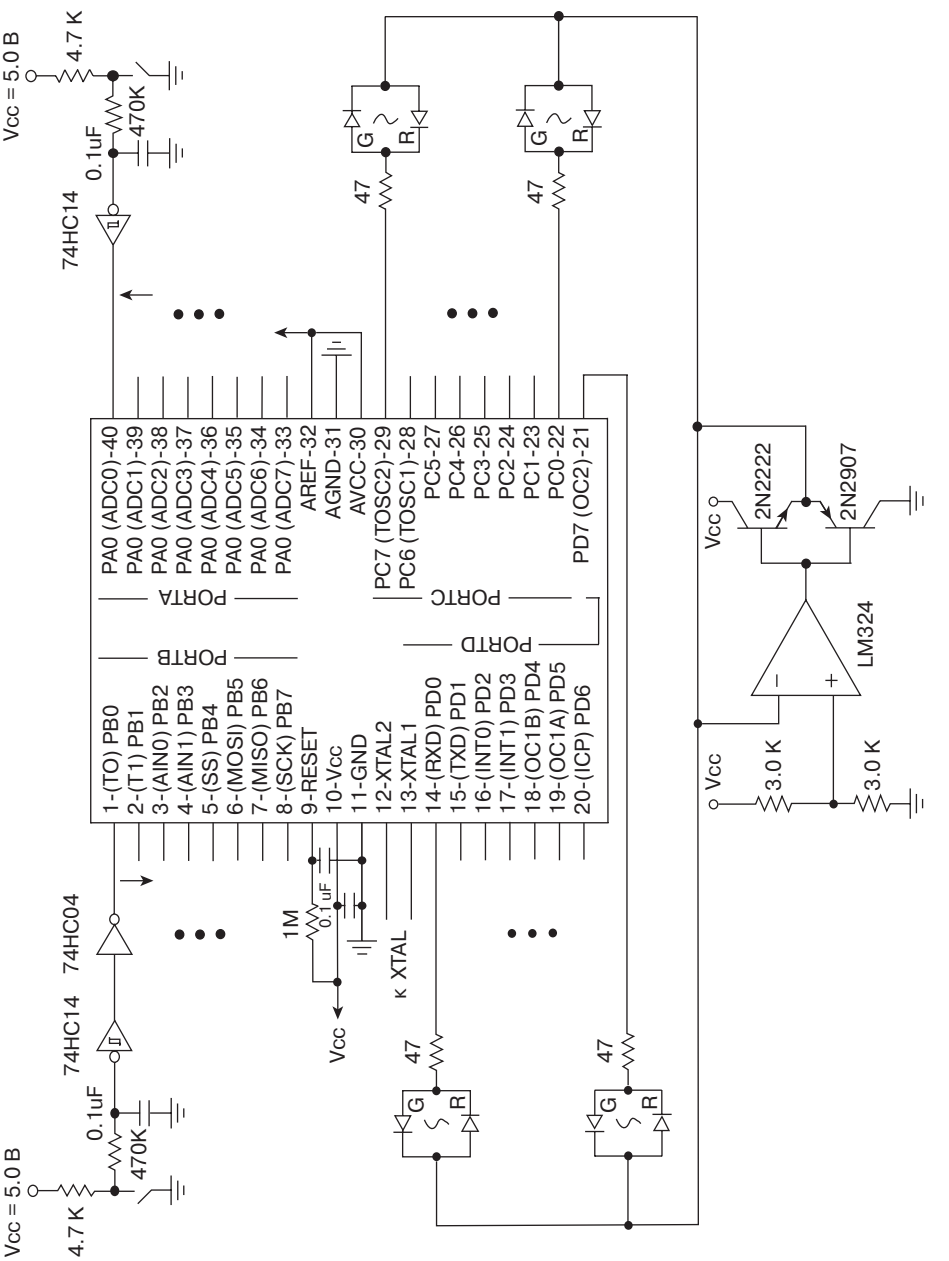


Рис. 2.5. *Функциональная схема имитатора для тестирования программы стереоусилителя*

Ошибки в проекте могут быть различной породы. Мы кратко рассмотрим их в порядке возрастания неприятностей от них.

Самыми простыми для устранения являются синтаксические ошибки. Эти ошибки выявляет компилятор в процессе обработки исходного текста программы. Компилятор выдает сообщения двух типов: предупреждения («Warning») и ошибки («Error»). Предупреждения выдаются компилятором в тех случаях, когда компилятору «кажется», что некоторые конструкции программы не удачны. При этих ошибках код ны выходе компилятора получается. Несмотря на то, что код будет образован, Вы должны будете принять решение по поводу исправления или нет этих мест в программе. Ошибки с сообщением «Error» не позволят Вам создать файл загрузочного модуля, поэтому Вам придется заняться их немедленным устранением. При этом следует знать, что всего лишь одна синтаксическая ошибка может вызвать генерацию сразу нескольких сообщений об ошибках.

Ошибки исполнения в реальном времени можно выявить только в процессе выполнения программы. Они обычно приводят к разрушению алгоритма управления. Например, если Вы при написании текста программы для выполнения задержки на 3 мс неправильно посчитали число отсчетов внутреннего генератора тактирования, то программа будет успешно исполняться, но задержка будет не соответствовать 3 мс. Ошибки исполнения в реальном времени достаточно сложно выявляются. Составление специальной методики тестирования поможет Вам выявить подобные неисправности.

Наиболее неприятный тип ошибок – принцип это неправильные начальные условия для составления программы. В этом случае может получиться, что программа полностью удовлетворяет требованиям, но сами технические требования неправильные. Никакие маленькие коррекции не исправят положения дел. Вот почему мы обязательно должны затратить так много времени на разбор и выработку всех условий работы устройства.

Всеобъемлющее тестирование может использовать технику сверху вниз, когда сначала исследуется общее поведение системы, а затем детали поведения в отдельных режимах. А может наоборот, снизу вверх, когда тестирование начинается с проверки правильного функционирования драйверов периферии. Иногда используется смешенная техника. В любом случае, методика тестирования определяется конкретным проектом. Хороший тест проверяет также живучесть (робастность) программного обеспечения. Под живучестью понимают ситуацию. При которой на вход системы могут поступить непредполагаемые комбинации сигналов, и при этом программное обеспечение должно оставаться работоспособным и формировать сигналы управления, по крайней мере. Не создающие аварийной ситуации для исполнительных устройств.

Применение. В процессе тестирования контроллера стереоусилителя были выявлены ситуации, которые не были предусмотрены начальными установками на проектирование программы. Например, если пользователь выбрал устройство для воспроизведения, то предыдущее устройство отключается от входа усилителя, а следующее выбранное подключается. При этом не подумали, а что следует делать контроллеру, если пользователь с пульта управления выбрал то же устройство, которое уже подключено. Контроллер в соответствие со сценарием сначала отсоединял устройство, а затем его же коммутировал опять. Это вызывало неприятные для слуха шумы. Только при тестировании устройства на реальном объекте мы смогли выявить эту неисправность и достаточно просто исправили программу.

Следует заметить, что процесс тестирования занимает много времени, поскольку он многоступенчатый. Если какие-либо ошибки обнаружены, то вносятся исправления в исходный текст программы, и все тесты должны быть проведены заново.

Итак, мы закончили свое первое знакомство с методом структурного проектирования. В следующем разделе мы остановимся на вопросах документирования процесса разработки.

2.5.2. Документирование программ

Теория. Многие начинающие программисты уверены, что документирование программного обеспечения проекта – это добавление некоторого количества коротких комментариев к работающей версии программы. Однако хорошее документирование программы значительно важнее самого программного обеспечения. Хорошая документация содержит все разработанные на предыдущих этапах структуры, которые были обсуждены нами в предыдущем разделе. Все эти структуры помогают нам не только закодировать программу, но и содержат информацию о том, как программа работает.

Полная документация на программное обеспечение содержит комментарии, письменное описание принципов построения программы, технические условия на написание программы, исходный текст программы, руководство пользователя, историю разработки программы и ее модификаций. Весь этот набор документов может быть разделен на внешнюю и внутреннюю документацию. Внешняя документация состоит из специально написанного текста, который поясняет, как работает программа. Внутренняя документация включает полный набор читаемых и понимаемых документов, который необходим для легкого исправления кода. Внутренняя документация состоит из комментариев, исходного текста программы. Структуры и специальных записей.

Комментарии помогают читателю определить, что программист хотел сделать в любой точке программы. В нашем контроллере стереоусилителя мы имели 4 страницы комментариев, предшествующих программе, которые со всей полнотой описывали работу системы. Мы также написали дополнительные комментарии по алгоритму работы программы, а также описание каждой отдельной функции. Более того, мы комментировали каждую строку исходного текста программы. Это может показаться избыточным. Но чем больше времени затрачено на составление комментариев, тем меньше времени понадобится для поддержания этого кода.

Самодокументирующий код означает, что имена переменных констант и функций выбираются такими, чтобы было интуитивно понятно ее назначение. Например, если функция называется «delay_3ms», то сразу понятно, что функция формирует задержку 3 мс.

Аккуратное форматирование исходного текста программы с помощью скобок и отступов позволяет показать структуру программы. Например, выполняемые в цикле команды записываются с отступом, позволяя легко идентифицировать набор выполняемых в цикле операций.

Применение. Несколько месяцев спустя после завершения разработки, заказчик (доктор Парис Нил) попросил меня внести изменения в алгоритм управления стереоусилителем. Благодаря хорошим комментариям автор быстро восстановил в памяти ход вычислительного процесса и внес изменения в исходный текст программы за 20 мин. Важно, что после этого определенное количество времени было затрачено на внесение дополнительных комментариев. Если бы мы не документировали программу качественно сразу, мы не смогли бы произвести необходимые доработки быстро.

2.5.3. Как язык Си соотносится со структурным проектированием

Язык Си как нельзя лучше сочетается со структурным проектированием. Программа на языке Си состоит из основной программы (main.c), которая описывает всю задачу. Основная программа вызывает программы-функции, которые выполняют определенные действия. Программы-функции можно рассматривать как реализацию модулей, которые мы обсуждали на этапе структурной декомпозиции проекта. Программы-функции могут, в свою очередь, вызывать другие функции, в соответствие с иерархической структурой проекта.

Метод проектирования сверху-вниз может быть легко реализован на Си. Основная программа показывает общий ход реализации алгоритма управления. Чем дальше Вы погружаетесь в функции все более низкого уровня, тем более детально Вы знакомитесь с особенностями алгоритма управления.

2.6. Рабочие тетради

С темой составления качественной документации проекта тесно связаны вопросы ведения рабочих тетрадей. В этом параграфе мы рассмотрим, зачем и почему нужны рабочие тетради. Представленный Вам материал в сокращенном виде является изложением статьи наших коллег, которая была написана в 1990 г [5].

Рабочие тетради предназначены для регистрации процесса научного поиска, обдумывания различных решений для проектов, сравнения и анализа предложенных решений, для записи результатов испытания собственных устройств. В инженерной практике одной из основных причин ведения рабочих тетрадей является накопление положительных результатов своей деятельности, которые потом, даже по прошествии нескольких лет, могут быть использованы в новом проекте, для подачи заявки на изобретение, могут быть включены в работу для присвоения научной степени.

2.6.1. Порядок ведения записей

Поскольку во многих организациях рабочая тетрадь является официальным документом, то ее ведение регламентируется жесткими правилами:

1. Листы тетради должны быть хорошо скреплены, так, чтобы страницы не могли быть из нее удалены свободным образом. Если же какие-то страницы вырываются, то для этого должны быть веские причины.
2. Все записи должны делаться последовательно несмываемыми чернилами.
3. Страницы тетради должны быть последовательно пронумерованы.
4. Каждая запись должна сопровождаться датой. При этом дата не должна быть двояко трактуемой. Запись «09 мая 2004 г.» лучше, чем «9/5/2004».
5. Если в предшествующие записи вносятся изменения, то эти изменения должны сопровождаться простановкой даты.
6. Нельзя стирать кажущиеся Вам ошибочными сведения. Их необходимо подчеркнуть их или выделить маркером.
7. Законченный материал должен быть помечен символами «X» или «Z» для уверенности, что ничего не было добавлено позже.

2.6.2. Содержание записей

Когда специалиста патентного отдела спрашивают, что должно быть включено в рабочие тетради, от отвечает «Все!». Для того, чтобы не заниматься излишним правописанием, но все таки выполнить требования по заполнению рабочих тетрадей, последовательно ответьте себе на следующие вопросы: что, кто, когда, где, почему и как? Когда Вы подготовите ответы на эти вопросы, выберите подходящий стиль изложения. Помните, что при необходимости, другой специалист должен понять проект и продолжить его.

Рабочие тетради , когда они ведутся должным образом, представляют собой прекрасный документ о содержании проекта. В следующем параграфе мы рассмотрим технику документирования алгоритмов с помощью блок-схемы алгоритма.

2.7. Блок-схемы алгоритмов

Теория. Универсальный язык моделирования (Unified Modeling Language – UML) – это язык для определения, представления, проектирования и документирования программных систем Основными составляющими языка UML являются элементы, связи, механизмы расширения и диаграммы. Универсальный язык моделирования предоставляет разработчику метод графической иллюстрации и имитационного моделирования системы, которая основана на программном принципе управления. Этот язык позволяет стандартизировать графическое изображение операций в системе с целью моделирования ее алгоритма работы в общем виде, без составления исходного текста управляющей программы. Универсальному языку моделирования посвящены более 100 научных монографий. Мы рассмотрим лишь малую часть сведений об этом языке, которые будут использованы нами для проектирования встраиваемых систем на протяжении всей этой книги. Если Вас будет интересовать получение более широких сведений по этой теме, обратитесь к литературе, рекомендованной в конце главы.

Свою историю универсальный язык моделирования ведет с начала 90х годов. В 1994 году Грэйди Буч (Grady Booch), Джеймс Рэмбо (James Rambaugh) и Ивар Якобсон (Ivar Jacobson) начали объединять несколько методов объектно-ориентированного моделирования в фирме Rational Software. Их целью была разработка методов сокращения времени реализации программыных продуктов для промышленности. И уже в 1995 году была представлена спецификация метода, названного «Unified Method». Первая версия универсального языка моделирования была принята консорциумом OMG (Object Management Group) в январе 1997 года. Утвержденная же в сентябре того же года версия UML 1.1 была принята на вооружение основными компаниями – производителями программного обеспечения, такими, как Microsoft, IBM, Hewlett-Packard.

Визуальные модели универсального языка моделирования широко используются в существующих технологиях управления проектированием систем, сложность, масштабы и функциональность которых постоянно возрастают. В практике эксплуатации программных информационных систем постоянно приходится

решать такие задачи как перераспределение вычислений и данных, обеспечение проведения параллельных вычислений, обеспечение безопасности доступа к информационным системам, оптимизация балансировки нагрузки систем, устойчивость к сбоям и многое другое. Основные виды визуальных моделей UML (универсального языка моделирования): диаграммы сценариев, диаграммы взаимодействия объектов, диаграммы классов, диаграммы состояний, диаграммы модулей и компонентов, диаграммы действий [2,3]. Графические средства представления алгоритмов позволяют переводить модели UML в исходный код объектно-ориентированных языков программирования, что значительно ускоряет процесс разработки. Поэтому универсальный язык моделирования прекрасно зарекомендовал себя на множестве успешных программных проектов. Мы не будем касаться объектно-ориентированного программирования в нашей книге. Поэтому мы используем лишь малую часть возможностей универсального языка моделирования для иллюстрации последовательности действий в проектируемых нами программ для встраиваемых систем.

Синтаксис языка универсального моделирования – условные графические обозначения различных типов операторов, которые позволяют составить блок-схему алгоритма. Блок-схема алгоритма – графическое изображение логической структуры алгоритма. Каждое действие алгоритма представляется в виде геометрической фигуры – условного графического обозначения оператора UML.

Основные типы операторов представлены на рис. 2.6. Программа начинается с оператора начального запуска и заканчивается оператором останова. Множество различных действий совершается под управлением программы в процессе ее выполнения. Каждое действие отображается оператором процесса, который производит изменение значения, формы представления или расположения данных. Оператор процесса может иметь любое количество входов, но только один выход. Переход от одного оператора к другому обозначается стрелкой. Если в процессе выполнения очередного действия производится анализ некоторого внешнего сигнала или внутреннего состояния программы с последующим выбором, по какому пути продолжить исполнения программы, то такое действие отображается условным оператором (другое название – оператор решения). Условный оператор имеет один вход и несколько выходов. После выполнения действий условного оператора программа может «избрать» только один путь. Поэтому условия, по которым определяется направление выхода из оператора решения, должны быть взаимоисключающими. Оператор слияния (поглощения), в соответствие со своим названием, позволяет соединить несколько отдельных ветвей алгоритма в одну. В завершение мы включили два оператора, которые характерны для структурного программирования. Это операторы ветвления и объединения, которые предназначены для отображения работы систем с параллельными вычислителями [2,3]. В качестве первого примера использования условных графических обозначений UML мы представили средствами универсального языка моделирования процесс структурного проектирования, блок-схема алгоритма которого приведена на рис. 2.7. На рис. 2.8 тот же процесс представлен с большей степенью подробности, что немедленно привело к усложнению блок-схемы алгоритма.

2.8. Пример применения

Основная идея, которую мы стремились развить на протяжении всей этой главы, состоит в том, что подлежащая реализации большая задача должна быть разбита на ряд малых взаимосвязанных задач. Идея декомпозиции задач лежит в основе метода структурного проектирования, который в равной степени применим к проектированию, как аппаратных, так и программных средств, что делает его особенно полезным при разработке встраиваемых систем. Более того, этот метод может быть использован на этапе постановки задачи, который предшествует структурному проектированию аппаратных и программных средств системы.

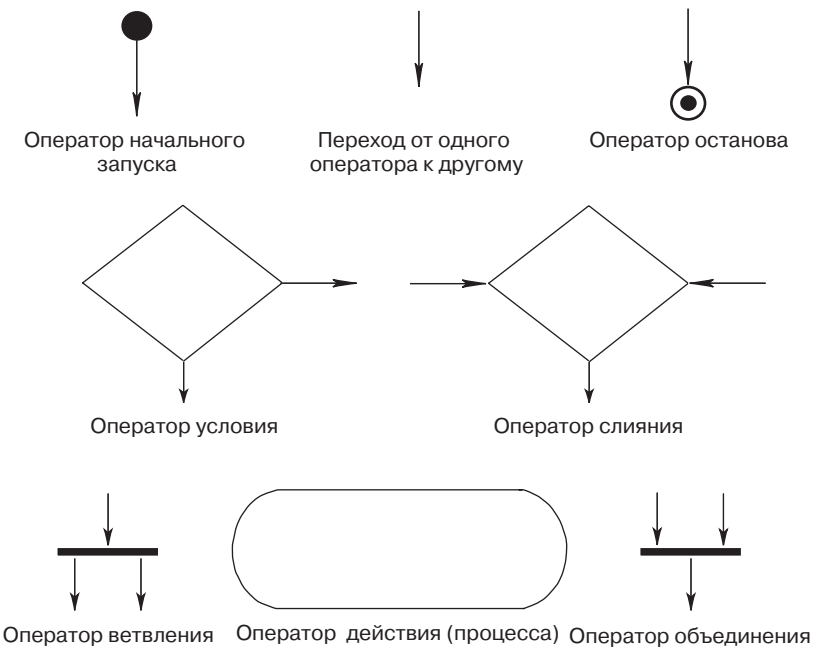


Рис. 2.6. *Условные графические обозначения операторов универсального языка моделирования UML*

В качестве темы дипломного проектирования одному из авторов этой книги было предложено разработать систему управления медицинским лазером для лечения различных расстройств зрения. Около шести месяцев он провел в библиотеке, изучая технологии лазерной глазной хирургии. В итоге, его представления о функциях контроллера управления хирургическим лазером изменились коренным образом в сторону их увеличения. Результаты своего исследования автор воплотил в структурной схеме рис. 2.9, которая отражает все возможные функции системы автоматизированного управления хирургическим лазером и их взаимосвязь в проекте. Обратите внимание, что в данном случае метод структурного проектирования использован для систематизации функций системы управления, когда задачи проектирования аппаратного и программного обеспечения системы еще не поставлены.

2.9. Заключение по главе 2

В этой главе мы провели сравнительный анализ преимуществ и недостатков языка ассемблер и языков программирования высокого уровня для разработки программного обеспечения встраиваемых систем. Среди языков высокого уровня мы сделали выбор в пользу языка Си, способного генерировать программный код, сравнимый по быстродействию и затратам памяти с кодом языка ассемблер. Мы установили, что

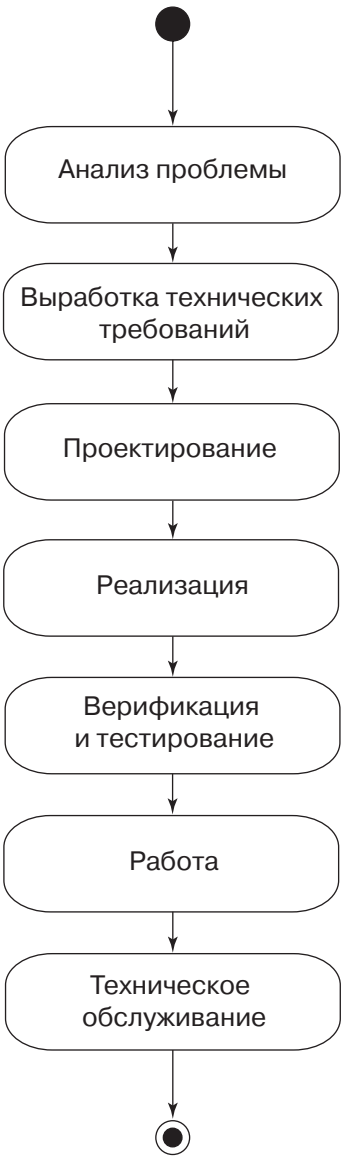


Рис. 2.7. *Блок-схема алгоритма структурного проектирования*

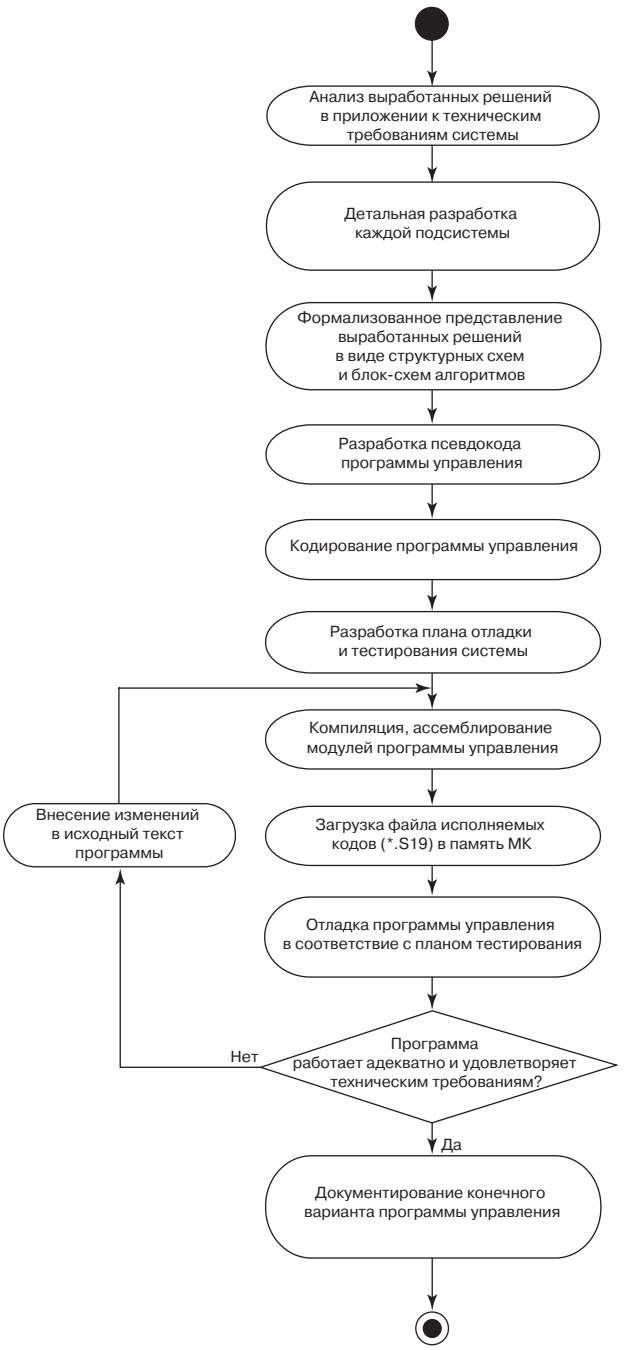


Рис. 2.8. *Блок-схема алгоритма разработки и тестирования программного обеспечения встраиваемой системы*

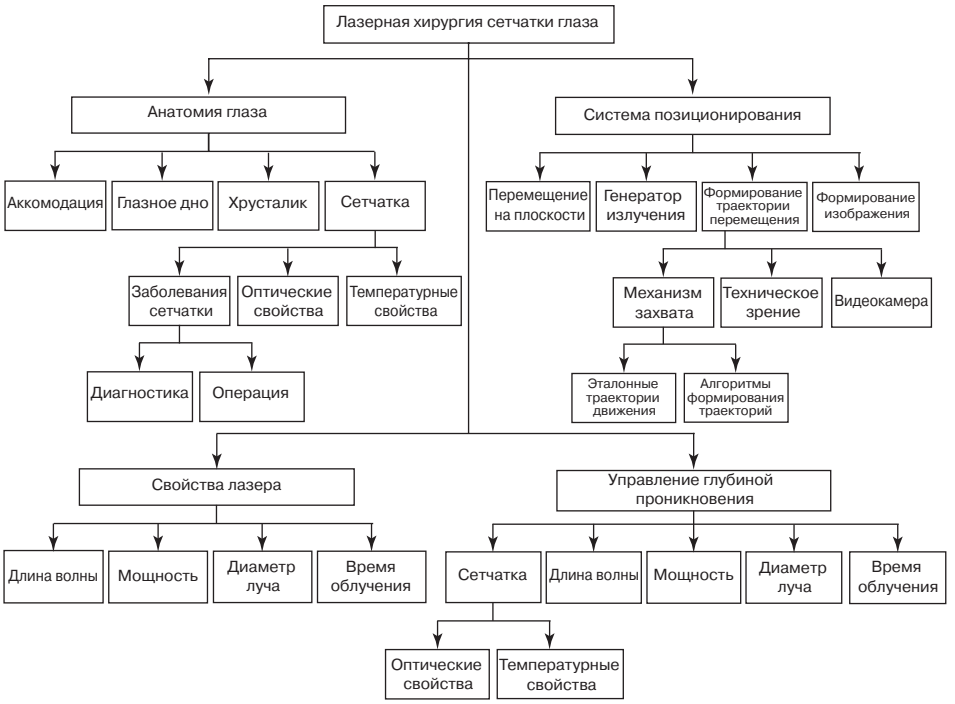


Рис. 2.9. *Структура функций медицинского лазера*

оптимальный результат может быть получен при программировании встраиваемых систем одновременно на Си и на ассемблере. Мы кратко познакомились с методом структурного проектирования и универсальным языком моделирования UML. Мы также познакомились с применениями метода структурного проектирования для определения функций управления медицинским лазером и для разработки структуры программного обеспечения контроллера стереоусилителя. Мы кратко рассмотрели условные графические обозначения операторов UML и познакомились с примерами описания алгоритмов посредством составления блок-схемы алгоритма из операторов UML.

2.10 Что еще почитать?

1. Dale, Nell, and Susan C. Lilly. Pascal Plus Data Structures. 4th ed. Englewood Cliffs, NJ: Jones and Bartlett, 1995.
2. Douglass, Bruce Powel. Real-Tire UML-Developing Efficient Objects for Erbedded Systers. 2nd ed. Boston: Addison-Wesley, 2000.
3. Fowler, Martin, with Kendall Scott. «UML Distilled-A Brief Guide to the Standard Object Modeling Language. 2nd ed. Boston: Addison-Wesley, 2000.
4. Kobryn, Chris. «UML 2001.» Corrunications o! the ACM 42, no. 10, (October 1999): 29-37.

5. McConnack, J. B., R. K. Morrow, H. F. Bare, R. J. Bums, and R. L. Rasmussen «The Comp1ementary Ro1es of Laboratory Notebooks and Laboratory Reports.» Paper presented at the annual meeting of the American Society for Engineering Educators Toronto, Canada, 1990.
6. Pack, Danie1, and Steven Barrett. 68HC12: Theory and Applications. Upper Sadd1e River, NJ: Prentice HaH, 2002.
7. Page-Jones, Meilir. The Practical Guide to Structured Systets Design. 2nd ed. Upper Sadd1e River, NJ: Yourdon Press, 1988.
8. Ritchie, Dennis M. «The Deve10pment of the C Language.» Paper presented at a meeting of the Association for Computing Machinery, Second History of Programming Languages Conference, Cambridge, MA, 1993.
9. Waker1y, John F. Digital Design Principles and Practices. 3rd ed. Upper Sadd1e River, NJ: Prentice HaH, 2000.

2.11 Вопросы и задания

Основные

1. Проведите сравнение достоинств и недостатков применения языка Си и языка ассемблер для программирования встраиваемых микропроцессорных систем.
2. Опишите следующие свойства метода структурного проектирования: переносимость, устойчивость, читабельность.
3. Какие другие компиляторы с языков высокого уровня, кроме компиляторов Си, Вы знаете? Укажите их источники в Internet.
4. Каковы основные причины быстрого распространения языка Си для программирования встраиваемых систем?
5. Что такое «черный ящик»?
6. Каково различие между структурой и блок-схемой алгоритма?

Более сложные

1. Преобразуйте блок-схему алгоритма рис. 2.3 в диаграмму действия UML.
2. Что называют псевдокодированием?
3. Объясните различия между методами проектирования сверху-вниз и снизу-вверх?
4. Каковы различия между внешней и внутренней документацией?
5. Поясните необходимость ведения рабочей тетради.

Исследовательские

1. Программно-аппаратный дуализм встраиваемой микропроцессорной системы?
2. Представьте себе, что Вас попросили разработать портативную метеостанцию, которая может быть установлена в некотором удалении от центрального поста и будет сообщать этому посту состояние окружающей среды на месте размещения. Метеостанция оснащена следующими приборами:

- Анемометром – прибором для измерения скорости ветра. Анемометр представляет собой колесо с лопастями, которое вращается со скоростью, пропорциональной скорости ветра. Будем считать, что электронная схема анемометра выдает один импульс амплитудой 5 В на один оборот колеса.
- Барометром – прибором для измерения атмосферного давления. На аналоговом выходе электронного преобразователя сигнала барометра формируется напряжение в диапазоне 0...5,0 В. Давлению в 640 мм рт. ст. соответствует напряжение 0 В, давлению 810 мм рт. ст. – напряжение 5,0 В. Преобразователь обладает линейной передаточной характеристикой.
- Гигрометром – прибором для измерения влажности воздуха. Гигрометр также оснащен электронным преобразователем сигнала с линейной передаточной

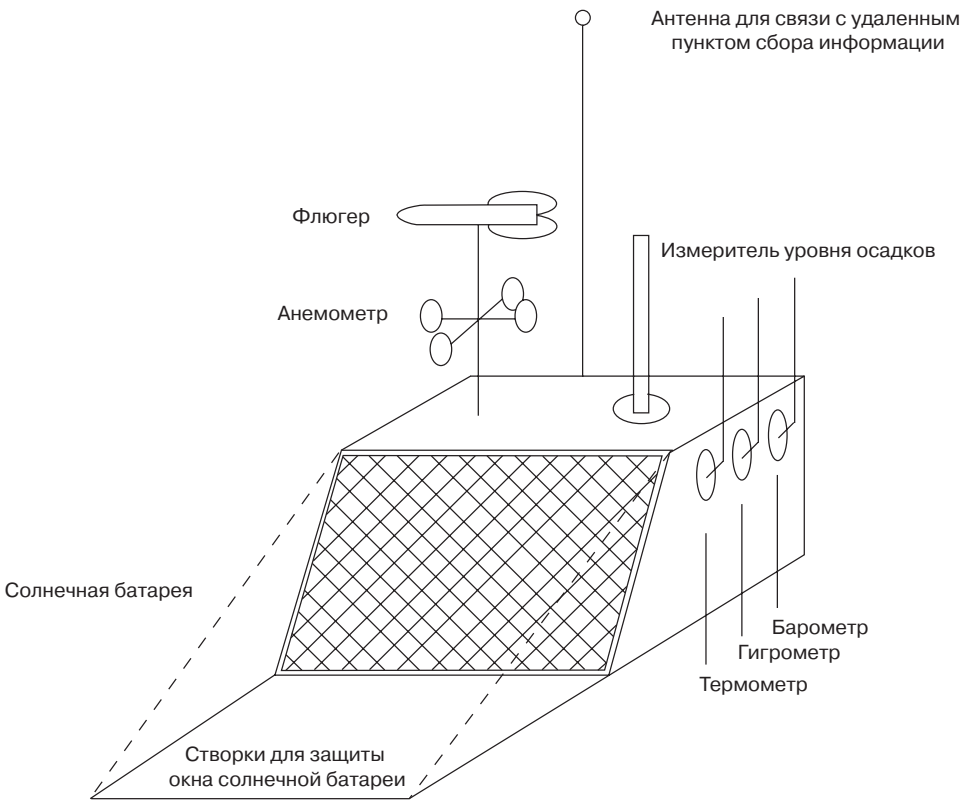


Рис. 2.10. *Портативная метеостанция*

характеристикой. Относительной влажности 0% соответствует напряжение 0 В, влажности 100% – напряжение 5,0 В.
- Термометром – прибором для измерения температуры воздуха. Температуре -50°С соответствует напряжение 0 В, температуре +120°С – напряжение 5,0 В. Передаточная характеристика преобразователя сигнала – линейная.
- Флюгером – прибором для определения направления ветра. Напряжение на выходе электронного преобразователя прямопропорционально углу поворота флюгера. Напряжение 0 В формируется при положении стрелки «север», далее

напряжение линейно возрастает, при положении стрелки «юг» напряжение равно 2,5 В, и на повороте в 180 град. эл. продолжает линейно возрастать до 5,0 В.

Проектируемая система должна уметь делать следующее:

- Станция должна периодически, один раз в 15 мин. передавать по радиоканалу информацию об измеренных значениях величин с выхода датчиков. Если скорость ветра превышает установленное значение, защитная панель станции должна автоматически закрывать окно солнечной батареи (рис. 2.10).
- Измеритель уровня осадков должен периодически очищаться.
- Кроме передачи информации об измеренных значениях с выхода датчиков, аналогичная информация должна отображаться на местном жидкокристаллическом индикаторе.

Разработайте структуру функций для управления такой станцией. Разработка передатчика не является Вашей задачей, Вы должны лишь подготовить собранные данные для передачи по радиоканалу.

3. Выберите один из домашних приборов, которые управляются встроенным контроллером. Разработайте структуру функций управления для одного из этих приложений. Выберите прибор из списка: микроволновая печь, холодильник, музыкальный центр.
4. Разработайте диаграмму действий UML для выбранного домашнего прибора.

Глава 3

ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ НА СИ

ПОСЛЕ ИЗУЧЕНИЯ ГЛАВЫ ВЫ СМОЖЕТЕ:

ЦЕЛИ

- Описать основные конструкции языка Си;
- Написать на Си простые программы для встроенных систем на основе микроконтроллеров;
- Объяснять последовательность действий, необходимую для получения исполняемого фрагмента кодов для МК 68HC12 и HCS12;
- Объяснить назначение отдельных программ в составе интегрированной среды разработки, позволяющие редактировать, компилировать, ассемблировать, объединять и отлаживать программные фрагменты управляющего кода будущих встраиваемых систем.

3.1. Введение в программирование на Си....71
3.2. Типы данных в Си....72
3.3. Операторы языка Си....76
3.4. Функции....82
3.5. Файлы заголовков....87
3.6. Директивы компилятора....88
3.7. Конструкции программирования....92
3.8. Операторы для организации программных циклов....93
3.9. Операторы принятия решения....95
3.10. Массивы....100
3.11. Указатели....102
3.12. Структуры....104
3.13. Процесс программирования и отладки микропроцессорной системы....106
3.14. Особенности компилятора и ассемблера....123
3.15. Заключение по главе 3....134
3.16. Что еще почитать?....134
3.17. Вопросы и задания....134

В главе 3 рассматривается технология написания и отладки программ на языке Си для микроконтроллеров 68HC12 и HCS12. На протяжении всей главы мы не будем делать различия между этими двумя типами МК и, как следствие, будем использовать для ссылок общую аббревиатуру 68HC12. Основное наше внимание будет уделено технике программирования на языке Си. Однако это не означает, что программирование на языке ассемблера может быть полностью забыто. Включение в исходный текст Си-программы фрагментов на языке ассемблера позволяет создать эффективный исполняемый код в критичных по времени исполнения задачах. Вопросы объединения программных фрагментов, записанных на Си и на ассемблере, также рассматриваются в данной главе.

Мы покажем, как из исходного текста программы, записанного на Си, создать файл исполняемого кода для микроконтроллера, используя для этого программы компилятора, ассемблера, линковщика и загрузчика в составе программного пакета ICC12 от компании Imagecraft (http:/www.imagekraft.com). Программный продукт ICC12 – достаточно простой в управлении, недорогой, но обладающий всеми необходимыми типовыми функциями программный пакет класса «интегрированная среда разработки и отладки программ управления для встраиваемых систем». Технология создания программного обеспечения для встраиваемых микроконтроллерных систем с использованием перечисленных выше программ в составе интегрированной среды разработки IDE (Integrated Development Environment) обладает достаточной степенью универсальности. Поэтому навыки создания и отладки программ для МК 68HC12, полученные с использованием пакета ICC12, могут быть использованы читателем при программировании 68HC12 или иных типов микроконтроллеров с использованием других более развитых программных пакетов IDE, например Code Warrior от компании Metrowerks (http:/www.metrowerks.com).

В этой главе мы обсудим технологию отладки программы, написанной на языке Си, в процессе ее исполнения реальным микроконтроллером. Все МК семейства 68HC12 обладают специальным режимом отладки в реальном времени BDM (Background Debug Mode). Мы рассмотрим основные свойства режима BDM в этой главе. В заключение мы приведем подробный пример преобразования исходного текста программы на Си в файл исполняемого кода для выбранного типа МК средствами программ, входящих в пакет интегрированной среды разработки IDE ICC12.

3.1. Введение в программирование на Си

В главе 2 было отмечено, что язык Си стал одним из наиболее часто используемых в техническом сообществе языков программирования высокого уровня. Связано это с тем, что при сохранении абстрактного характера представления алгоритмов и данных, которое свойственно языкам программирования высокого уровня, язык Си позволяет программистам непосредственно управлять компонентами аппаратных средств компьютеров и микроконтроллеров. Например, в Си программист имеет возможность непосредственного обращения к содержимому ячеек памяти и регистров периферийных модулей с конкретными физическими адресами, что свойственно языку ассемблер.

Для того чтобы начать программировать микроконтроллеры 68HC12 на Си, мы должны сначала изучить основные конструкции этого языка, которые и рассматриваются в следующих параграфах главы 3. Следует заметить, что в этих параграфах приведен лишь минимально необходимый набор сведений по Си. Для более полного знакомства с этим языком программирования читателю следует обратиться к специальной литературе. Список рекомендуемых авторами изданий приведен в конце главы.

Си – это структурированный язык программирования. Для записи программ на языке Си используются идентификаторы, ключевые слова, числа и операторы, которые располагаются в тексте программы в соответствие с предписанными правилами. Идентификаторы – это определяемые пользователем имена переменных, функций, меток и объектов. В языке Си для того, чтобы использовать переменную в программе, её необходимо предварительно создать. Создать переменную – это значит выделить для её хранения одну или несколько ячеек памяти и присвоить ей имя, с помощью которого можно будет ссылаться на эту. В отличие от ассемблера, в языке Си необходимо указать также тип переменной. Тип служит для того, чтобы сообщить компилятору о том, как интерпретировать значение, хранящееся в ячейках памяти, отводимых под переменную. Например, имеет ли число знак (старший бит отводится для хранения знака) или старший бит является значащим разрядом числа и т. д.

3.1.1. Глобальные и локальные переменные

Каждая объявленная в начале программы переменная должна храниться в одной или нескольких ячейках памяти в течение времени исполнения программы. В микроконтроллерах переменные могут размещаться в ОЗУ, ПЗУ и регистрах центрального процессора.

В языке Си различают два типа переменных: глобальные переменные, к которым можно обратиться из любой функции программы, и локальные переменные, которые доступны только при исполнении той функции, в которой они были объявлены. В системах на микроконтроллерах, локальные переменные обычно размещаются в области стека.

Объявление различных идентификаторов в начале программы – особенность языка Си по сравнению с языком ассемблер. Служебные слова, которые будут сопровождать объявление идентификатора, определяют правила доступа к переменной, функции, макросу или структуре, которая именована этим идентификатором. Не все идентификаторы будут доступны из любого места программы на Си. Подробное описание служебных слов для объявления классов хранения переменных выходит за рамки этой главы. Вы можете найти его в [3].

В микроконтроллерах семейства 68HC12, переменные размещаюися в ячейках памяти ПЗУ или ОЗУ. Переменная, которая не изменяет своего значения в течение выполнения программы, именуется константой и хранится в ПЗУ. Переменная, значение которой изменяется в процессе исполнения программы, должна храниться в ОЗУ. Например, предположим, что некоторый контроллер связан с датчиком температуры посредством аналого-цифрового преобразователя. Прикладная программа должна постоянно обновлять значение температуры в памяти контроллера. Для этого объявим переменную с именем temp (от слова temperature – температура) и разместим ее в ОЗУ МК. Приведенный ниже фрагмент программы на Си демонстрирует, как обратиться к размещенной в ОЗУ переменной temp и отобразить ее значение на дисплее. Заметим, что номера строк в приводимых фрагментах программ не являются частью записи операторов языка Си и не должны присутствовать в исходном тексте программы, подлежащем компиляции. Они введены искусственно для ссылки на отдельные операторы при обсуждении конструкций языка Си.

```
1      while (1)
2      {
3      temp = * (unsigned char volatile *) (0xl000);
4      printf ("The current temperature is %d\n",    temp);
5      }
```

Данный фрагмент кода предполагает, что переменная с именем temp создана (объявлена) ранее, и также ранее реализован фрагмент кода на Си для опроса аналого-цифрового преобразователя с целью обновления значения температуры. Для определенности в строке 3 мы сами присваиваем этой переменной значение $1000 в шестнадцатеричном коде (или 4096 в десятичной системе счисления). Префикс 0x служит в Си для обозначения шестнадцатеричной системы счисления. Не следует тревожиться, если не все записи в данном примере ясны для Вас. Пройдет немного времени, Вы закончите изучение материалов данной главы, и все рассмотренные примеры станут для Вас простыми и понятными.

В приведенном фрагменте переменная temp может быть объявлена как глобальная или как локальная переменная. Назначение статуса переменной (глобальная или локальная) определяет программист посредством записи оператора для объявления переменной в определенном месте текста программы. Глобальная переменная должна быть объявлена вне всех функций данной программы. В то время как локальная переменная объявляется внутри той функции, в которой используется. Мы рассмотрим способы объявления переменных после обсуждения в следующем параграфе типов данных, используемых в программах на Си.

3.2. Типы данных в Си

Язык Си оперирует с восемью основными типами данных, которые представлены в табл. 3.1. Тип переменной определяет число байтов в памяти микроконтроллера, которые выделяются компилятором для ее хранения.

Спецификация типа	Описание	Размер, байт	Допустимый диапазон чисел
char	Однобайтовое целое со знаком	1	-128 ÷ +127
unsigned char	Однобайтовое целое без знака или символ из набора символов системы ASCII	1	0 до 255
int	Целое значение естественного размера (машинное слово) со знаком	2	-32768 ÷ +32767
short int	Целое значение естественного размера (машинное слово) со знаком	2	-32768 ÷ +32767
unsigned int	Целое значение естественного размера без знака	2	от 0 до 65535
long int	Целое значение двойного естественного размера (два машинных слова) со знаком	4	-2147483648 ÷ +2147483647
float	Число в формате с плавающей запятой	4	±1,176E-38 ÷ ±3,40E+38
double	Не рекомендуется для использования	8	±1,7E-308 ÷ ±1,7E+308

Табл. 3.1. *Спецификация типов данных языка Си, используемых компилятором ICC12*

Область памяти, отводимая для хранения объекта типа char, всегда составляет один байт. Размер области памяти, отводимой для хранения объектов int и long, определяется типом используемого компилятора. Как правило, int – машинное слово длиною в 16 бит, long - двойное машинное слово, т.е. 32 бита. Формат и размер области памяти объектов float и double также зависит от типа используемого компилятора. В данном случае приведены значения для компилятора Си из среды разработки ICC12.

Служебные слово unsigned (без знака) используется вместе с целочисленными типами. Указание служебного слова unsigned перед спецификацией типа целочисленного значения определяет использование его старшего разряда. Для целых чисел со знаком старший разряд используется для хранения знака, что приводит к сужению диапазона модуля допустимых значений. В целых числах без знака старший разряд используется как дополнительный разряд числа, следовательно, диапазон допустимых значений расширяется. По умолчанию, т.е. без указания служебного слова unsigned, переменные типа char, int и long всегда считаются знаковыми. Числа в формате с плавающей запятой типов float и double также всегда знаковые.

В сочетании со спецификацией типа int возможно применение служебного слова short (короткий). Служебное слово short используется для сужения типа int до «короткого» целого. Применение служебного слова short имеет смысл, когда длина машинного слова микроконтроллера составляет четыре и более байтов. В этом случае спецификация типа short int соответствует целому числу, разрядностью в половину машинного слова. В случаях, когда машинное слово составляет два байта, спецификации типов int и short int совпадают (например, в 16-ти разрядном микроконтроллере HCS12). В некоторых компиляторах языка Си разрешается опускать слово int в спецификации типа short int и использовать одно слово short.

Слово long, обозначающее целое число, соответствующее двойному машинному слову, является таким же служебным словом как unsigned и применяется перед спецификацией типа int для его расширения до «длинного» целого (до двойного машинного слова). Однако в языке Си разрешается опускать слово int в спецификации типа long int и использовать одно слово long.

В дополнение к рассмотренным выше основным типам данных в языке Си существуют пять дополнительных типов данных:

- Array – массив, набор элементов одного типа;
- Pointer – указатель, переменная, которая содержит адрес переменной определенного типа;
- Structure – структура, набор элементов различного типа;
- Union – объединение, одна область памяти для двух различных типов данных;
- Function – функция, являясь сама определенным типом данных, может генерировать типы данных и возвращать типы данных.

Иногда эти пять типов данных характеризуют как производные типы, поскольку для их описания используются основные типы данных из табл. 3.1. Мы рассмотрим эти пять типов данных более подробно в этой главе, а также в главе 8.

В языке Си в простейшем случае синтаксис определения переменной или какого-либо другого объекта данных выглядит так:

```
<спецификация типа>    <идентификатор>
```

В более сложных случаях указывается также способ доступа к переменной и/или класс ее хранения:

```
<способ доступа/класс хранения> <спецификация типа> < идентификатор >
```

При объявлении массива может быть задано число элементов и их значения:

```
char a[10]
int m[] = {5,10,4000,34}
```

Первое поле, <способ доступа/класс хранения>, используется для задания типа памяти, куда должна быть помещена переменная. Второе поле, <спецификация типа>, содержит определение типа из табл. 3.1. Третье поле, <идентификатор>, содержит в себе придуманное программистом имя переменной. Четвертое поле, в котором проставлено численное значение или строка символов для массива, необязательное. Оно необходимо, если программист в строке объявления переменной желает также задать ее начальное значение, т.е. инициализировать переменную. Ниже приведен пример определения двухбайтовой переменной с именем «change» в целочисленном формате со знаком (тип int в соответствее с табл. 3.1):

```
int    change;
```

Точка с запятой в конце строки должна быть проставлена обязательно, поскольку она обозначает завершения выражения в тексте программы на Си.

Имя переменной может быть произвольным, но оно не должно совпадать с зарезервированными в языке Си служебными словами:

```
auto, break, case, char, const, continue, default, do, double, else,
enum, extern, float, for, goto, if, int, long, register, return,
short, signed, sizeof, v struct, switch, typedef, union, unsigned,
void, volatile, while
```

Шесть слов из приведенного списка используются в Си для определения класса хранения переменной:

```
extern, auto, static, register, const, volatile
```

Программист может управлять размещением данных, поместив их либо в сегмент данных, либо в сегмент стека. Осуществляется это путём размещения объявлений данных в различных частях модуля программы на языке Си, что определяет так называемый класс хранения объекта, который в свою очередь задаёт время его жизни и область действия.

К внешнему классу хранения (extern) относятся переменные, определения которых в тексте программы размещены вне текста некоторого программного модуля, т.е. в другом файле. Переменные с внешним классом хранения размещаются в сегменте данных (т.е. в ОЗУ микроконтроллера) и сохраняются в течение всего времени выполнения программы. Областью действия переменной с внешним классом хранения является вся программа. Т.е. такая переменная доступна во всех функциях данного модуля и во всех других модулях программы.

Служебное слово auto определяет автоматический класс хранения переменной. Если при определении переменной указание на ее класс хранения отсутствует, т.е. ни одно из служебных слов не предшествует спецификации типа переменной, то по умолчанию переменной назначается автоматический класс хранения. Областью действия переменных с автоматическим классом хранения является функция (блок), в которой они определены, а также все вложенные функции (блоки). Автоматические переменные размещаются в сегменте стека на время выполнения функции, в которой они определены. После окончания выполнения функции память, отводимая под эти переменные (стек) освобождается и может быть использована для размещения других переменных. Таким образом, время жизни переменных с автоматическим классом хранения является время выполнения функции, в которой они определены. Эти переменные доступны для чтения или изменения только в тех функциях, в которых определены.

Служебное слово static в строке определения переменной обозначает статический класс ее хранения. Переменные с этим классом хранения имеют только одно отличие от переменных с автоматическим классом хранения. Статические переменные размещаются в сегменте данных (ОЗУ микроконтроллера) и сохраняют свои значения в промежутках между выполнением функций или блоков, в которых они определены. Таким образом, время жизни данных, относящихся к статическому классу хранения, совпадает с временем выполнения всей программы.

Если в определении переменной указано служебное слово register, то компилятор постарается создать код так, чтобы переменная размещалась в регистре центрального процессора. В основном это необходимо для увеличения быстродействия. Если регистры заняты (уже все используются), то переменная будет размещена в стеке, аналогично автоматическому классу хранения.

Служебное слово const указывается в определении переменных, значения которых не изменяются на протяжении исполнения программы. Эти переменные обычно помещаются в область ПЗУ микроконтроллера. Следует обратить внимание, что в некоторых аппаратных средствах отладки для инициализации начального значения переменной типа const должен быть использован режим программирования постоянной памяти микроконтроллера.

Класс хранения volatile назначается для тех переменных, которые могут изменять свое значение не в результате действия программы, а под управлением аппаратных средств микроконтроллера. Мы рассмотрим примеры с использованием переменных класса volatile при обсуждении приложений с использованием периферийных модулей МК 68HC12.

В процессе определения переменной, которое назначает область памяти и число байтов для ее хранения, можно также установить начальное численное значение этой переменной, т.е. инициализировать переменную. Например, определение целочисленной константы «change» со значением 23 может быть проведено посредством записи;

```
const int change = 23
```

После того, как мы научились задавать необходимые для использования в прикладной программе переменные, следует научиться выполнять арифметические и логические операции над этими переменными. Этим мы займемся в следующем параграфе.

3.3. Операторы языка Си

Язык Си обладает некоторым набором операторов, которые представлены в табл. 3.2. Полное множество операторов разбито на пять групп: общие, арифметические, логические, битовые манипуляции, унарные.

Операторы общей группы предназначаются для записи выражений на языке Си. Арифметические операторы предназначены для выполнения математических действий над переменными, таких как сложение, вычитание, умножение и деление. Логические операторы используются в выражениях для определения истинности некоторого условия. Битовые операции предназначены для модификации одного или нескольких битов переменной. Унарные операции используются для выполнения действий только над одной переменной.

В первой колонке табл. 3.2. указан приоритет оператора в том случае, если выражение содержит сразу несколько операторов. Приведем примеры последовательности применения операторов.

Операторы общей группы. Круглые скобки из группы общих операторов используются для определения порядка выполнения действий над операндами. Допустим, в строке программы записано следующее выражение:

2 * 23 + 15

Приоритет в выражениях	Имя оператора	Символ для обозначения
Общие		
1	Скобки	(), {}
1	Разделители	- > , .
11	Условие	?:
12	Присваивание	=, +=, *= и т.д.
Арифметические		
3	Умножение	*
3	Деление	/
3	Получение целочисленного остатка от деления	%
4	Сложение	+
4	Вычитание	-
Логические		
6	Меньше	<
6	Меньше или равно	< =
6	Больше	>
6	Больше или равно	> =
7	Равно	= =
7	Не равно	! =
9	Логическое И	&&
10	Логическое ИЛИ	‖
Битовые манипуляции		
5	Сдвиг влево	<<
5	Сдвиг вправо	>>
8	Поразрядное И	&
8	Поразрядное исключающее ИЛИ	^
8	Поразрядное ИЛИ	|
Унарные		
2	Инверсия	!
2	Взятие обратного кода числа	~
2	Инкремент	++
2	Декремент	- -
2	Минус	-
2	Привести к типу	(type)
2	Указатель	*
2	Взять адрес	&
2	Определить размер	sizeof

Табл. 3.2. *Операторы языка Си*

Оператор умножения имеет приоритет над оператором сложения, поэтому результат вычисления выражения будет равен 61. Какие изменения мы должны внести в выражение, если хотим сначала сложить 23 и 15, и лишь затем умножить сумму на 2? Для изменения порядка действий над операндами мы воспользуемся круглыми скобками:

```
2 * (23 + 15)
```

И результат вычислений станет равным 76.

Фигурные скобки следует использовать для объединения некоторого множества операторов в законченный смысловой блок, например в функцию или цикл «loop». Примеры использования фигурных скобок, а также других операторов из группы общих рассматриваются в разделе 3.4.

Операторы группы арифметических операций. Рассмотрим последовательность выполнения действий над операндами при исполнении микроконтроллером следующего выражения:

```
Sum = 2 + 3;
```

Для вычисления значения переменной sum сначала реализуется оператор сложения «+», а затем оператор присваивания «=». В колонке 1 табл. 3.2 отражено, что оператор сложения имеет приоритет над оператором присваивания. При этом необходимо, чтобы переменная sum ранее была определена как int. Что произойдет в случае, если переменная sum ранее была объявлена как переменная другого типа, например float? После сложения результат будет преобразован к тому формату представления числа, который был объявлен при определении переменной sum.

Рассматриваемый пример может быть реализован с использованием другого синтаксиса:

```
num1 = 2;
num2 = 3;
sum = num1 + num2;
```

Подразумевается, что все упомянутые переменные num1, num2 и sum ранее были определены.

Операторы инкремента и декремента производят действия только над одной переменной. При этом, три приведенных ниже записи, отличаются синтаксисом выражения, но производят одинаковые действия:

```
number = number + 1;
number++;
++number;
```

Аналогично, три следующих записи реализуют операцию декремента, т.е. уменьшения на единицу переменной number.

```
number = number - 1;
number--;
--number;
```

Операция получения целочисленного остатка от деления 2%3 возвращает 2, так как целочисленное деление 2 на 3 не может быть произведено. Результат операции 14%3 также равен 2, поскольку результат целочисленного деления 14 на 3 равен 4 с остатком 2.

Операторы логической группы. Операторы этой группы используются для определения условий, по которым реализуется ветвление алгоритма. Операторы логической группы возвращают в виде результата 1, если результат операции «правда», и 0, если результат операции «ложь». Допустим, мы хотим сравнить текущее значение некоторой переменной с пороговым значением 82. Для этого могут быть использованы операторы больше «>», меньше «<», больше или равно «> =», меньше или равно «< =», не равно «! =» или равно «= =». Рассмотрим следующую запись на Си:

```
value = temperature > 82;
```

После исполнения приведенной строки программы переменной value будет присвоено значение 1 или 0. Уместно вспомнить, что логические операции имеют приоритет над операцией присваивания. Поскольку результатом «вычисления» выражения справа может быть только 0 или 1, то и переменная value должна быть ранее объявлена соответствующим образом.

Операторы группы битовых манипуляций. Как было отмечено ранее, одним из преимуществ языка Си для программирования микроконтроллерных систем по сравнению с другими языками высокого уровня, является возможность непосредственного изменения данных в ячейках памяти, например с использованием оператором побитового логического И, ИЛИ и Исключающего ИЛИ. Самый простой пример применения операций сдвига - это умножение и деление числа на число 2n . Рассмотрим результат выполнения следующих трех операторов:

```
number = 24;
new_number_one = number << 1;
new_number_two = number >> 1;
```

Допустим, что три используемые в примере переменные определены как int. В первой строке переменной number присваивается значение 24 в десятичной системе счисления. Это же значение в двоичной системе счисления будет равно 00000000 00011000. Результатом действия оператора «<<» будет сдвиг влево на один разряд значения переменной number , т.е. 00000000 00110000 или 48 в десятичной системе счисления. Это значение и будем присвоено переменной new_number_one. В третьей строке оператор «>>» реализует сдвиг вправо числа number. Получится новое двоичное число 00000000 00001100 или 12 в десятичной системе счисления. В результате, значение переменной new_number_one будет равно удвоенному значению переменной number, в то время как переменная new_number_two будет равна поделенному на 2 значению number. С использованием рассматриваемых операторов мы можем также выполнить сдвиг на несколько разрядов, тогда результат операции будет эквивалентен умножению или делению на 2^n. Например, если n = 3, то после выполнения следующих трех операторов:

```
number = 24;
new_number_one = number << 3;
new_number_two = number >> 3;
```

значение переменной new_number_one будет равно 192 (двоичный код 00000000 11000000), а значение переменной new_number_two – 3 (двоичный код 00000000 00000011).

Рассмотрим два других логических оператора: поразрядное логическое И и поразрядное логическое ИЛИ.

Символ	Операция	Пример
&	Логическое И	*(0x0023) & 0x57
|	Логическое ИЛИ	*(0x0000) | 0x35

Все числа, записанные в колонке «Пример», представлены в шестнадцатеричном коде, поскольку содержат префикс 0x. Унарный оператор * показывает, что действие будет производиться над содержимым ячейки памяти с физическим адресом, значение которого в шестнадцатеричном коде указано в скобках.

Результат операции логического И над двумя двоичными числами 01011100 и 11000111 будет равен :

```
    01011100
&   11000111
----------------------------
    01000100
```

Результат операции логическое ИЛИ над теми же числами:

```
    01011100
|   11000111
----------------------------
    11011111
```

В каких задачах управления используются эти логические операторы? В прикладных программах (т.е. программах управления) часто приходится изменять сигналы на отдельных линиях портов ввода/вывода. Регистры данных портов расположены по строго определенным в техническом описании физическим адресам. Так для того, чтобы сконфигурировать все линии порта PORT A на ввод, необходимо в регистр направления передачи порта DDRA (физический адрес 0x0002) записать все нули. Это может быть выполнено под управлением следующей стоки:

```
* (unsigned char volatile *) (0x0002) = 0x00;
```

Если порт Port A настроен на вывод, то установить линию PTA7 в единицу без изменения состояния остальных линий порта можно посредством следующей записи:

```
PORTA | = 0x80;            //установить PTA7
```

Выше использована сокращенная форма записи выражения:

```
PORTA = PORTA | 0x80;      //установить PTA7
```

Выражение возвращает результат операции поразрядного логического ИЛИ числа 0x80 (10000000 в двоичной системе счисления) и содержимого порта PortA. После операции старший бит Port A будет установлен в 1, остальные биты останутся без изменения.

Аналогично, старший бит порта Port A может быть установлен в 0 (сброшен) посредством записи выражения:

```
PORTA & = ~ 0x80;          //сбросить бит PTA7
```

Это выражение аналогично другому, более понятному для начального уровня освоения языка Си:

```
PORTA = PORTA & 0x7F;          //сбросить бит PTA7
```

Для установки в 0 старшего разряда порта Port A содержимое порта побитно логически умножается на константу 0x7F (01111111 в двоичном коде). В результате старший бит становится равным 0, а остальные биты остаются без изменения. Запись ~ 0x80 в первом выражении предписывает перед выполнением операции логического И взять инверсию константы 0x80 (10000000), которая будет равна 0x7F (01111111). Вторая запись более понятна на начальном этапе программирования на Си, в то время как первая запись позволяет использовать одну и ту же константу в выражениях по установки и сбросу бита, что в практическом программировании удобно.

Операцию поразрядного логического И также следует использовать, если необходимо проверить, установлены или сброшены биты порта с определенными номерами. Например, приведенный ниже фрагмент программы производит чтение регистра данных порта Port A, логически умножает его содержимое на константу 0x81 и сравнивает полученный результат с нулем. Если условие равенства нулю выполняется, то это означает, что биты 7 и 0 порта Port A одновременно равны нулю, и следует выполнить действия, которые описаны операторами в фигурных скобках. Если хотя бы один бит PTA7 или PTA0 не равен нулю, то условие ((PORTA & 0x81) = = 0) не выполняется, и операторы в фигурных скобках будут пропущены при исполнении.

```
it ((PORTA & 0x81) = = 0
{
:
}
```

В качестве примера использования оператора ИСКЛЮЧАЮЩЕГО ИЛИ приведем выражение для инвертирования значения бита 7 порта Port A:

```
PORTA ~ =  0x80;          //инвертировать бит PTA7
```

Операторы группы унарных операций. Поскольку операторы инкремента и декремента были рассмотрены выше, основное внимание уделим операторам указателя и косвенной адресации (см. табл. 3.2). Для иллюстрации действия этих операторов рассмотрим следующий пример. Определим три целочисленных переменных с именами num, address, и new_num:

```
Int    num, address, new_num;
```

Также предположим, что переменная num расположена в памяти по адресу 0x2000. Запишем следующее выражение:

```
address = &num;
```

Результатом исполнения выражения будет присвоение переменной address значения адреса переменной num , т.е. новое значение переменной address будет равно 0x2000.

Запишем новое выражение:

```
new_num = *address;
```

Результатом выполнения этого выражения будет присвоение переменной new_num значения, которое содержится в ячейке памяти, адрес которой равен текущему значению переменной address. Поскольку содержимое address равно 0x2000, т.е. адресу переменной num, то рассматриваемое выражение в нашем случае эквивалентно выражению:

```
new_num = num;
```

Несмотря на то, что в рассмотренных примерах используется корректный синтаксис, в стандарте ANSI C переменную, в которой будут храниться адреса, используемые в качестве указателей на ячейки памяти других переменных, следует определять следующим выражением:

```
int    *addres
```

Отличие от предыдущего способа определения состоит в том, что теперь компилятор самостоятельно определяет формат представления данных для переменной address, чтобы в этой переменной было бы возможно разместить численное значение адреса. Если бы в предыдущем случае программист ошибся и определил тип переменной address как char, то в процессе исполнения выражения address = &num возникла бы потеря информации. В последнем случае ошибка формата исключается.

Обсудим действие следующего выражения:

```
address = (int *) 0x1000;
```

Это выражение назначает ячейку памяти с адресом 0x1000 как указатель с именем address. Для того, чтобы извлечь содержимое ячеек памяти следует поместить оператор * перед именем address.

3.4. Функции

В этом параграфе мы познакомимся с Вами с понятием «функция». Мы покажем Вам, как в языке Си определить функцию, как передать в функцию численные значения параметров и как получить после выполнения функции рассчитанные ею значения переменных.

3.4.1. Что такое функция?

Функция – это независимый фрагмент исходного текста программы, предназначенный для решения некоторой задачи. Функции состоят из операторов языка Си и представляют собой обычные подпрограммы.

Представьте себе, что Вы работаете членом большой бригады инженеров, которой предстоит разработать программное обеспечение для встраиваемых систем самолета. Очевидно, что на начальной стадии проекта вся команда разработчиков должна пройти через те этапы структурного проектирования, которые были рассмотрены нами в главе 2. На завершающем этапе структурного проектирования, когда одна большая задача будет поделена на множество мелких, но функционально законченных фрагментов, каждому из членов команды будет поручено выполнение какого-либо фрагмента общей программы. Эти фрагменты оформляются как функции, из которых впоследствии будет состоять большая программа.

В соответствие с приведенной стратегией составления большой программы, каждая функция должна обладать тремя свойствами: независимостью, гибкостью и переносимостью. Функция должна быть относительно независима от другого программного кода, поскольку эта функция в дальнейшем может быть использована различными программистами в данном проекте или даже в другом проекте. Возвращаясь к примеру, предположим, что Вам предложили написать функцию, которая устраняет шумовую составляющую входного аналогового сигнала (цифровой фильтр). Ваша программа цифрового фильтра будет использоваться многими соисполнителями проекта для устранения шума различных входных сигналов. Следовательно, Ваша программа, оформленная как функция, должна обеспечивать возможность ее вызова из любого места большой программы (свойство независимости) и должна легко настраиваться на прием сигнала с различных портов МК (свойство гибкости). Термин «относительно независима» в начале этого параграфа мы применили потому, что функция может получать от ранее исполненного программного кода некоторые численные значения, которые будет использовать при своей работе. Например, в Вашу функцию могут передаваться имя порта ввода и номер линии, на которой присутствует сигнал, который подлежит цифровой фильтрации. Вы значительно увеличите гибкость своего решения, если предусмотрите возможность изменения частотного диапазона шумовой составляющей сигнала, которая будет устранена после исполнения программного кода Вашей функции.

И, наконец, о свойстве переносимости. В рассматриваемом примере управления самолетом, многочисленные встраиваемые системы могут быть выполнены на разной элементной базе, в том числе на МК с различным процессорным ядром, и даже от разных производителей. При этом крайне желательно, чтобы разработанные специалистом функции цифровой фильтрации могли быть использованы во всех решениях. Поэтому следует написать исходный текст программы на Си таким образом, чтобы он не был ориентирован на специальные команды процессорного ядра, и мог быть обработан любым компилятором языка Си из перечня используемых в проекте.

3.4.2. Основная программа

Основная программа – это особый тип функции, ее отличие от других функций заключается в том, что она исполняется, когда запускают программу с определенным именем. Корректно написанная основная программа работает как программа «управленец» (менеджер высшего звена). Текст основной функции main.c отражает структуру всей прикладной программы, при этом не затрагивая специфических особенностей отдельных задач по управлению объектом. Мы можем представить основную программу в роли управляющего, который контролирует выполнение отдель-

ных «команд» управления путем запуска программ-функций. Предположим, что мы хотим выполнить задачи с первой по n-ую. Для этого оформлены n функций. Тогда мы запишем основную программу, в которой будут последовательно вызываться эти функции:

```
1       void   main(void)
2       {
3       function_one();
4       function_two();
5       function_three();
6               :
7               :
8               :
9       function_n();
10      }
```

Строка 1 содержит идентификатор функции main. Служебное слово void в круглых скобках информирует о том, что данная функция не требует входных аргументов. Строки с третьей по девятую содержат операторы вызова функций, причем каждая функция вызывается один раз. После завершения исполнения функции 1 начинается исполнение функции 2, затем функции 3 и так до конца программного фрагмента.

3.4.3. Прототипы функций

Любая функция перед тем, как в тексте программы будет записан ее программный код или оператор ее вызова, должна быть объявлена. Объявление функции в языке Си называют прототипом функции. Формат записи прототипа функции следующий:

```
<тип возвращаемой переменной> <имя функции>
                                    (<тип переменной1> <имя переменной1>,
                                    <тип переменной2> <имя переменной2>,
                                           :
                                    <тип переменнойN> <имя переменнойN>;
```

Имена переменных в круглых скобках приводить необязательно, но полезно для лучшего документирования. Обратите внимание на обязательное использование точки с запятой в конце записи прототипа. Приведем три примера прототипов функций:

```
Пример 1:     int compute(int, int);
Пример 2:     float change(char name, float number, int a);
Пример 3:     double find(unsigned int, float, double);
```

В примере 1 функция compute использует два аргумента. Аргументы функции - это те переменные, которые необходимы для ее корректного исполнения, Спецификация типа аргументов функции приведена в круглых скобках. В данном случае указано, что функция compute будет использовать два целочисленных аргумента, т.е. при вызове функции ей должны быть указаны для целочисленных значения. Результатом действия функции compute будет вычисление значения некоторой перемен-

ной. Спецификация типа возвращаемой переменной приведена перед именем функции. В данном случае это тоже целочисленный 16-разрядный формат.

В примере 2 записан прототип функции change. Эта функция предполагает наличие трех аргументов: однобайтового целочисленного name, числа с именем number в формате с плавающей запятой и двухбайтового целого числа с именем a. Функция change должна возвратить значение переменной в формате с плавающей запятой. В примере 3 объявляется функция find с тремя аргументами, для которых указан тип данных, но не указаны имена.

При знакомстве с программами на Си Вы можете встретить прототип функции, в котором на первом месте указано слово extern:

```
extern not_here(int a, int b, int c)
```

Подобная запись означает, что определение функции not_here не включено в текст данного программного модуля (текущего файла), а располагается во внешнем модуле (другом файле), который будут объединен с текущим модулем при составлении конечного исполняемого кода программы.

Если функция была объявлена в тексте какого-либо программного модуля, то она должна быть определена в этом же модуле или другом модуле, например в файле библиотеки. Мы обсудим правила определения функции в следующем параграфе, а пока приведем примеры вызова каждой из трех объявленных функций:

```
compute(23, 12);
change('b', 7.825, 2);
find(25, 5.1524, 23.54721);
```

Обратите внимание, спецификация типа возвращаемой переменной в записи вызова функции исчезла, а в поле аргументов появились записи их численных значений.

3.4.4. Описание функций

Каждая объявленная в начале некоторого программного модуля функция должна быть определена в этом модуле или в тексте программы другого модуля, который в процессе генерации исполняемого кода программы будет присоединен к текущему модулю. Функция может быть также определена в подключаемом файле стандартной библиотеки. Текст определения функции может быть записан в любом месте программного модуля, однако принято определения всех используемых функций располагать сразу за текстом основной программы main.c. Например, предположим, что объявленная в предыдущем параграфе функция compute вычисляет модуль вектора двух ортогональных составляющих a и b и возвращает его в переменной с именем result. Прототип функции:

```
int compute(int a, int b);
```

Определение функции начинается с комментария, в котором перечисляются производимые функцией действия. Далее следует повтор строки прототипа функции, но без заключительных точки с запятой. Ниже последовательно располагаются все операторы функции, которые сверху и снизу заключаются в фигурные скобки.

```
/*Функция compute: вычисляет модуль вектора по двум его ортогональным */
/*составляющим                                                          */

1       int compute(int a, int b)
2       {
3       int sum, result;
4       sum = a*a + b*b;
5       result=(int) (sqrt(sum));
6       return(result);
7       }
```

В приведенном примере строка 1 открывает определение функции, информируя компилятор о том, что имя функции compute, она использует для своей работы две целочисленных переменных и возвращает одну целочисленную переменную. Фигурная скобка в строке 2 открывает область операторов определяемой функции. В строке 3 объявляются локальные переменные, т.е. те переменные, которые используются только внутри функции. Это переменные sum и result. Операторы, расположенные в строках 4 и 5 выполняют заявленные в описании функции вычисления. Причем в строке 5 используется функция извлечения квадратного корня sqrt, которая определена в библиотеке математических вычислений. При компиляции эта библиотека должна быть обязательно присоединена к файлу с рассматриваемой функцией посредством специальных директив компилятора, которые мы рассмотрим в следующем параграфе. В строке 5 следует обратить внимание на оператор (int) перед вызовом функции извлечения квадратного корня. Этот оператор осуществляет преобразование типа данных к заявленному в прототипе функции compute целочисленному формату int, поскольку функция извлечения квадратного корня возвращает данные в другом формате. В строке 6 применен оператор возврата return, которые позволяет использовать значение переменной result другими операторами основной программы. Фигурная скобка в строке 7 завершает определение функции. Любые операторы, записанные после скобки, уже не будут ассоциироваться с функцией compute.

На основании анализа примера Вам следует запомнить, что каждая функция должна быть определена в строго заданном формате исходного текста программы. Сначала следует строка прототипа функции, в которой указывается имя функции, используемые переменные и возвращаемые переменные. В отличие от строки объявления функции, точка с запятой в конце строки прототипа при определении функции не ставятся. Затем следуют операторы функции, заключенные в фигурные скобки. Если в поле возвращаемой переменной прототипа указан ее тип, то последним оператором функции должен быть оператор return. Если же в поле возвращаемой переменной прототипа стоит служебное слово void, то функция не возвращает данных. Назначение такой функции – выполнить определенный набор действий по управлению периферийными модулями МК или внешними устройствами. Соответственно и оператор return в последней строке отсутствует.

3.4.5. Вызов функций, передача параметров, возврат полученных значений

Если функция объявлена и определена, то она может быть вызвана из любой части программного модуля. Для вызова функции необходимо записать ее имя и в круглых скобках указать значения параметров, если таковые присутствовали в прототи-

пе функции. Например, для вызова функции compute с двумя параметрами запишем выражение:

```
magnitude = compute(12, 24)
```

В этом примере мы предполагаем, что переменная magnitude ранее была объявлена как целочисленная. После того, как функция была вызвана и выполнена, переменной magnitude будет присвоено значение 26. Истинный результат вычисления равен 26,832816. Именно это значение будет вычислено функцией извлечения квадратного корня sqrt. Однако перед присвоением переменной result этого значения производится смена формата представления данных, и дробная часть результата отбрасывается.

Для того, чтобы функция могла произвести требуемые вычисления, ей должны были быть переданы два параметра, которые мы указали в круглых скобках. Число параметров и формат их представления указываются в прототипе функции. В прототипе функции compute было также заявлено, что функция возвращает одно целочисленное значение. А может ли функция вернуть сразу несколько значений? Да может, если Вы корректно запишете прототип и определение функции. Для этого необходимо познакомиться с понятиями указателя и структуры, которые мы рассмотрим несколько позже. А пока лишь заметим, что функция может возвратить некоторый набор вычисленных значений посредством возврата начального адреса этих данных в памяти. Такой подход предполагает, что следующие программные фрагменты должны «знать» последовательность расположения данных в строке с указанного адреса, т.е. знать структуру представления данных в строке.

3.5. Файлы заголовков

В этом параграфе мы расширим наши знания о технике программирования на Си посредством знакомства с заголовочными файлами (header file). Заголовочный файл – это внешний файл, помещаемый в начало программы с помощью директивы #include, обычно содержащий определения типов и переменных, используемых в программе. Язык Си предоставляет программисту некоторый набор стандартных функций, определения которых находятся в нескольких заголовочных файлах. Например, в созданной нами функции compute мы использовали функцию извлечения квадратного корня sqrt, которая определена в файле математических функций math.h.

Выражения языка С для включения файлов заголовков в модуль разрабатываемой программы обычно располагаются в начале программы. Заголовочные файлы содержат определения переменных, макросы, объявления функций, позволяя программисту возможность вызывать эти функции, использовать переменные и макросы без дополнительного определения их в тексте создаваемой программы. В процессе компиляции значения постоянных переменных замещают их символьные значения, упомянутые в основной программе. Далее в примерах программ мы будем использовать заголовочный файл stdio.h, в котором определены функции библиотеки стандартного ввода/вывода. Эти функции позволяют отобразить результаты преобразования данных в МК на экране дисплея. А также передать в МК код нажатой клавиши на клавиатуре. Поставляемые фирмами производителями программного обеспечения компиляторы уже содержат

библиотеки и соответствующие им заголовочные файлы. Например, нами будет использована библиотека математических функций, и соответствующий ей файл math.h. Во многих случаях у пользователя возникает необходимость создания своего собственного заголовочного файла, в котором будут содержаться определения констант.

Для того чтобы включить файл заголовка в разрабатываемый программный модуль, следует воспользоваться директивой #include. Приведем три примера:

```
#include <stdio>
#include <math.h>
#include «myheader.h»
```

В первых двух записях имя подключаемого файла заключено в «< >», что информирует компилятор о том, что названные файлы располагаются в определенной директории (папке). Обычно это папка с именем include, которая располагается в основном каталоге компилятора Си. В третьей записи имя подключаемого файла заключено в двойные кавычки. Для компилятора это означает, что данный файл располагается в той же папке, что и создаваемый программный модуль.

3.6. Директивы компилятора

Директивы компилятора – это инструкции для программы компилятор, которые указывают ему каким образом следует обрабатывать исходный текст программы. Достаточно часто эти инструкции называют директивами препроцессора компилятора, акцентируя внимание пользователей на том, что эти директивы выполняют обработку исходного текста программы перед тем, как компилятор начнет генерацию ассемблерного текста программы. Известно 11 директив компилятора Си: #if, #ifdef, #ifndef, #else, #elif, #include, #define, #undef, #line, #error, #pragma. Из приведенного списка понятно, что директивы отмечаются символом # в первом знаке имени. Далее мы рассмотрим наиболее часто используемые директивы.

3.6.1. Директива условной компиляции

Директивы #if, #ifdef, #ifndef, #else, #elif и #endif относятся к группе директив условной компиляции. Эти директивы используются для того, чтобы обозначенный фрагмент исходного текста программы можно было бы включать или не включать в компилируемый код в зависимости от выполнения некоторого наперед заданного условия. Такое действие может быть полезным, например, в процессе отладки программы. Тогда в отладочной версии программы промежуточные результаты вычислений будут выводиться на экран дисплея, в рабочей версии эти действия выполняться не будут.

Директивы #if и #endif обязательно используются вместе, чтобы обозначить начало и конец условно компилируемого текста программы. В строке с директивой #if записывается условие компиляции. Если это условие выполняется, то выражения, записанные в программе между директивами #if и #endif включаются в компилируемый текст программы. В противном случае эти выражения исключаются из генерированного кода программы. Например, в процессе отладки мы хотим вывести на экран указанную в тексте программы фразу:

```
1       #include <stdio.h>
2       #define DEBUG 1
3       void main(void)
4.      {
                :
                :
m       #if DEBUG
m+1     printf{"The program reached this point in the program\n"};
m+2     #endif
                :
                :
n       }
```

Для этого в строке 2 мы присвоили переменной DEBUG значение 1, используя директиву препроцессора #define. Эту директиву мы обсудим несколько позже, а пока констатируем, что единичное значение переменной DEBUG соответствует условию «истина» в строке m с директорией #if. Поэтому вызов функции prinf, записанный в строке m+1, будет включен в компилируемый текст программы. При ее исполнении мы увидим строку «The program reached this point in the program» на экране монитора. Обратите внимание, что точка с запятой в конце строки с директорией не ставится.

В рассмотренном выше примере предполагалось только две возможности: включать или не включать в исполняемый код программы определенный фрагмент. Директивы #else и #elif расширяют возможности условной компиляции и позволяют выбрать для компиляции один из нескольких фрагментов текста. Например:

```
1       #define M68HC11 0
2       #define M68HC12 1
3       #define M8051 2
4       #define Processor 1
5       void main(void)
6       {
7               #if Processor == M68HC11
8               Instruction(s) A
9       #elif processor == M68HC12
10              Instruction(s) B
11      #elif processor == M8051
12              Instruction(s) C
13      #else
14              Instruction(s) D
15      endif
16
```

В этом примере исходный текст программы написан таким образом, что он может быть компилирован для исполнения различными микроконтроллерами: Motorola HC11, Motorola HC12 и Intel 8051. Директивы #if, #elif и #else позволяют для данного сеанса компиляции выбрать конкретный тип МК. Для этого в строках 1...3 программы каждому символьному имени МК присвоено определенное численное значение. Далее в зависимости от выбранного типа МК переменной Processor присваивается желаемое значение. В примере мы собираемся компилировать программу для МК HC12, поэтому присвоили переменной Processor значение 1. В строке 7 ком-

пилятор проверяет истинность выражения, записанного в качестве условия директивы #if. Это условие не выполняется, поскольку Processor = 1 ≠ M68HC11 = 0. Поэтому группа инструкций Instruction(s) A не будет включена в программу. Далее в строке 9 компилятор обнаружит выполнение условия директивы #elif, и выражения Instruction(s) B будут присутствовать в конечном варианте программы. Условие строки 11 не выполняется, и группа инструкций Instruction(s) C в исполняемом коде программы присутствовать не будет. Если ни одно из условий для директив #elif не выполнено, то выражения, следующие за директивой #else, будут включены в программу автоматически.

Воспользуемся приведенной конструкцией условной компиляции. Допустим, мы предполагаем исполнение некоторого программного кода как на МК семейства Motorola HC11, так и на МК семейства Motorola 68HC12. Эти МК имеют различные карты памяти, и, соответственно, их порты ввода/вывода расположены по различным адресам. Для возможной адаптации текста программы к одному из типов МК воспользуемся директивами условной компиляции:

```
1      #if (Processor == 68HCll)
2            #define PORTA *(unsigned char volatile *) (0xl000)
3      #elseif (processor == 68HC12)
4            #define PORTA *(unsigned char volatile *) (0x0000)
5      #endif
```

В строках 1 и 3 располагаются директивы, которые проверяют условия компиляции. Значение переменной Processor должно быть определено выше по тексту программы директивой #define, или в подключаемом заголовочном файле. Строки 2 и 4 содержат директивы определения адреса для порта PORTA для двух различных типов МК. Директива #endif в строке 5 отмечает окончание фрагмента текста, который подлежит условной компиляции.

Директивы #ifdef и #ifndef используются для организации процесса компиляции при условии, что некоторая переменная с указанным именем была определена (#ifdef) или не определена (#ifndef) в тексте программы. Например:

```
1      #ifdef OUTPUT
2            Instruction(s) A
3      #else
4            Instruction(s) B
5      #endif
```

Если переменная с именем OUTPUT была определена в тексте программы до строки 1 с директивой #ifdef, то группа инструкций Instruction(s) A будет включена исполняемый код программы. В противном случае в конечный вариант программы будет включена группа инструкций Instruction(s) B.

Другой пример:

```
1      #ifndef OUTPUT
2            Instruction(s) A
3      #else
4            Instruction(s) B
5      #endif
```

Если переменная с именем OUTPUT не была определена в тексте программы до строки 1 с директивой #ifndef, то в конечный вариант программы будет включена группа инструкций Instruction(s) A. Если же эта переменная была определена ранее, то исполняемый код программы будет включена группа инструкций Instruction(s) B.

Директива #define используется в двух случаях. Во-первых, она позволяет задать численные значения для символьных констант. Например, константе с именем HIGH необходимо присвоить значение 98:

```
#define HIGH 98
```

После записи этого выражения, если в тексте программы будет использовано имя HIGH, то при компиляции оно будет заменяться числом 98. Это удобно, поскольку в тексте программы имя HIGH может быть упомянуто сколь угодно большое число раз. Но для изменения его численного значения понадобится внести изменения только в одну строку с директивой #define.

Во вторых, директива #define используется для определения макросов. Макрос - это набор выражений языка Си, которому поставлено в соответствие определенное имя. При записи этого имени в программе, компилятор произведет замену этого имени обозначенным набором выражений. Например, Вам необходимо разрешить прерывания в МК. Для этого в МК 68HC12 используется команда ассемблера CLI. Для ее записи в тексте программы на Си определяют макрос:

```
#define CLI ( ) asm(«cli\n»); //разрешить маскируемые прерывания
```

Далее в программе используют только имя макроса:

```
CLI();
```

Кроме директивы определения символа или макроса #define, существует директива обратного действия #undef. Приведем пример ее использования:

```
#define VALUE10
int number[VALUE];
#undef VALUE
```

В этом примере мы сначала назначили переменной VALUE значение 10. Далее в строке 2 мы воспользовались этим значением, чтобы определить массив целых чисел из 10 элементов. Далее переменная VALUE нам не нужна. И мы отменили ее определение директивой #undef.

Следующая рассматриваемая нами директива – это директива #include. Ранее мы установили, что эта директива используется для присоединения к разрабатываемому программному модулю другого файла. При этом у программиста появляется возможность использовать в тексте программы ранее объявленные переменные или вызывать функции, которые были определены в другом файле. Присоединяемые файлы называют заголовочными файлами. Например, следующая запись необходима для присоединения к разрабатываемой программе файла стандартных функций ввода/вывода:

```
#include     <stdio.h>
```

Символы < > указывают на определенное место расположение файла stdio.h в папках директории компилятора.

Назначение директивы #error – упрощение процесса отладки разрабатываемой программы. Вы можете записать следующее выражение:

```
#error Programm made a logic error
```

Если в процессе выполнения программа достигнет приведенной строки, то на экран будет выведено приведенное сообщение.

Также для целей отладки используется директива #line. Эта директива отмечает номерами те инструкции программы, которые следуют за директивой. В результате, в процессе отладки можно идентифицировать тот фрагмент программы, который исполняется в текущий момент отладки.

Функции директивы #pragma определяются конкретным типом используемого компилятора. Для компилятора ICC12 эта директива определения сегментов данных и программы в исходном тексте программы на Си, для объявления подпрограмм прерывания, а также для присвоения желаемых значений ячейкам памяти с фиксированными адресами. Последнее позволяет инициализировать таблицу векторов прерываний в микроконтроллерах.

Приведенный ниже пример демонстрирует использование директивы #pragma для объявления подпрограммы с именем TOISR в качестве подпрограммы прерывания:

```
#pragma interrupt_handler TOISR()
void TOISR(void);
```

Объявление подпрограммы TOISR как подпрограммы прерывания информирует компилятор о том, что в конце этой подпрограммы он должен расположить ассемблерную инструкцию возврата из прерывания rti. В конце обычной функции компилятор подставляет инструкцию возврата из подпрограммы rts.

Директива #pragma также используется для задания начального адреса расположения в памяти сегментов программного кода или кодов данных. Запишем вектор прерывания для подпрограммы TOISR в таблицу векторов прерывания МК. Мы знаем, что в соответствие с картой памяти МК, вектор прерывания по переполнению таймера должен располагаться по адресу 0x0B1E. Следующая запись помещает адрес начала подпрограммы TOISR в две ячейки памяти, начиная с адреса 0x0B1E:

```
#pragma abs_adress:0xB1E
void (*Timer_Overflow_interrupt_vector[]) () = {TOISR};
#pragma end _abs_adress
```

Более подробно оформление подпрограмм прерывания мы обсудим в главе 4.

3.7. Конструкции программирования

В теории программирования доказано, что любая программа может быть закодирована с помощью комбинаций трёх конструкций: последовательность операторов, выбор и итерация. В этом разделе мы рассмотрим набор операторов языка Си и типовые примеры их использования для реализации конструкций цикла (итерации) и принятия решения (выбора).

3.8. Операторы для организации программных циклов

В языке Си существует несколько операторов, которые позволяют реализовать циклические вычисления (итерации). В этом параграфе мы рассмотрим программные конструкции циклов с операторами for, while, do-while.

3.8.1. Оператор FOR

Оператор for предназначен для реализации циклов со счетчиком. В операторе for могут автоматически реализоваться сразу три операции: инициализация счётчика цикла, проверка его значения и модификация. Синтаксис оператора for:

```
for  (<выражение1>; <выражение2>; <выражение3>)
<операторы тела цикла>
```

Рассмотрим типичный пример реализации цикла с оператором for:

```
1      for ( i = 0 ; i < 10 ; i = i + +)
2      {
3      inst 1;
4      inst 2;
5      :
6      :
7      inst n;
8      }
```

В строке 1 записывается сам оператор for, за которым обязательно следуют три выражения, заключенные в круглые скобки. Выражение 1 вычисляется один и только один раз перед проверкой условия цикла. В нашем примере выражение 1 присваивает начальное значение переменной i. Выражения два и три могут иметь произвольный характер, но обычно их используют для проверки и модификации условия продолжения цикла. Выражение 2 задаёт условие продолжения цикла. Если его значение отлично от нуля ("истина"), будут выполнены операторы 3...7, составляющие тело цикла. После этого вычисляется выражение 3, указанное в скобках первой строки. В нашем примере выражение i = i + += i + 1 осуществляет увеличение на 1 внутреннего счетчика циклов оператора for. Поэтому операторы тела цикла 3...7 будут выполняться 10 раз при значениях переменной цикла i от 0 до 9. В конце первого цикла значение i будет равно 1, в конце второго – i=2, и т.д. В конце десятого цикла переменная i примет значение 10. Далее начнется исполнение 11-ого цикла оператора for, но, проанализировав условие выражения 2 оператора for, программа выйдет из цикла, не исполняя операторов тела цикла.

Достаточно часто переменная счетчика циклов оператора for используется также в теле цикла этого оператора. Например, следующий программный фрагмент вычисляет таблицу соответствия значений температуры, записанных по

шкале Цельсия и по шкале Фарингейта, и последовательно выводит эти значения на экран монитора. Диапазон исходных значений температуры составляет от –10 С до +40 С.

```
1       for ( k = -10 ; k <= 40 ; k + +)
2       {
        //преобразовать значения температуры, измеренные по шкале Цельсия
        // к численному значению по шкале Фарингейта
3       Temperature = k*9/5+32;
4       printf(«Current temperature is \%f\n», Temperature);
5       }
```

Как видите, значение переменной k используется как параметр для вычисления новых значений температуры в теле цикла.

3.8.2. Оператор WHILE

Второй способ организации циклов использует оператор while. Применение оператора while иллюстрирует следующий программный фрагмент:

```
1       k= -10
2       while ( k < 40 )
3       {
4       Temperature = k*9/5+32;
5       k + +
6       printf(«Current temperature is \%f\n», Temperature);
7       }
```

В отличие от оператора for, переменная k, используемая в качестве счетчика циклов, должна быть инициализирована перед оператором while, например, в строке 1. Обратите внимание, что в строке 2 в скобках оператора while записано всего лишь одно выражение, которое называется условием цикла. Выполнение оператора while начинается с вычисления этого выражения. Если значение выражения отлично от нуля («истина»), то выполняются операторы 4..6 тела цикла. После выполнения операторов тела цикла снова вычисляется выражение условия и процесс повторяется. Таким образом, выполнение тела цикла происходит пока значение выражения условия цикла отлично от нуля («истина»).

Следует заметить, что если условие цикла не выполнится на первой итерации, то тело цикла не будет выполнено ни разу. Для того чтобы тело цикла выполнялось хотя бы один раз, в языке Си предусмотрен оператор do-while, который рассматривается далее.

Одним из результатов выполнения тела цикла, как правило, является изменение условия цикла, иначе цикл будет бесконечным. Но, естественно, возможны случаи, когда условие цикла зависит от результата работы вызываемой из выражения функции, или условие цикла меняется в функции обработки прерывания, которая активизируется во время выполнения тела цикла.

Тело цикла может вообще отсутствовать, в случае применения на его месте пустого оператора. Это бывает нужно, например, при программном ожидании установки какого-либо аппаратного флага микроконтроллера, который изменяется встроенной периферией. С помощью оператора while можно создавать бесконечные циклы.

```
1      while (1)
2      {
3      Instructions //выполнение блока операторов
4      }
```

Так в приведенном примере блок операторов с именем Instructions будет исполняться микроконтроллером до тех пор, пока МК не перейдет в состояние сброса или не произойдет прерывание. В первом случае МК начнет исполнение программы сначала, во втором – МК перейдет к исполнению программы обслуживания прерывания.

3.8.3. Оператор DO- WHILE

Третий способ организации циклов в Си использует оператор do-while. Синтаксис оператора do-while:

```
do
{
<операторы тела цикла>
}
while ( <выражение 1> );
```

Пример записи вычисления таблицы соответствия температур с использованием оператора do-while приведен ниже:

```
1      k= -10
2      do
3      {
4      Temperature = k*9/5+32;
5      k + +
6      printf(«Current temperature is \%f\n», Temperature);
7      }
8      while ( k < 40 )
```

Оператор do продолжает циклическое исполнение операторов тела цикла 4...6 до тех пор, пока значение выражения 1 не станет равным нулю («ложным»). Оператор do-while похож на оператор while, но условие цикла в нём вычисляется и проверяется после очередного исполнения операторов тела цикла. Таким образом, операторы тела цикла выполняются, по крайней мере, один раз, даже если условие цикла заведомо ложно.

3.9. Операторы принятия решения

В этом параграфе мы обсудим примеры применения операторов if-then-else. Анализируя в главе 2 различные блок-схемы алгоритмов управления, Вы часто наблюдали ситуацию, при которой некоторые действия должны были быть произведены только в том случае, если выполняется определенное условие. Мы рассмотрим четыре способа записи программного кода на Си, реализующего выполнение отдельных фрагментов программы при соблюдении заданного условия.

3.9.1. Оператор IF

Оператор if – это оператор выбора. Синтаксис оператора if:

```
if ( <выражение> )
{
<оператор 1>
}
else
{
<оператор 2>
}
```

Работа оператора if заключается в следующем. Сначала вычисляется заключенное в скобки выражение. Если его значение оказалось отличным от нуля («истина»), то выполняется <оператор 1>. Если используется служебное слово else и значение выражения равно нулю («ложь»), то выполняется <оператор 2>, указанный после else. Если значение выражения равно нулю («ложь»), а служебное слово else не указано, управление передаётся следующему за if оператору программы.

```
1      if (input = = 0x00)
2      {
3      output = 0x0F;
4      }
5      ……..
```

В этом примере переменная input проверяется на равенство 0. Если текущее значение этой переменной действительно равно 0, то выполняется оператор, записанный в строке 3. Служебное слово else в данном примере отсутствует, поэтому, если текущее значение переменной input не равно 0, то программа осуществляет переход к строке 5. В качестве первого и второго операторов в условном операторе if можно применять блоки операторов.

Приведенный выше пример может быть записан в сокращенной форме, без выделения фигурными скобками блока операторов условно выполняемых операторов:

```
1      if (input = = 0x00)
2      output = 0x0F;
```

Применяя конструкции с оператором if, полезно знать, что программа принимает решение о выполнении того или иного блока операторов по значению внутренней логической переменной, которая устанавливается в 0 или в 1 по результату анализа заключенного в скобки условного выражения. Поэтому в качестве выражения условия программист может записать некоторую логическую формулу. В ходе исполнения конструкции if ее значение будет вычислено, по результату вычисления (0 или 1) будет принято решение о выполнении той или иной группы операторов. Частным случаем логического выражения условия является запись:

```
1      if (1)
2      output = 0x0F;
```

Понятно, что действие output = 0x0F будет выполняться всегда, что иногда полезно на этапе отладки программы.

3.9.2. Оператор IF-ELSE

Многие алгоритмы управления требуют применения оператора if-else, который позволяет выполнить ту или иную группу операторов, в зависимости от результата анализа условия, следующего в скобках за оператором if. Допустим, встраиваемая система должна осуществлять управление кондиционером, анализируя текущее значение температуры воздуха. Эта задача может быть решена посредством записи следующего фрагмента программы:

```
1      if (input > 78)            // если температура по шкале Фарингейта
                                  //больше 78
2      air_condision = on;        // то включить кондиционер
3      else
4      air_condision = off;       //иначе выключить кондиционер
```

В этом примере переменная input содержит в себе код температуры окружающей среды, который программа должна подвергнуть сравнению с пороговым значением 78. Учитывая, что в процессе вычисления условия оператора if, программа возвращает значение внутренней логической переменной, то же действие можно записать, используя обратную логику:

```
1      if (input <= 78)           //если температура меньше или равна 78
2      air_condision = off;       //то выключить кондиционер
3      else
4      air_condision = on;        //иначе включить кондиционер
```

3.9.3. Оператор IF-ELSE IF-ELSE

Операторы if могут быть вложенными. В этом случае служебное слово else (если оно используется) связывается с последним оператором if, с которым ещё не было связано else. Во избежание путаницы с вложенными операторами, лучше пользоваться фигурными скобками или структурировать текст отступами для явного указания того, к какому из операторов if принадлежит слово else. Примеры применения вложенных операторов if – else приведены ниже:

```
1      if (input > 78)
2             air_condision = on; //включить кондиционер, если жарко
3      else
4             if (input > 58)
5                    fan = on;    //включить вентилятор,если душно, но
                                  //не жарко
6             else
7             heater = on; //включить обогреватель, если прохладно
8      ……
```

В этом примере программа должна различить, к какому из трех диапазонов принадлежит текущее значение температуры окружающей среды. Если температура превышает 78 градусов по Фарингейту, то должен быть включен кондиционер. Если

температура меньше 79 градусов, но больше 58, то следует оставить включенным только вентилятор, а если температура равна или ниже 58 градусов, то следует включить обогреватель.

Те же управляющие действия могут быть записаны с применением другой конструкции операторов if - else:

```
1      if (input > 78)
2      air_condision = on; //включить кондиционер, если жарко
3      if (input > 58)
5      fan = on;           //включить вентилятор,если душно и жарко
6      else
7      heater = on;        //включить обогреватель, если прохладно
```

Отличие этого варианта программы от предыдущего состоит в том, что оба оператора if – else стали независимыми друг от друга. А в предыдущем примере второй оператор if – else был вложен в первый. Это конструктивное изменение программы внесло коррективы в реализуемый ею алгоритм управления. Так в первом варианте при значении переменной input=80 выражение первого оператора if окажется истинным и будет включен кондиционер. И далее управление будет передано строке 8, т.е.вторая конструкция if будет пропущена из рассмотрения. Во втором случае при том же значении переменной input=80 после проверки первого условия будет также включен кондиционер, но затем управление будет передано второму оператору if. Поскольку второе условие также выполняется, то вентилятор будет также включен.

Еще один вариант реализации того же задачи:

```
1      if (input > 78)
2      air_condision = on; //включить кондиционер, если жарко
3      if ((input > 58)&&(input < 79))
5      fan = on;           //включить вентилятор,если душно, но не жарко
6      if (input < 59)
7      heater = on;        //включить обогреватель, если прохладно
```

Проанализировав этот программный фрагмент, Вы увидите, что он соответствует первому варианту управления кондиционером, вентилятором и обогревателем. Логика построения этого программного фрагмента хорошо структурирована и позволяет реализовать множество разных действий при различных значениях переменной input. Причем ни одно действие по логике построения программы не будет сочетаться с каким-либо другим из перечисленного списка:

```
1      if (condition 1)
2             (instruction set 1);
3      if (condition 2)
5             (instruction set 2);
6      if (condition 3)
7             (instruction set 3);
```

Если число различных значений переменной условия condition превышает 4 или 5, то для выполнения подобной задачи следует использовать оператор swith.

3.9.4. Оператор SWITCH

Данный оператор применяется, когда требуется передавать управление одному из нескольких операторов, в зависимости от значения выражения. Синтаксис оператора swith:

```
1       switch  ( <выражение> )
2       {
3           case <константное_выражение_1> :
4              <оператор_1>;
5              break;
6           case <константное_выражение_2> :
7              <оператор_2>;
8              break;
9           :
10          :
11          case <константное_выражение_n> :
12             <оператор_n>;
13             break;
14          default :
15             <оператор_n+1>;
16      }
```

Выполнение оператора switch начинается с вычисления <выражения> в скобках. Результат вычисления должен быть целочисленным в одно- или двухбайтовом формате. Затем последовательно просматриваются префиксы case, указанные после них константные выражения вычисляются и их результаты сравниваются с результатом вычисления выражения в скобках после слова switch. Если результаты совпадают, то управление передаётся оператору, следующему за соответствующим префиксом case. Если ни одного совпадения не произошло и при этом указано служебное слово default (его указание необязательно), то управление передаётся оператору, следующему за этим служебным словом. Если совпадения не обнаружилось, а служебное слово default не указано, то ни один из операторов блока не выполняется. Заметим, что в константных выражениях после префиксов case не допускается применения переменных, выражение должно состоять из константных величин.

Приведем пример использования конструкции с оператором switch:

```
1       swith (a){
2           case 1:
3             printf(«Correct value%d was chosen\n»,a);
4             break
5           case 2:
6             printf(«Close but try again\n»);
7             break;
8           case 3:
9             printf(«Value %d is two away from the answer\n»,a);
10            break;
11          default:
12            printf(«Your chosen value is way off\n»);
13      }
```

В примере текущее значение переменной a сравнивается со значениями 1, 2 и 3. Желаемое сообщение о правильном выборе появится на экране монитора в том случае, если переменная равна 1. Если же переменная равна 2 или 3, то будут выведены соответствующие сообщения об ошибках. Если ни одно из трех возможных значений не выбрано, то отобразится сообщение, записанное в строке 12. Обратите внимание, в примере мы не должны записывать слово break в строке 13, т.к. после выполнения оператора строки 12 программа автоматически продолжит исполнение следующих за конструкцией switch операторов.

А как будет вести себя программа, если мы пропустим слово break где-то в середине конструкции switch, например, в строке 4? После отображения сообщения строки 3 программа перейдет к исполнению следующих операторов, пока не встретит следующее слово break или не дойдет до конца конструкции switch. В нашем случае программа отобразит сообщение строки 6, а затем предаст управление строке 13. Зная подобную особенность оператора switch, Вы можете пропустить несколько break в своей программе.

3.10. Массивы

Массив определяет непрерывный набор однотипных объектов данных. Признаком массива служит использование квадратных скобок после идентификатора переменной. При определении в квадратных скобках указывается количество элементов массива (его размер), а при использовании в выражениях – индекс требуемого элемента. Массивы могут содержать элементы в любом из ранее рассмотренных типов представления данных: char, int, float double. Пример определения массива из 10 двухбайтовых целочисленных элементов:

```
int list[10];
```

Данное определение зарезервирует в памяти МК 20 однобайтовых ячеек памяти. В выражениях программы доступ к каждому элементу массива возможен с использованием индекса. Индекс в Си автоматически отсчитывается с 0. Поэтому в нашем примере индекс может принимать значения от 0 до 9 включительно.

Как и любая другая переменная, массив может быть объявлен с одновременной инициализацией его значений. Для этого в правой части операции присваивания «=» в фигурных скобках, через запятую перечисляются значения всех элементов массива:

```
int list[10]= {1,2,3,4,5,6,7,8,9,10};
```

Если при определении массива его размер не указывается, такой массив обязательно необходимо явно проинициализировать. При этом компилятор автоматически приравнивает размер массива к числу проинициализированных элементов.

```
int list[]= {1,2,3,4,5,6,7,8,9,10}; // Размер массива будет равен
//числу проинициализированных переменных т.е. 10
```

Также можно инициализировать произвольное число первых элементов массива с указанным размером:

```
int list[10]= {1,2,3,4,5,}; // Инициализация первых пяти элементов
                            //массива, состоящего из 10 элементов
```

Для массивов символов можно применять сокращённую запись, при которой в правой части операции присваивания в кавычках записывается требуемая строка символов. При этом если не указан размер массива, компилятор принимает его равным количеству символов строки плюс один, поскольку в Си строка символов заканчивается ASCII символом с кодом ноль, который автоматически добавляет компилятор.

```
char message[6]= "Smile";

char message[]= "Smile";
```

Первый элемент массива message[0] содержит код символа «S», последний элемент message[5] – код конца строки 0x00.

Массив может быть многомерным. Количество измерений определяется количеством индексов массива. Пример определения двухмерного массива:

```
int matrix[2],[3];
```

По аналогии с одномерными массивами, для многомерных массивов возможна инициализация в процессе его определения:

```
int matrix[2],[3]= {1,2,3}{4,5,6};
```

Многомерный массив будет располагаться в памяти в следующем порядке:

```
m[0][0]  m[0][1]  m[0][2]  m[1][0]  m[1][1]  m[1][2]
```

Обращение к элементам многомерного массива осуществляется посредством указания сразу нескольких индексов. Так после выполнения следующего выражения переменная m примет значение 6:

```
m = matrix[1] [2]
```

Объединение некоторой группы данных в массив позволяет оперировать с этой группой, используя всего лишь одно имя переменной. Программы становятся более логичными и читабельными, что уменьшает вероятность появления ошибок программирования. Допустим по ходу исполнения алгоритма необходимо каждый элемент массива odd из 100 элементов, увеличить на 1. Следующий программный фрагмент демонстрирует реализацию этой задачи:

```
For (i=0, i<100, i++)
odd[i] = odd[i] +1;
```

Внимательный читатель должен был бы заметить, что в примерах этого параграфа мы использовали только массивы целочисленных элементов. Далее в примерах программирования МК семейства 68HC12 будут использоваться именно целочисленные исчисления, поскольку вычисления в формате с плавающей запятой предполагают достаточно сложный алгоритм управления, который вряд ли уместен при начальном обучении. Однако мы рассмотрим более сложный вариант массива – массив структур в параграфе 3.12.

3.11. Указатели

В языке Си заложена возможность определения отдельного класса переменных, которые называются указателями. Также предусмотрены операции, которые могут быть использованы для доступа к этим указателям и для манипулирования ими.

Указателем называется переменная, предназначенная для хранения адреса другой переменной. Например, указатель однобайтовой переменной содержит в себе адрес переменной типа char. Указатели могут применяться для упрощения манипулирования массивами, структурами и другими блоками данных, а также для доступа к конкретным физическим ячейкам памяти микроконтроллерных систем. Именно это свойство языка Си делает его столь популярным среди сообщества разработчиков встраиваемых систем.

Для определения переменной-указателя используется оператор * :

```
int *ptr
```

Приведенная запись означает, что переменная ptr содержит в себе адрес старшего байта двухбайтовой переменной типа int. Переменная-указатель может быть определена для адресации различных типов переменных: целочисленных однобайтовых типа char и двухбайтовых типа int, одномерных и многомерных массивов из переменных char или int, одиночных переменных или массивов из элементов в формате с плавающей запятой, а также для структур, определенных пользователем. Спецификация типа указателя информирует компилятор о размере и типе объекта, на который показывает указатель. Таким образом, при обращении к любой области памяти через указатель, содержимое этой области будет трактоваться в соответствие с заданным типом указателя. В микроконтроллерах, поддерживающих различные способы формирования адресов объектов, спецификация типа может влиять на размер области памяти, отводимой для хранения указателя.

Применение указателей в реальных программах требует особого внимания программистов, поскольку именно с указателями связано основное количество ошибок в исходном тексте программы на Си. При манипулировании указателями, а также при выполнении операций присваивания следует убедиться в корректном использовании различных типов переменных. Приведем пример применения указателя:

```
1       void main(void)
2       {
3       char *ptr;
```

```
4       static char message[] = «What a wonderful day!»;
5       ptr = message;
6       printf(«%s\n», ptr);
7       }
```

В этом примере переменная ptr была определена как указатель для однобайтовой переменной типа char. Поэтому переменная ptr должна содержать 16-разрядный адрес однобайтовой переменной. В строке 4 примера был определен и инициализирован массив однобайтовых переменных message. Этот массив содержит 22 элемента: 21 символ в кодах ASCII и один байт признака конца строки. Выражение в строке 5 имеет своей целью запись в переменную ptr начального адреса массива message, т.е. адреса символа «W». Выражение, записанное в строке 6, должно начать вывод на экран монитора символа, на который указывает содержимое переменной-указателя ptr. В строке 5 данного примера может быть записано другое, более понятное выражение:

```
ptr = &message[0];
```

В этом выражении символ «&» выполняет операцию взятия адреса первого элемента массива message. И именно эта функция была реализована нами в строке 5 исходного примера.

Выполняемое в рассмотренном выше программном фрагменте действие, может быть оформлено на Си другим образом, без использования указателя:

```
void main(void)
       {
static char message[] = «What a wonderful day!»;
int i;
for (i=0, i<21, i++)
putchar(message[i]);
}
```

Обратите внимание, что в данном примере внутренняя переменная цикла i принимает значения от 0 до 20 включительно, выводя при этом на экран 21 символ. Приведенный способ реализации задачи вывода на экран строки символов обладает одним существенным недостатком. При смене числа символов в записи, выводимой на экран, придется написать новый фрагмент кода. Указанный недостаток исправлен в следующем программном фрагменте:

```
void main(void)
       {
       char *ptr;
       static char message[] = «What a wonderful day!»;
       ptr = message;
       while (*ptr != '\0')
              {
              putchar(*ptr);
              ptr++;
              }
       }
```

В этом примере указатель ptr используется для передачи в функцию putchar одного кода символа, который будет выведен на экран. Далее содержимое ptr увеличивается на 1, адресуя тем самым следующий элемент массива. По условию цикла while вывод на экран закончится, если в последовательно перебираемом массиве будет найден код конца строки.

Во встраиваемых системах указатели используются для обращения к регистрам специальных функций микроконтроллера. Рассмотрим следующую запись:

```
#define PORTA *(volatile unsigned char *) 0x1000
```

Эта запись информирует компилятор о том, что однобайтовая целочисленная переменная с именем PORTA располагается в памяти по адресу 0x1000. Служебное слово volatile информирует программу (и отладчик) о том, что значение PORTA может изменяться не только под управлением программы. Любое упоминание имени PORTA далее по тексту программы будет связано с выполнением операций чтения или записи в регистр данных PORTA по его физическому адресу 0x1000. Поэтому следующий программный фрагмент позволит прочитать содержимое порта PORTA в переменную new_value:

```
DDRA = 0x00;         //инициализировать порт PORTA на ввод
new_value = PORTA;   //читать содержимое регистра данных PORTA в
                     //переменную new_value
```

3.12. Структуры

Возможности языка Си позволяют объединить под одним именем переменные с разным форматом представления данных. Для этого используется понятие структуры.

Структура – это объект, состоящий из данных различных типов. При использовании структур следует различать объявление (или описание) структуры, как нового типа данных, например тип данных «структура x», от фактического определения некоторой переменной с типом данных «структура x». Пример объявления структуры типа car:

```
Struct car {
       int doors;
       char color[10];
       char *maker;
       int num_cyl;
       int year;
       }
```

Служебное слово struct указывает начало объявления или определения структуры. За ним следует имя, которое присваивается данной структуре, оно называется идентификатором структуры. В фигурных скобках указывается список определений отдельных элементов, образующих структуру. Элементы структуры могут иметь любой тип, включая массивы, другие структуры и объединения.

Анализируя приведенную запись, отметим, что имя car – это имя нового типа данных. Не следует путать его с именем переменных типа car, которые далее будут определены в программе:

```
Struct car your_car;
Struct car my_car;
Struct car his_car;
```

Записав приведенные выше строки, мы определили в программе три переменные с именами your_car, my_car, his_car, и каждая из этих переменных представляет собой структуру с одним и тем же набором элементов. Так первым элементом каждой из этих переменных-структур будет двухбайтовое целое число. Численные значения этих первых элементов для всех переменных-структур различаются, но формат представления данных для всех трех первых элементов одинаков.

Каждый элемент структуры имеет свое собственное имя: doors, color, maker и т.д. Допускается обращение к каждому элементу структуры с использованием его имени:

```
your_car.doors = 4;
```

Если переменная типа структура передается в качестве параметра в какую-либо функцию, то следует помнить, что передаются не сами элементы структуры, а лишь указатель на первый элемент названной структуры. Поэтому при обращении внутри функции к отдельным элементам структуры следует вместо оператора «.» использовать оператор «–>». Например, в программе определяется функция assign_doornumbur, которая присваивает численное значение первому элементу структуры типа car:

```
void assign_doornumbu (struct car *some_car);
:
:
;
assign_doornumber(&your_car)
:
:
void assign_doornumbu (struct car *some_car)
{
some_car–>doors = 4;
return 0;
}
```

А сейчас мы обсудим, как создать массив структур. Допустим, что мы уже определили структуру circuit_board как новый тип данных:

```
Struct circuit_board {
       int trunsistor;
       int bus;
       int serial_port;
       int parallel_port;
       }
```

Создадим теперь массив new_board с пятью элементами, каждый из которых является структурой типа circuit_boar :

```
Struct circuit_board newboard[5];
```

К каждой записи внутри каждого элемента этого массива можно обратиться, используя номер элемента (фактически это номер структуры внутри массива) и имя записи внутри структуры:

```
newboard[2].transistor = 100;
```

Это выражение присваивает значение 100 записи под именем transistor третьего элемента массива newboard (элементы в составе массива отсчитываются с нулевого).

Этим разделом мы закончили краткий обзор техники программирования МК на Си. Перейдем теперь к рассмотрению процесса генерации файла исполняемого кода программы для выбранного типа МК из файла исходного текста программы на Си.

3.13. Процесс программирования и отладки микропроцессорной системы

В данном разделе мы обсудим технологию создания и отладки прикладной программы в микроконтроллерных системах. Сначала рассмотрим процесс создания исходного текста управляющей программы и его превращение в файл исполняемого кода (executable file).

3.13.1. Технология создания программного кода

Процесс программирования кажется достаточно простой монотонной деятельностью до тех пор, пока задача управления, которую требуется реализовать, достаточно проста. Однако программисты с опытом знают, что большинство технических заданий удивительно быстро превращается достаточно сложный проект, часть задач которого разработчиком ранее не реализовывалась. Именно по этой причине необходимо правильно организовать исполнение проекта, даже если первоначальная задача кажется простой. Поэтому авторы настоятельно советуют читателю приобрести навыки системного подхода к написанию программ, независимо от языка программирования, на котором реализуется проект. Поскольку данная книга не претендует на системное изложение вопросов теории программирования, то наши советы по данной теме будут достаточно краткими.

При написании программы первым делом необходимо как можно более полно понять принцип действия и особенности работы того устройства, для которого программа предназначена. На основе этих знаний следует создать такую структуру программы, которая позволит удовлетворить как текущему техническому заданию, так и возможным будущим его дополнениям. На первых порах Вы можете начать самостоятельную работу, следуя рекомендациям главы 2. Вы уже умеете перейти от про-

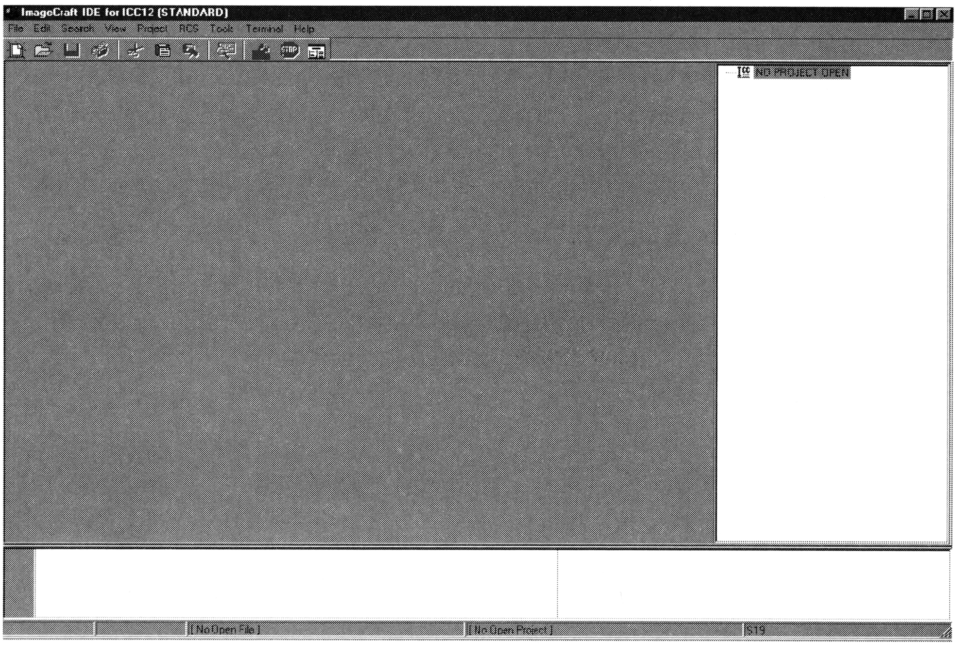


Рис. 3.1. *Интерфейс пользователя интегрированной среды разработки ICC12*

извольного описания алгоритма управления к структуре функций управления и блок-схемам алгоритмов. Рассмотренные в главе 2 методы структурного проектирования позволят Вам сначала увидеть задачу управления целиком, а затем произвести ее разбиение на отдельные блоки (подзадачи). Каждая такая подзадача должна быть оформлена в виде законченной подпрограммы или функции. Выделенные функции впоследствии будут объединены в один программный модуль, который должен быть отлажен и тестирован до того, как все другие составляющие модули программы будут объединены. Процессы написания, тестирования и отладки программ близко связаны. Вы никогда не должны сначала собрать полный текст программы, а затем начать ее отлаживать. Правильным подходом является написание, тестирование и отладка каждого программного модуля по отдельности перед тем, как эти модули будут объединены вместе. Процесс объединения должен использовать тот же подход: добавлять уже оттестированные модули к отлаженной части общей программы следует последовательно. И после добавления каждого нового модуля не лениться тестировать функционирование получившегося нового промежуточного программного продукта, в том числе убедиться в исполнении новых добавленных функциональных возможностей.

Напомнив Вам основные моменты структурного подхода к процессу написания исходного кода и его отладке, рассмотрим процесс превращения файла исходного кода на Си в файл исполняемого кода для определенного типа микроконтроллера. Мы рассмотрим эту технологию с использованием интегрированной среды разработки ICC12 версии 6.12B от компании ImageCraft. Если Вы используете программную среду разработки от другого производителя, то этот материал не будет для Вас

бесполезным, поскольку принципы преобразования кодов в процессе создания исполняемого модуля прикладной программы одинаковы для всех аналогичных программных продуктов.

На рис. 3.1 представлен интерфейс пользователя (картинка на экране монитора), возникающая при запуске среды ICC12. Большое пустое окно в центре экрана предназначено для ввода и редактирования исходного текста программы на Си. Окно меньшего размера в правой части экрана – окно менеджера проектов. Оно предназначено для отображения списка всех используемых в текущем проекте файлов. Окно состояния в нижней части экрана предназначено для отображения текущей информации о режимах работы и состоянии обрабатываемых в среде файлов. В этом окне будут выводиться сообщения об ошибках, возникающих при работе программ компилятора, Ассемблера, линковщика и загрузчика/программатора в процессе обработки файлов текущего проекта. Теоретически, для ввода и редактирования файлов исходного текста программ могут быть использованы любые текстовые редакторы, однако программные продукты класса «интегрированная среда разработки» обязательно содержат собственный редактор текста, которым и следует воспользоваться. Пакет ICC12, как и любая другая интегрированная среда разработки, предоставляет программисту удобный интерфейс пользователя для работы с встроенными в среду программами компилятора, Ассемблера, линковщика, загрузчика и программатора. Каждая из программ может быть запущена на исполнение, как из контекстного меню, так и с помощью кнопок на панели управления. На рис. 3.2 показан интерфейс пользователя среды ICC12 с текстом программы в окне редактирования и сообщениями о результатах ее компиляции в окне состояния.

После того, как исходный текст программы написан и находится в окне редактирования, файл программы должен быть обработан препроцессором компилятора Си. Препроцессор – это часть программы компилятора, которая анализирует выражения в исходном тексте, которые начинаются с символа «#». Вспомните, с этого символа начинаются директивы подключения заголовочных файлов, директивы условной компиляции и директивы для объявления подпрограмм прерывания. Если препроцессор не зафиксировал синтаксических ошибок, то вступает в работу синтаксический анализатор и генератор ассемблерного текста компилятора Си. В результате, после обработки компилятором исходного текста программы на Си, будет получен текст исходной программы на языке Ассемблера для данного типа МК. В нашем случае это МК семейства 68HC12. Сгенерированный компилятором текст будет содержать как мнемоники команд ассемблера МК данного семейства, так и псевдокоманды и директивы для программы Ассемблер в составе пакета ICC12. Смысловые названия и шаблоны имен входных и выходных файлов компилятора Си для среды ICC12 приведены на рис. 3.3. Программы компиляторов, которые способны генерировать инструкции языка ассемблера для процессорного ядра, отличающегося (т.е. программно несовместимо) от ядра, на котором программа компилятора исполняется, называют кросс-компиляторами. Так в нашем случае программа кросс-компилятора, исполняемая на персональном компьютере, генерирует ассемблерный текст для МК 68HC12.

Исходный текст программы не обязательно должен быть написан на Си. Однако его написание на языке ассемблера для больших задач приводит к большому числу ошибок, что требует длительного процесса отладки. В проектах, которые связаны с программным обслуживанием быстропротекающих процессов, целесообразно наиболее критичные по времени исполнения фрагменты программы писать на ассемблере, в то время, как основную часть программы – на Си. Компиляторы Си допускают объединение этих двух языков в файле исходного текста программы.

Полученный после обработки программой компилятора файл прикладной программы далее обрабатывается программой Ассемблер. Эта программа преобразует файл программы в файл объектного кода с расширением «o» (см. рис. 3.3).

В соответствие с методом системного проектирования, разрабатываемая прикладная программа должна состоять из нескольких файлов, каждый из которых содержит отдельный модуль программы. Часть этих модулей может быть разработана и отлажена в составе другого проекта, и поэтому не требует повторной компиляции. В этом случае для создания конечного варианта разрабатываемой программы необходимо объединить в один файл ранее полученные объектные файлы модулей и вновь разработанные, прошедшие обработку компилятором и Ассемблером файлы. Для объединения нескольких объектных модулей в один файл исполняемого кода используется программа линковщика. Линковщик генерирует три типа файлов с расширениями «s19», «map» и «lst». Файл карты памяти «xxx.map» содержит в себе информацию о расположении кодов прикладной программы в адресном пространстве МК. Файл листинга «xxx.lst» отражает процесс перевода мнемоник команд ассемблера в машинные коды. Файл в формате «s19» именуют файлом исполняемого кода или загрузочным модулем, поскольку именно этот файл заносится в постоянную память МК и исполняется им в процессе управления проектируемым устройством. Таким образом, в результате работы специальных программ в составе пакета интегрированной среды разработки, один или несколько файлов на Си были обработаны программами компилятора, Ассемблера и линковщика с целью получения одного исполняемого на выбранном типе МК файла машинных кодов прикладной программы.

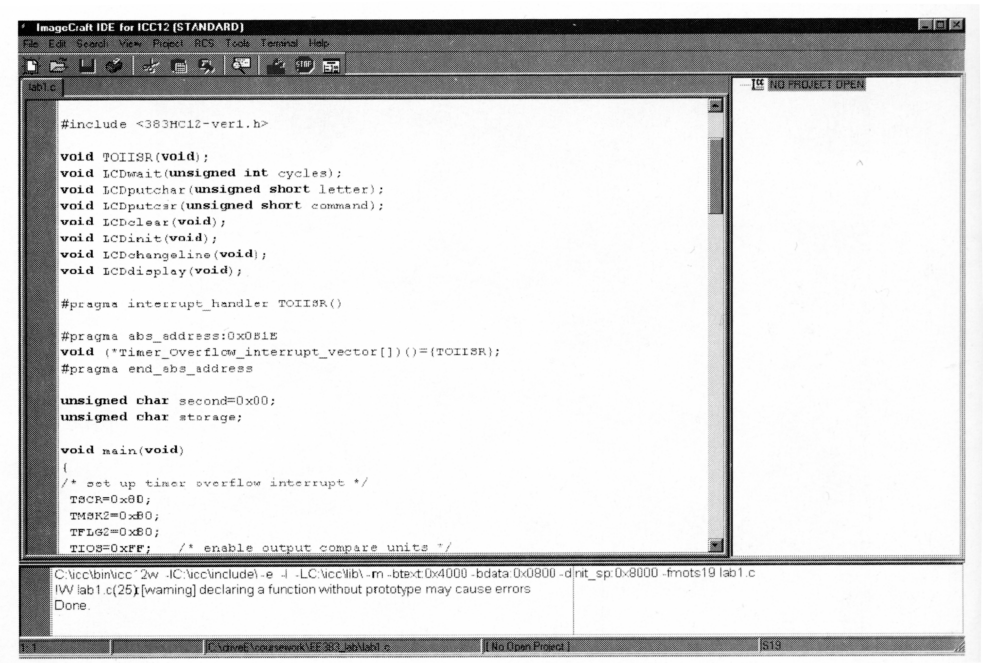


Рис. 3.2. *Интерфейс пользователя интегрированной среды разработки ICC12*
В окне редактирования загружен файл исходного текста программы на Си.
В окне состояния сообщение об успешной компиляции этого файла.

Процесс разбиения задачи для программирования на множество программных модулей, каждый из которых функционально завершен и предназначен для реализации отдельной функции управления, называется структурным программированием. Для объединения отдельных программных модулей в одну программу управления с возможностью независимой модификации и отладки каждого модуля недостаточно располагать только исходными текстами или объектными файлами этих модулей. Необходим также достаточно большой набор служебных файлов, которые вместе с файлами модулей составляют так называемый проект задачи. Поэтому программа в составе интегрированной среды разработки, которая осуществляет управление процессом компиляции, ассемблирования, объединения модулей программой линковщика и загружает полученный исполняемый код в память микроконтроллера на отладочной плате, называется менеджером проектов (Project Manager или Project Builder). Это еще одна программа в составе пакета интегрированной среды разработки типа ICC12.

Программа менеджера проектов считается основной в составе IDE, поскольку она управляет доступом пользователя и взаимодействием с обрабатываемыми файлами всех остальных программ пакета интегрированной среды разработки и

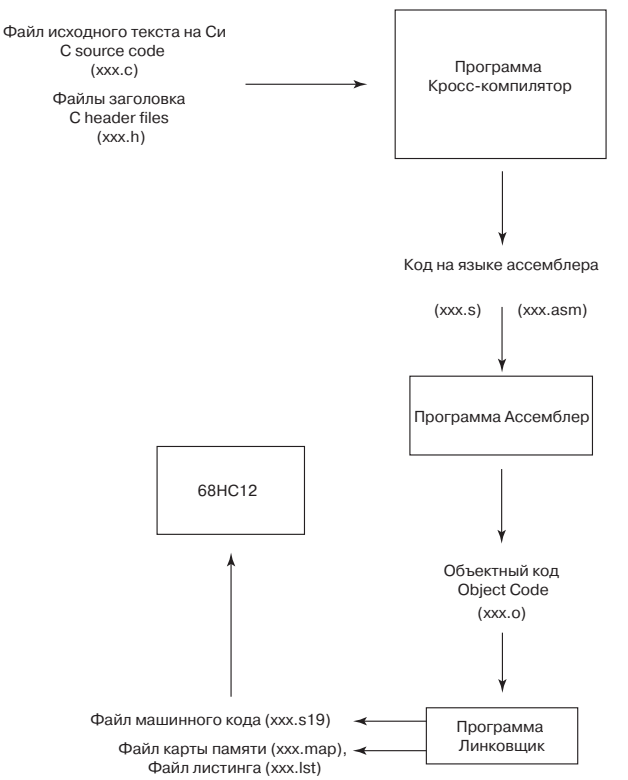


Рис. 3.3. *Последовательность работы программ в процессе генерации файла исполняемого кода прикладной программы*

отладки программного обеспечения (ПО) микропроцессорных систем. Кроме используемого в книге пакета ICC12, для создания прикладного ПО микроконтроллеров семейства 68HC12 могут также использоваться другие аналогичные пакеты, например CodeWarrior компании Metrowerks.

В параграфе 3.14 на примере простой программы управления светодиодами мы рассмотрим особенности генерации всех промежуточных файлов при работе комплекта программ в составе интегрированной среды ICC12. В настоящем параграфе на рис. 3.4 мы демонстрируем лишь смысловые преобразования исходного текста на Си в процессе получения исполняемого кода прикладной программы.

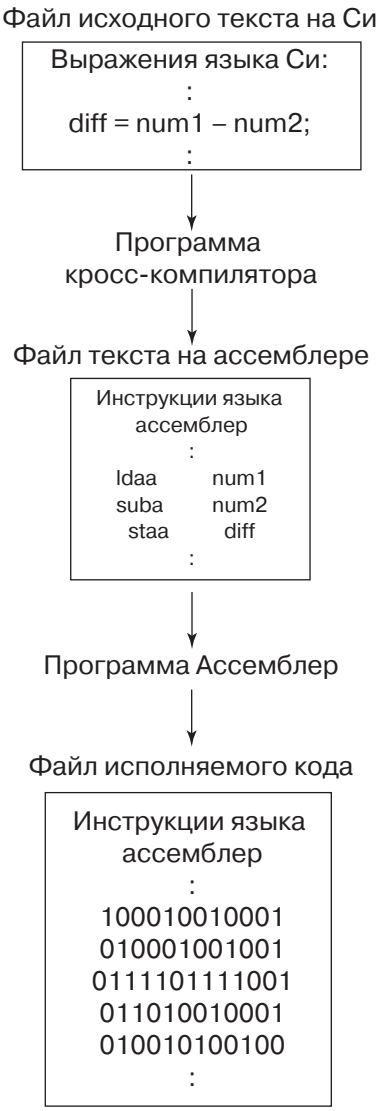


Рис. 3.4. *Форматы представления прикладной программы на разных этапах создания файла исполняемого кода*

3.13.2. Режим отладки BDM

В отличие от МК предыдущего поколения, например, 68HC11, аппаратные средства МК семейства 68HC12 позволяют вести отладку без остановки выполнения прикладной программы. В микроконтроллерах семейства 68HC11 наблюдение за состоянием внутренних ресурсов МК в процессе отладки осуществлялось с использованием специальной программы монитора отладки, которая загружалась в память МК сразу после включения питания. Эта программа передавала в персональный компьютер содержимое регистров центрального процессора и ячеек памяти каждый раз, когда этого пожелает оператор. Однако каждое обращение к программе монитора для нового просмотра вызывало генерацию программного прерывания с последующим остановом отлаживаемой прикладной программы и запуском программы монитора отладки. Такой принцип организации отладки не позволял наблюдать в реальном времени функционирование прикладных программ с множеством аппаратных прерываний.

В МК семейства 68HC12 реализован иной, более совершенный режим отладки BDM (Background Debug Mode), что в переводе означает «фоновый режим отладки». Этот режим позволяет выполнить основные процедуры отладки – просмотр и модификацию содержимого регистров и ячеек памяти без останова выполнения прикладной программы.

В процессе отладки МК обменивается данными с персональным компьютером, используя последовательный интерфейс с оригинальным протоколом. Для подключения микроконтроллера, который установлен на плате проектируемого устройства, к персональному компьютеру разработан унифицированный интерфейс, который носит название «BDM порт» (рис. 3.5). Стандартизация линий связи и типа разъема интерфейса отладки BDM позволяет разрабатывать универсальные программные пакеты и аппаратные средства отладки так, что любая аппаратная платформа на основе МК семейства 68HC12, в том числе и плата собственной разработки, способна работать под управлением любой интегрированной среды разработки для 68HC12/HCS12.

Используя порт BDM, встроенный в МК блок отладки принимает от персонального компьютера команды отладки и возвращает в персональный компьютер запрашиваемые данные. Часть команд отладки может выполняться только аппаратными средствами блока BDM, без остановки выполнения прикладной программы. При этом ис-

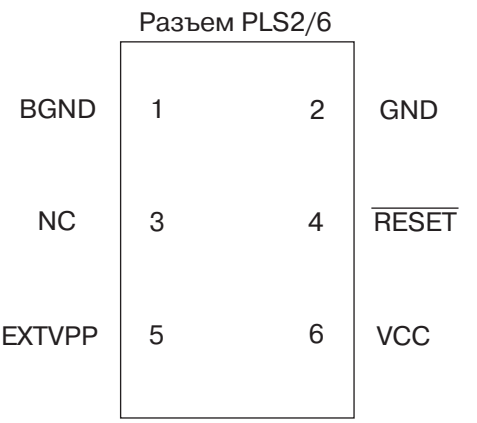


Рис. 3.5. *Цоколевка разъема BDM порта*

пользуются «холостые» машинные циклы внутренних магистралей, когда исполняемая прикладная программа не производит обращения к памяти. Если такие «холостые» циклы не возникают в течение 128 машинных циклов, то блок BDM захватыва-

№ команды	Имя команды	Код операции	Данные
1	BACKGROUND	90	– (нет)
2	READ_BD_BYTE	E4	16 бит адреса, 16 бит данных (вывод)
3	STATUS	E4	FF01, 00000000 (вывод)
4			FF01, 10000000 (вывод)
5			FF01, 110000000 (вывод)
6	READ_BD_WORD	EC	16 бит адреса, 16 бит данных (вывод)
7	READ_BYTE	E0	16 бит адреса, 16 бит данных (вывод)
8	READ_WORD	E8	16 бит адреса, 16 бит данных (вывод)
9	WRITE_BD_BYTE	C4	16 бит адреса, 16 бит данных (ввод)
10	ENABLE_FIRMWARE	C4	FF01, 1xxxxxxx (ввод)
11	WRITE_BD_ WORD	CC	16 бит адреса, 16 бит данных (ввод)
12	WRITE_BYTE	C0	16 бит адреса, 16 бит данных (ввод)
13	WRITE_ WORD	C8	16 бит адреса, 16 бит данных (ввод)

Табл. 3.3. *Команды отладки, исполняемые аппаратными средствами модуля отладки BDM*

Имя регистра: STATUS Адрес: $FF01

7	6	5	4	3	2	1	0
ENBDM	BDMACT	ENTAG	SDV	TRACE	0	0	0
0	0	0	0	0	0	0	0

Сброс:

Рис. 3.6. *Формат регистра состояния модуля отладки BDM*

№ команды по табл.3.3	Описание команды
1	Ввод в режим отладки с использованием монитора BDM
2	Чтение байта из области памяти блока BDM. Адрес указан в команде. Если адрес четный, то искомый байт содержится в старшем байте возвращаемого 16-ти разрядного слова. Если адрес нечетный, то искомый байт в младшем байте 16-разрядного слова.
3	Частный случай команды READ_BD_BYTE. Производится чтение регистра состояния модуля BDM. Считанный код 00000000 означает, что МК не может быть переведен в режим отладки и работает только под управлением прикладной программы.
4	Частный случай команды READ_BD_BYTE. Производится чтение регистра состояния модуля BDM. Считанный код 10000000 означает, что режим отладки разрешен и МК может быть переведен в режим отладки инструкцией BACKGROUND из табл. 3.3.
5	Частный случай команды READ_BD_BYTE. Производится чтение регистра состояния модуля BDM. Считанный код 11000000 означает, что МК находится в режиме отладки.
6	Чтение 16-разрядного слова из области памяти блока BDM по указанному в команде адресу. Адрес должен быть четным и указывать на старший байт возвращаемого из МК слова.
7	Чтение байта из области памяти МК. Адрес указан в команде. Если адрес четный, то искомый байт содержится в старшем байте возвращаемого 16-ти разрядного слова. Если адрес нечетный, то искомый байт в младшем байте 16-разрядного слова.
8	Чтение слова из области памяти МК по указанному в команде адресу. Адрес должен быть четным и указывать на старший байт возвращаемого из МК слова.
9	Запись байта в область памяти блока BDM. Если в команде указан четный адрес, то байт для записи содержится в старшем байте передаваемого 16-ти разрядного слова. Если адрес нечетный, то байт для записи – в младшем байте 16-разрядного слова.
10	Частный случай команды WRITE_BD_BYTE. Производится запись в регистр состояния STATUS модуля BDM. Передаваемый в регистр состояния код 1xxxxxxx разрешает работу программно-аппаратных средств модуля отладки BDM. Для перевода МК в режим отладки необходимо далее подать команду BACKGROUND из табл. 3.3..

Табл. 3.4. *Описание аппаратных команд модуля отладки BDM*

11	Запись слова в область памяти блока BDM по указанному в команде адресу. Адрес должен быть четным и указывать на старший байт передаваемого в память BDM слова.
12	Запись байта в область памяти МК. Если в команде указан четный адрес, то байт для записи содержится в старшем байте передаваемого 16-ти разрядного слова. Если адрес нечетный, то байт для записи – в младшем байте 16-разрядного слова.
13	Запись слова в область памяти МК по указанному в команде адресу. Адрес должен быть четным и указывать на старший байт передаваемого в память МК слова.

Табл. 3.4. *Описание аппаратных команд модуля отладки BDM (продолжение)*

Имя команды	**Код операции**	**Данные**	**Описание**
GO	08	–	Исполнять прикладную программу
TRACEl	10	–	Выполнить одну команду прикладной программы и вернуться в монитор отладки
TAGGO	18	–	Разрешить режим отладки и вернуться к исполнению прикладной программы
WRITE_NEXT	42	16 бит данных (ввод)	X=X+2. Записать следующее слово по 0,X.
WRITE_PC	43	16 бит данных (ввод)	Записать данные в счетчик команд
WRITE_D	44	16 бит данных (ввод)	Записать данные в аккумулятор D
WRITE_X	45	16 бит данных (ввод)	Записать данные в регистр X
WRITE_Y	46	16 бит данных (ввод)	Записать данные в регистр Y
WRITE_SP	47	16 бит данных (ввод)	Записать данные в указатель стека
READ_NEXT	62	16 бит данных (вывод)	X=X+2. Читать следующее слово по 0,X.
READ_PC	63	16 бит данных (вывод)	Читать счетчик команд
READ_D	64	16 бит данных (вывод)	Читать аккумулятор D
READ_X	65	16 бит данных (вывод)	Читать регистр X
READ_Y	66	16 бит данных (вывод)	Читать регистр Y
READ_SP	67	16 бит данных (вывод)	Читать указатель стека

Табл. 3.5. *Команды отладки, исполняемые монитором BDM*

ет последующие циклы для выполнения поступившей команды отладки. При этом выполнение прикладной программы слегка притормаживается. Обсуждаемые так называемые аппаратные команды отладки могут поступать в блок BDM от персонального компьютера не чаще, чем 1 раз в 150 машинных циклов. Перечень аппаратных команд отладки представлен в табл. 3.3. В табл. 3.4. дано описание этих команд.

Другая часть команд отладки исполняется под управлением программы монитора отладки, которая хранится в ПЗУ модуля BDM. Это ПЗУ располагается в общем адресном пространстве МК по адресам 0xFF00...0xFFFF. Память блока BDM доступна только в режиме отладки. В рабочем режиме ячейки памяти с этими адресами используются для других целей, в частности для размещения векторов прерываний. Перечень команд, исполняемых монитором отладки, представлен в табл. 3.5.

Для того, чтобы использование команд монитора отладки стало возможным, необходимо сначала установить бит ENBDM в регистре состояния STATUS (0xFF01), а затем выполнить команду BACKGROUND. Формат регистра состояния STATUS представлен на рис. 3.6.

Запись бита разрешения работы монитора отладки ENBDM осуществляется командой ENABLE_FIRMWARE из перечня аппаратных команд BDM (табл. 3.3). Отметим, что если МК 68HC12B32 сконфигурирован для работы в однокристальном режиме, то во время сброса бит ENBDM устанавливается в 1. Аппаратная команда отладки BACKGROUND также передается из персонального компьютера. Эта команда переводит МК в режим работы под управлением монитора отладки, когда центральный процессор на время прекращает выполнение основной программы и реализует команды монитора отладки. Для выполнения команд монитора отладки модуль BDM анализирует состояние внутренних магистралей МК. Если команда монитора требует для реализации только один машинный цикл, то работа прикладной программы не нарушается. Если же монитору необходимо несколько циклов, то работа процессора приостанавливается до завершения выполнения отладочной команды.

В области памяти модуля BDM расположены пять служебных регистров (табл. 3.6). Регистр INSTRUCTION хранит переданный из персонального компьютера код исполняемой команды отладки.

Адрес	Имя регистра
0xFF00	INSTRUCTION – регистр кода выполняемой команды BDM
0xFF01	STATUS – регистр состояния блока BDM
0xFF02-0xFF03	SHIFTER – данные, передаваемые блоком BDM
0xFF04-0xFF05	ADDRESS – адрес регистра или ячейки памяти BDM
0xFF06	CCRSAV – содержимое регистра признаков CCR

Табл. 3.6. *Регистры модуля отладки BDM*

Регистр состояния STATUS (рис. 3.6) отражает текущий режим работы модуля BDM. Бит ENBDM установлен, если работа программы монитора отладки разрешена, т.е. могут реализовываться не только аппаратные, но и программно исполняемые команды отладки. Установленный в 1 бит BDMACT показывает, что МК прекратил выполнение прикладной программы и ожидает поступления команды отладки. Бит ENTAG отражает перевод МК в специальный режим тегирования команд. Этот режим устанавливается после исполнения команды TAGGO монитора отладки (табл. 3.5). Бит SDV является служебным битом монитора отладки, он отражает наличие данных в регистре SHIFTER блока BDM. И, наконец, бит TRACE – это признак работы МК в режиме трассировки, который назначается после исполнения инструкции TRACE1 из списка табл. 3.5.

Регистр сдвига SHIFTER предназначен для хранения данных, передаваемых или получаемых модулем отладки по последовательному интерфейсу.

Регистр ADDRESS хранит принятый в команде отладки адрес регистра или ячейки памяти. В регистре CCRSAV сохраняется состояние регистра признаков CCR центрального процессора во время исполнения команд монитора отладки.

Режим тегирования используется для автоматического перевода МК в режим отладки при исполнении команды, которая ранее была отмечена программистом для более подробного рассмотрения результатов ее исполнения.

Обсудив принципы организации режима отладки для МК семейства 68HC12, мы перейдем к рассмотрению программно-аппаратных средств отладки, которые используют режим BDM и обеспечивают простой и удобный интерфейс пользователя.

3.13.3. Аппаратные и программные средства отладчика P&E от компании PEMICRO

В данном параграфе представлены краткие сведения об аппаратных и программных средствах отладки для МК семейства 68HC12, которые используют порт модуля BDM для связи с микроконтроллером. Мы остановимся на описании возможных режимов отладки с использованием пакета внутрисхемного отладчика ICD12Z в составе интегрированной среды разработки WinIDE Pemicro HC12. Отладчики от других производителей работают схожим образом. Используя набок предоставляемых команд отладки, пользователь может обнаружить и зафиксировать ошибки в исполнении программы. Набор команд отладки приведен в табл. 3.7 и 3.8.

На рис. 3.7. представлен необходимый для организации процесса отладки набор аппаратных средств. На рис. 3.8. показан вид экрана монитора компьютера в процессе использования пакета отладчика ICD12Z. Как показано на рис., пользователь имеет доступ к регистрам центрального процессора (левое верхнее окно), может наблюдать за изменением используемых в программе символьных переменных (среднее верхнее окно), следить за состоянием и изменять по желанию коды в ячейках памяти (правое верхнее окно), исходный текст отлаживаемой программы ((два средних окна), осуществлять ввод команд отладки и наблюдать за их исполнением в окне состояния (нижнее окно).

Вы можете также организовать процесс отладки, используя другие аппаратные средства, например две платы MC68HC912B32EVB. На рис. 3.9. показана инсталляция аппаратных средств для этого случая. Одна из отладочных плат используется в качестве отладочного интерфейса BDM между персональным компьютером и платой, которая подлежит отладке. К плате MC68HC912B32EVB прилагается программное

обеспечение – Motorola D-Bug12 монитор, который и будет использован для управления процессом отладки. Для того, чтобы воспользоваться таким режимом работы платы MC68HC912B32EVB, следует установить переключатели W3 и W4 платы в состояние 0 и 1 соответственно. Далее подсоединить кабель BDM от платы интерфейса к оставшейся плате MC68HC912B32EVB. Эта вторая плата будет платой целевой системы. Подключите источник питания к целевой системе. При этом плата интерфейса отладки будет питаться от этого же источника, используя BDM кабель. Теперь Вы можете использовать команды монитора отладки Motorola D-Bug12 для управления исполнением испытуемой прикладной программой.

3.13.4. Эмуляторы

Другой способ отладки прикладной программы – использование программного пакета класса эмулятор, который изображает ход исполнения прикладной программы и при этом генерирует все аппаратные сигналы, которые были бы на выходе реального МК. Такие эмуляторы выпускают компании Noral и Hitex.

Эмуляторы полезны для тех пользователей, которые не могут купить настоящих аппаратных средств отладки. Вместе с тем, эмуляторы не следует использовать для завершающих этапов проверки работоспособности тех систем, которые жестко привязаны к реальному времени. Эмуляторы, исходя из своего принципа действия, используют персональный компьютер. Поскольку частота тактирования персонально-

Имя команды	Описание
A или ACC	Установить значение аккумулятора A
B	Установить значение аккумулятора B
BR	Установить контрольную точку
CCR	Установить значение регистра признаков
CLEARSYMBOL	Очистить массив символов
CODE	Показать дизассемблированный код в окне отладчика «Code window»
DASM	Дизассемблировать инструкцию
DUMP	Отобразить память в окне журнала отладки «Status window»
EXIT	Выйти в DOS
G или GO	Запустить программу на исполнение
GONEXT	Выполнить, начиная с текущего состояния счетчика PC до начала следующего оператора

Табл.3.7. *Команды интерфейса пользователя отладчика P&E*

GOTILROM	Выполнить, начиная с текущего состояния счетчика PC до достижения указанного адреса в ПЗУ
HELP	Показать справочную информацию
IX	Установить значение индексного регистра X
LF или LOGFILE	Открыть или закрыть файл журнала отладки
LOADALL	Выполнить команды загрузки LOAD и LOADMAP
LOADV	Выполнить команды загрузки LOAD и побайтового сравнения VERIFY
MACRO	Выполнить файл макрокоманд
MACROSTART	Начать запись файла макрокоманд
MD или MDx	Отобразить содержимое ячеек памяти в окне «Memory window»
N	Установить/сбросить бит знака N в регистре признаков CCR
REG	Отобразить регистры центрального процессора в окне журнала отладки «Status window»
RTVAR	Отобразить заданный адрес и содержимое ячейки с этим адресом в окне переменных «Variable window»
S	Установить/сбросить бит S в регистре признаков CCR
SERIAL	Установить параметры обмена для последовательного порта
SERIALON	Открыть окно интерфейса связи с отладочной платформой
SS	Выполнить один оператор программы на языке исходного текста
STEP or ST or T	Выполнить один оператор (команда пошаговой отладки)
STEPTIL	Выполнять команду пошаговой отладки, начиная с текущего состояния счетчика PC до заданного адреса
T [n]	Выполнить заданное число n команд пошаговой отладки
TRACE	Запустить программу на исполнение и включить режим трассировки
V	Установить/сбросить бит переполнения V в регистре признаков CCR
VERIFY	Сравнить содержимое памяти программ МК с кодами файла в формате S19
WHEREIS	Отобразить код названного символа
Z	Установить/сбросить бит нулевого результата Z в регистре признаков CCR

Табл.3.7. *Команды интерфейса пользователя отладчика P&E (продолжение)*

Имя команды	Описание
ASM [add]	Записать в память по заданному адресу код введенной команды
BELL	Подать звуковой сигнал
BF	Заполнить блок памяти константой
C	Установить/сбросить бит нулевого переполнения C в регистре признаков CCR
CLEARMAP	Очистить файл карты памяти
COLORS	Изменение цветовой гаммы интерфейса пользователя
D	Установить значение аккумулятора D
DUMP_TRACE	Вывести содержимое памяти трассировки в окно «Debug window»
EVAL	Вычислить выражение
FILL	Заполнить блок памяти константой (аналог BF)
GOUNTIL	Выполнить программу до указанного адреса
H	Установить/сбросить бит дополнительного переноса H в регистре признаков CCR
I	Установить/сбросить бит глобальной маски прерывания I в регистре признаков CCR
IY	Установить значение индексного регистра Y
LOAD	Загрузить файл в формате S19
LOADMAP	Загрузить файл символьных имен *.map
LOAD_BIN	Загрузить файл исполняемого кода с указанного в команде адреса
LPT1,LPT2,LPT3	Выбрать параллельный порт для обмена
MACROEND	Остановить запись файла макрокоманд
MACS	Вывести перечень макрокоманд
мм or MEM	Изменить содержимое ячеек памяти
NOBR	Сбросить все контрольные точки
QUIT	Выход из программы
REM	Добавить комментарии к файлу макрокоманд
RESET	Имитировать сброс микроконтроллера
RUN	Начать исполнение программы

Табл.3.8. *Команды интерфейса пользователя отладчика P&E*

SCRIPT	Выполнить файл макрокоманд
SERIALOFF	Закрыть окно интерфейса связи с отладочной платформой
SHOWTRACE	Показать результаты трассировки
SOURCEPATH	Указать имя и путь к файлу
STATUS	Отобразить регистры центрального процессора в окне журнала отладки «Status window»
STEPFOR	Выполнить по шагам до контрольной точки
SYMBOL	Добавить символ в текущий список символьных имен
TIME	Показать время исполнения программы
UPLOAD_SREC	Обновить содержимое ячеек памяти на экране отладчика
VAR	Показать значение переменной или ячейки памяти в окне переменных «Variable window»
VERSION	Показать версию программного обеспечения
X	Установить/сбросить бит X в регистре признаков CCR

Табл.3.8. *Команды интерфейса пользователя отладчика P&E (продолжение)*

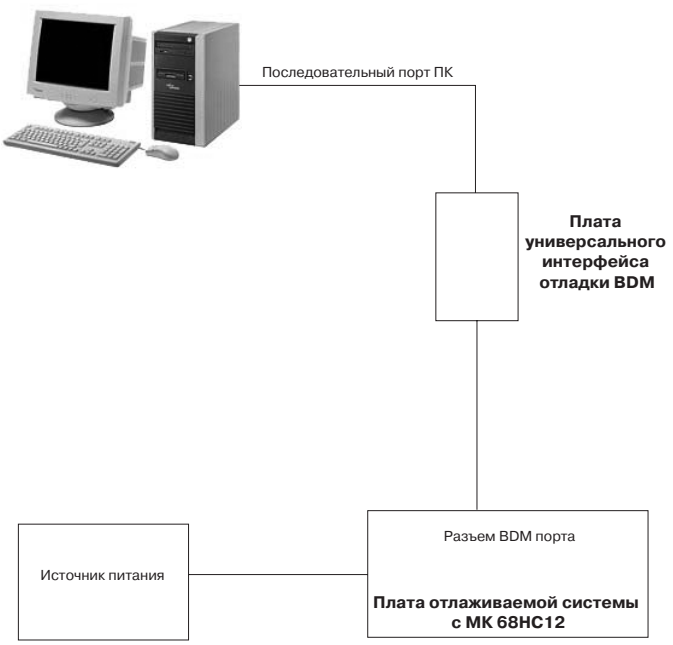


Рис. 3.7. *Система отладки на основе интерфейса BDM и платы микроконтроллером 68HC12B32*

го компьютера отличается от частоты тактирования МК HC12, то и корректную имитацию исполнения программы в реальном времени гарантировать нельзя. Однако следует заметить, что некоторые компании предлагают эмуляторы с полной имитацией всех временных характеристик МК.

3.13.5. Логические анализаторы

Логический анализатор – это интеллектуальный электронный измерительный прибор, который используется для запоминания, отображения на экране и измерения временных параметров нескольких существующих одновременно логических сигналов. Функции логического анализатора по исследованию и наладке цифровых систем аналогичны функциям осциллографа для аналоговых систем. На рис. 3.10 представлена фотография логического анализатора компании Hewlett-Packard, в настоящее время аналогичные приборы выпускаются под маркой Agilent.

Исследуемое устройство подключается к логическому анализатору посредством набора пробников со специальными пружинными контактами. Число одновременно наблюдаемых на экране логических сигналов может достигать двадцати, что позволяет разработчику проверить реальное состояние магистралей адреса, данных и управления при взаимодействии различных интегральных схем системы. Отличительная особенность логического анализатора – возможность запоминания группы логических сигналов в течение длительного времени во внутренней памяти прибора. Позднее эти сигналы могут быть воспроизведены на экране прибора с целью анализа и измерения временных параметров.

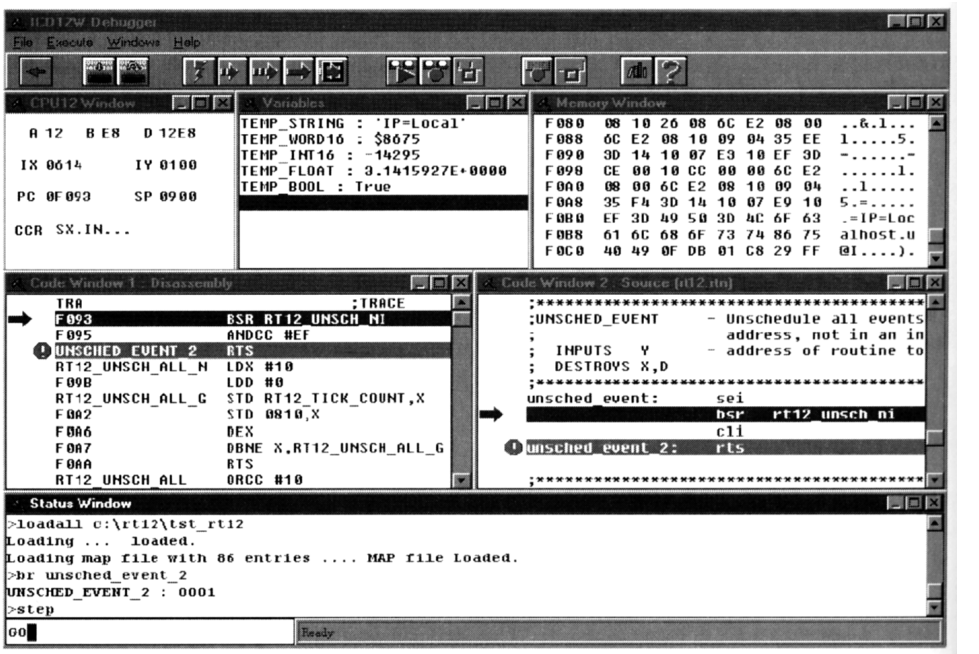


Рис. 3.8. *Интерфейс пользователя отладчика P&E ICD12Z компании PEMICRO*

Логический анализатор чрезвычайно полезен при поиске неисправностей в программе управления при взаимодействии микроконтроллера с другими периферийными ИС системы. В этом случае логический анализатор позволяет наблюдать одновременное состояние всех линий связи МК с ИС и установить, какие сигналы генерируются программой МК неверно, вследствие чего неверно реализуется обмен данными между МК и ИС.

3.14. Особенности компилятора и ассемблера

В данном параграфе мы рассмотрим пример преобразования исходного файла с программой на Си компилятором ICC12 к файлу с исполняемым кодом. Этот процесс был описан в разделе 3.13.1. Следует заметить, что данный пример демонстрирует отнюдь не все особенности механизма действия компилятора. Для получения более полных сведений следует обратиться к техническому описанию компилятора. Мы же постараемся сконцентрировать внимание читателя на ключевых моментах преобразования кодов. Эти знания необходимы Вам для того, чтобы начать работу с 68HC12, программируя их на Си.

В представленном примере прикладная программа управляет светодиодами, подключенными к выходам порта PORTA микроконтроллера 68HC12B32. Периодически, по первому сигналу переполнения таймера светодиоды зажигаются, а по следующему сигналу переполнения таймера эти светодиоды гасятся. Аппаратные средства, используемые для отладки этой задачи, представлены на рис. 3.11.

```
/*********************************************************/
/* Название: Sample.c                                    */
/* Описание: Эта программа производит включение          */
/* и выключение светодиодов с интервалом                 */
/* времени 1 с. Используется МК 68HC12B32                */
/* Файл заголовка header содержит адреса всех            */
/* портов и регистров специальных функций                */
/* Контроллер должен быть сконфигурирован                */
/* для работы в однокристальном режиме                   */
/* Дата создания: May 15, 2004                           */
/* Авторы: Daniel Pack and Steve Barrett                 */
/*********************************************************/
1      #include <68HC12B32.h>
2      /*****************************************************/
3      void TOIISR(void);
4      /*****************************************************/
5      #pragma interrupt_handler TOIISR()      /* Объявление подпрограммы
6                                 прерывания по пеполнению таймера*/

7      #pragma abs_address:0x0B1E         /*задать адрес подпрограммы
                                                прерывания ISR */
8      void (* Timer_Overflow_interrupt_vector[]) ()={TOIISR}
9      #pragma end_abs_address

10     unsigned char second = 0x00;
11     void main (void)
12     {
```

```
13     TSCR=0x80;    /*включить таймер*/
14     TMSK2=0x80;   /*разрешить прерывания по таймеру*/
15     TFLG2=0x80;   /*очистить флаг TOIF*/
16     DDRA=0xFF;    /* настроить порт Port A на вывод*/
17     CLI ();       /*разрешить прерывания*/
18     while (1) {} /*пустой бесконечный цикл - ожидание прерывания*/
19     EXIT ()
20     }
21
22     void TOIISR(void)   /*подпрограмма прерывания*/
23     {
24     TFLG2=0x80;         /*очистить флаг TOIF*/
25     second + = 1; /*увеличить на 1 программный счетчик с именем second*/
26     if (second = = 122)
27            {
28            PORTA = ! PORTA;    /*инвертировать порт PORT A*/
29            second = 0x00;      /*обнулить программный счетчик*/
30            }
31     }
```

Обратите внимание! Каждая программа должна обязательно иметь заголовок, в котором прописаны: название программы, краткое описание реализуемого алгоритма, авторы и дата создания программы. Файл заголовка 68HC12B32.h содержит определения регистров специальных функций МК B32 и макроопределения для препроцессора компилятора. Номера строк в приведенном тексте программы при вводе исходного текста в редакторе интегрированной среды ICC12 не должны присутствовать. Это наше дополнение для удобства восприятия материала.

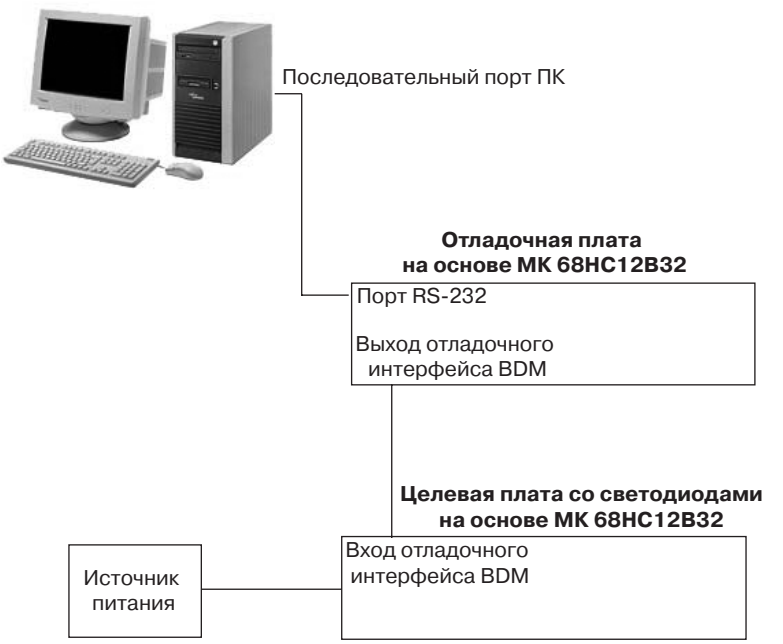


Рис. 3.9. *Система отладки на основе двух отладочных плат с микроконтроллером 68HC12B32*

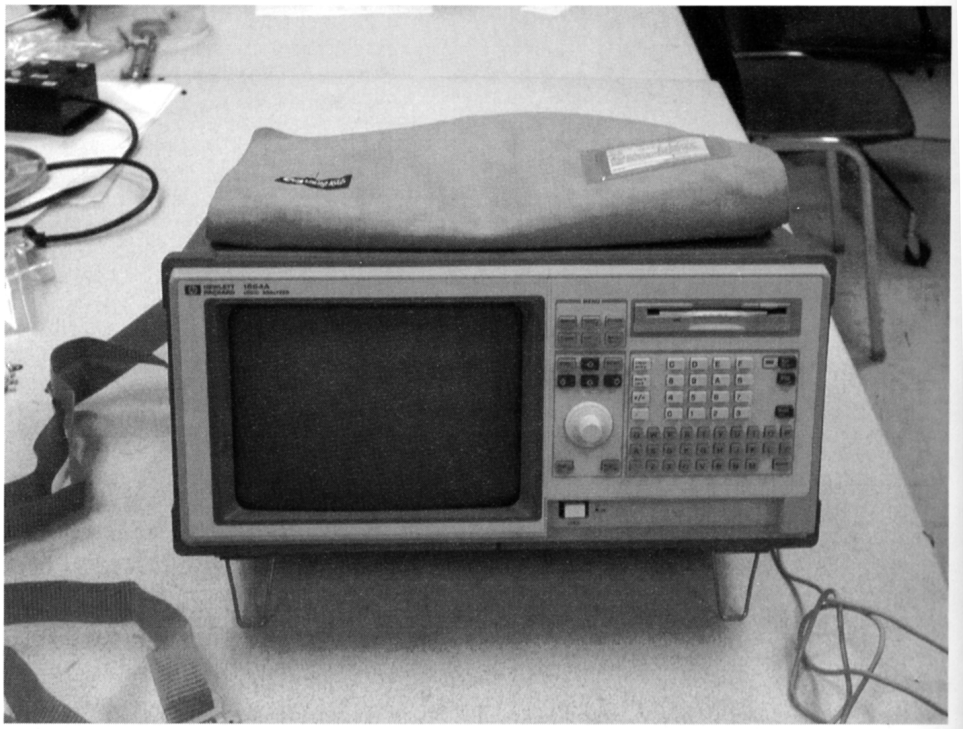


Рис. 3.10. *Логический анализатор фирмы Agilent Technologies*

Разберем назначение отдельных элементов программы. В первой строке записана директива препроцессора компилятора #include, которая предписывает присоединить к программе заголовочный файл с именем 68HC12B32.h. Содержимое этого файла мы рассмотрим ниже. Строка 3 содержит объявление функции TOIISR как подпрограммы прерывания по переполнению таймера. Строки с пятой по девятую содержат директивы, которые назначают ячейку памяти с адресом 0x0B1E для размещения в ней адреса начала подпрограммы прерывания TOIISR. В строке 10 осуществляется определение и инициализация глобальной переменной с именем second, которая будет использоваться в качестве программного счетчика. Строки с одиннадцатой по двадцатую содержат текст основной программы, в которой происходит инициализация подсистемы таймера. Таймер запускается на счет, разрешаются прерывания по его переполнению. Обратите внимание, в строке 17 вызывается макрос разрешения прерывания, который был определен в заголовочном файле. Строка 18 содержит конструкцию бесконечного цикла, который обеспечивает выполнение пустых команд микроконтроллером, пока не поступит запрос на прерывание от таймера. В строке 19 записан макрос программного прерывания EXIT(), который также определен в заголовочной файле. В строке 22 начинается подпрограмма прерывания по таймеру. В ней сбрасывается флаг переполнения таймера (строка 24), а затем инкрементируется программный счетчик second (строка 25). Заметим, что период счета 16-разрядного счетчика таймера микроконтроллера 68HC12 при частоте шины 8 МГц составляет 8,19 мс. Поэтому для отсчета 1 с требуется 122

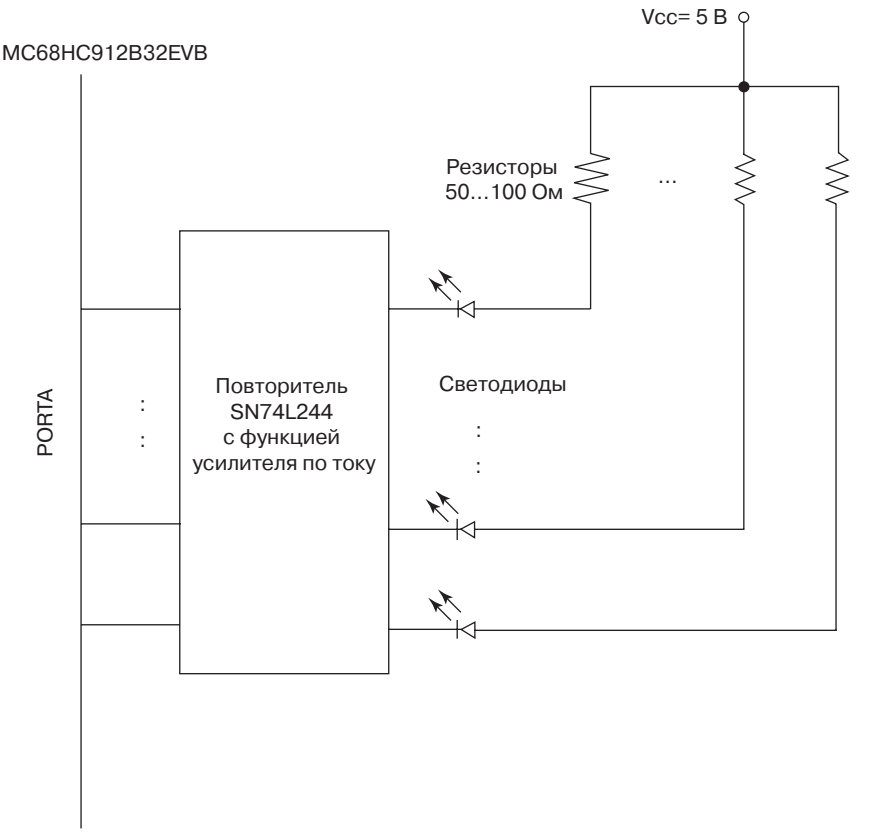


Рис. 3.11. *Схема подключения светодиодов к микроконтроллеру 68HC912B32*

периода переполнения этого таймера. В строке 26 записана конструкция условия if. Выражения строк 28 и 29 будут исполняться, только если счетчик second достиг значения 122. Тогда код на линиях PORTA будет инвертирован, а содержимое программного счетчика обнулено.

Обратимся теперь к разъяснению содержимого заголовочного файла. Мы не будем приводить его текст целиком, а приведем лишь те строки, которые необходимы для рассматриваемой в основном примере программы:

```
1       #define _IO_BASE 0
2       #define _P(off)  *(unsigned char volatile*) (_IO_BASE + off)
3       #define TSCR  _P(0x86)
4       #define TMSK2 _P(0x8D)
5       #define TFLG2 _P(0x8F)
6       #define DDRA _P(0x02)
7       #define PORTA _P(0x00)
8       #define CLI( )  asm(«cli\n»)
9       #define EXIT( )   asm(«swi\n»)
```

Две первые строки приведенного фрагмента заголовочного файла используются для определения макроса _P с аргументом off. Обратите внимание на символ указателя в макросе. Все следующие выражения в строках с 3 по 7 определяют численные значения для символьных обозначений регистров специальных функций МК. Эти численные значения – адреса регистров в соответствии с картой памяти МК. Любое упоминание имен регистров в тексте программы связано с выполнением операций чтения или записи в эти регистры по их физическим адресам. Этим объясняется необходимость применения указателя в определении макроса _P(off). Две последние строки 8 и 9 являются примерами определения макросов.

Вернемся к примеру управления светодиодами. После обработки программой компилятора исходного текста программы Sample.c будет получен следующий текст программы на языке ассемблера.

```
1      .module interrupt.c
2      .area memory(abs)
3      .org 0xb1e
4  _Timer_Overflow_interrupt_vector::
5      .word _TOIISR
6      .area data
7  _second::
8      .blkb  1
9      .area idata
10 .byte 0
11 .area data
12     .area text _main::
13 ; #include <383HC12-ver1.h>
14 ; void TOIISR(void);
15 ; #pragma  interrupt_handler TOIISR() ;
16 ; #pragma  abs_address:0x0B1E
17 ; void (*Timer_Overflow_interrupt_vector[]) ()={TOIISR};
18 ; #pragma end_abs_address ;
19 ; unsigned char second=0x00;
20 ;
21 ;void main(void)
22 ;{
23 ;   TSCR=0x80;
24            ldab #128
25            stab 0x86
26 ;   TMSK2=0x80;
27            ldab #128
28            stab 0x8d
29 ;   TFLG2=0x80;
30            ldab #128
31            stab 0x8f
32 ;   DDRA=0xFF;
33            ldab #255
34            stab 0x2
35 ;   CLI();
36            cli
37  L3: L4:
38  bra L3
39  X0:
40 ;   while(1) {};
```

```
41 ;   EXIT();
42             swi
43 ;  }L2
44 .dbline 0
45 ;   func end
46  rts
47 _TOIISR: :
48     void  TOIISR(void)
49 ; {
50 ;   TFLG2=0x80;
51             ldab #128
52             stab 0x8f
53 ;   second  + = 1;
54             ldab  _second
55             clra
56             addd #1
57             stab _second
58 ; if(second = = 122)
59             1dab _second
60             cmpb #122
61     bne    L7
62 ; {
63 ; PORTA = ~ PORTA;
64 ; vol
65             1dab 0
66             c1ra
67             coma
68             comb
69             stab  0
70 ; second = 0x00;
71             c1r _second
72 ; }
73 L7:
74 ; }
75 L6:
76 .dbline 0
77 ; func end
78 rti
```

Итак, мы видим текст после обработки кросс-компилятором, который содержит инструкции команд на языке ассемблера микроконтроллера 68HC12 и директивы языка Ассемблер в составе интегрированной среды разработки ICC12. Директивы программы Ассемблер - это специальные команды, которые осуществляют управления процессом генерации кодов команд для МК при обработке приведенного выше текста программой Ассемблер. В среде ICC12 директивы выделяются точкой перед их именем. Например: .area data или .byte 0. Разберем текст представленного файла.

В строке 1 записана директива Ассемблера, определяющая название программы. Директивы .area и .org генерируются при обработке строки 7 исходной программы на Си: #pragma abs_address:0x0B1E. Они устанавливают адрес ячейки памяти для записи адреса начала подпрограммы прерывания по переполнению таймера. Это адрес принято называть вектором прерывания. Для микроконтроллеров семейства 68HC12 ячейки памяти для хранения вектора прерывания от каждого источника запросов определены техническим описанием. В частности для МК модели

68HC12B32 в ячейке памяти с адресом 0x0B1E хранится вектор прерывания по переполнению таймера. Компилятор среды ICC12 добавляет символ подчеркивания перед именами идентификаторов исходного кода на Си (это имена переменных и функций). Это можно наблюдать в строках 4, 7 и 47. Два двоеточия после имени переменной отражают тот факт, что эти переменные доступны из всех программ, т.е. из текущей функции и из всех внешних функций. Директива .word в строке 5 производит запись адреса начала подпрограммы прерывания с именем TOIISR в ячейку памяти разрядностью в 2 байта. Директивы в строках с 7 по 11 выделяют память для хранения переменной second и инициализируют ее нулевым значением.

Начиная с линии 12 можно видеть сгенерированные кросс-компилятором команды ассемблера микроконтроллера 68HC12, соответствующие основной программе. Заметьте, что все записи исходного текста на Си из основной программы перенесены в текст ассемблерной программы (строки с 13 по 23), но перед ними установлен символ «точка с запятой». Это означает, что эти записи переведены в статус комментария, что удобно при чтении программы. Аббревиатуры команд ассемблера начинаются со строки 24. Причем в строке 23 написана инструкция Си, которая присваивает регистру управления таймером TSCR значение 0x80. Ниже в строках 24 и 25 записаны две команды ассемблера, которые реализуют данное действие. Причем, кросс-компилятор уже использовал заголовочный файл для определения абсолютного адреса регистра управления TSCR как 0x86. Строки с 26 по 36 завершают процесс инициализации микроконтроллера, но уже на языке ассемблерных команд. В строках 37...40 записаны команды бесконечного цикла. Строки с 41 по 46 завершают ассемблерный текст основной программы. Обратите внимание, макросы CLI () и EXIT () генерируют ассемблерные команды cli и swi соответственно. Основная программа оформлена в виде подпрограммы и завершается командой возврата из подпрограммы rts. В строке 47 начинается программа прерывания по переполнению таймера. Анализируя ее текст, можно проследить соотвествие команд ассемблера каждому оператору исходного текста на Си. Подпрограмма прерывания завершается командой rti в строке 78.

Следующий шаг в процессе генерации исполняемого машинного кода – это генерация объектного кода (файл interrupt.o) из рассмотренного ассемблерного исходного кода. После обработки программой Ассемблер среды ICC12 рассмотренного текста будет получен следующий объектный код:

```
XH
H4 areas 4 global symbols
M interrupt.c

A text size 3D flags 0

S _main Def0000
S _TOIISR Def001A
A memory size B20 flags C

S _Tirmer_Overflow_interrupt_vector Def0B1E
A data size 1 flags 0

S _second Def0000
A idata size 1 flags 0
```

```
T 0B 1E 00 1A
R 00 00 00 01 00 02 00 00
T 00 00 00
R 00 00 00 03

T 00 00 C6 80 7B 00 86 C6 80 7B 00 80 C6 80 7B
R 00 00 00 00

T 00 00 00 8F C6 FF 7B 00 02 10 EF 20 FE 3F 30 C6
R 00 00 00 00

T 00 1B 80 7B 00 8F F6 00 00 87 C3 00 01 7B 00 00

R 00 00 00 00 00 07 00 02 00 OE 00 02

T 00 29 F6 00 00 C1 7A 26 OC F6 00 00 87 41 51 7B

R 00 00 00 00 00 03 00 02

T 00 37 00 00 79 00 00 OB

R 00 00 00 00 00 05 00 02
```

Заметим, что в верхней половине представленного объектного кода, содержатся директивы для программы линковщика, а в нижней половине читатель может увидеть шестнадцатеричные коды инструкций ассемблера МК семейства 68HC12.

На заключительной стадии представленный выше объектный код обрабатывается программой линковщика. В результате формируются три файла: interrupt.lst, interrupt.map и interrupt.s19.

Файл листинга программы interrupt.lst представляет собой текстовый файл, который содержит команды ассемблера, машинные коды этих команд и абсолютные адреса в памяти микроконтроллера, в которых эти коды располагаются. Сгенерированный линковщиком файл листинга представлен ниже:

```
                        .module interrupt.c
                        .area memory(abs)
                        .org 0xb1e
OB1E   _                _Timer_Overflow_interrupt_vector: :
0B1E   8044             .word _TOIISR
                        .area data
0800                    _second::
0800                    .blkb  1
                        .area idata
--- 0000 00             .byte 0
                        .area data
                        .area text
802A                    _main: :
                        ;#include <383HC12-ver1.h>
                        ;void TOIISR(void) ;
                        ;#pragma interrupt_handler TOIISR()
                        ;
                        ;#pragma abs_address:0x0B1E
                        ;void(*Timer_Overflow_interrupt_vector[]) ()={TOIISR};
```

```
                    ;#pragma end_abs_address
                    ;
                    ;unsigned char second=0x00;
                    ;
                    ;void main(void)
                    ;{
                    ;TSCR=Ox80;
802A   C680         ldab #128
802C   7BO086       stab 0x86
             ;TMSK2=Ox80;
802F   C680         ldab #128
8031   7B008D       stab Ox8d
             ;TFLG2=Ox80;
8034   C680         ldab #128
8036   7B008F       stab Ox8f
             ;DDRA=OxFF;
8039   C6FF         ldab #255
803B   7B0002       stab 0x2
             ;CLI();
803E   10EF         cli

8040                L3:
8040                L4:
8040   20FE         bra L3
8042                X0:
             ;while (1) {};
;EXIT();
8042 3F                   swi
             ;      }
8043                L2:
8043                .dbline 0; func end

8043   3D           rts
8044                _TOIISR: :
                    ;
                    ; void TOIISR(void)
{
;TFLG2=0x80;
8044   C680         ldab #128
8046   7B008F       stab 0x8f
;second += 1;
8049   f60800       ldab _second
804C   87           clra
804D   C30001       addd #1
8050   7B0800       stab _second
;if(second == 122)
8053   F60800       ldab _second
8056   C17A         cmpb #122
8058   260C         bne L7
;{
;PORTA = ~PORTA;
; vol
805A   F60000       ldab 0
805D   87           clra
805E   41           coma
```

```
805F   51           comb
8060   7B0000       stab 0
;second = 0x00;
8063   790800       clr _second
                    ;}
8066          L7:
;}
8066          L6:
8066          .dbline 0; func end
8066   0B           rti
```

Файл листинга обычно используется в процессе отладки прикладной программы при выявлении несоответствий между задуманными программистом действиями и реальным ходом исполнения программного кода. Кроме того, в процессе отладки иногда полезно знать, какие коды должны быть расположены в ячейках с фиксированными адресами. Последнюю информацию наиболее удобно получить из файла карты памяти *.map (иногда этот файл называют файлом символьных меток). Пример файла карты памяти для программы sample.c приведен ниже.

```
Area
(Attributes)  Addr   Size   Decimal Bytes
   ------------------------   ----   -----        ------------        --
--------
 text  8000   006B = 107. bytes    (rel,con)

       Addr          Global Syrnbol
       -----         ---------------------
       8000          __start
       8028          _exit
       802A          _main
       8044           _TOIISR
       8067          __ HC12Setup
       806B          __ text_end

Area
(Attributes)  Addr   Size   Decimal Bytes
   ------------------------   ----   -----        ------------        --
--------
idata  806B   0001 = 1. bytes     (rel,con)

       Addr          Global Syrnbol
       -----         ---------------------
       806B          __idata_start
       806C          __idata_end

Area
(Attributes)  Addr   Size   Decimal Bytes
   ------------------------   ----   -----        ------------        --
--------
data   0800   0001 = 1. bytes     (rel,con)

       Addr          Global Syrnbol
```

```
       -----          --------------------
       0800           _second
       0800           __data_start
       0801           __data_end

Area
(Attributes)  Addr   Size   Decimal Bytes
   ------------------------   ----   -----         ------------        --
--------
memory 0000   0B20 = 2848. bytes   (abs,ovr)

       Addr           Global        Syrnbol
       -----          --------------------
       0B1E           _ Timer_Overflow_interrupt_vector

Files Linked [ module(s)]
C:\icc \lib \crt12.o       [crt12.s] interrupt.o [ interrup ]
<library>     [psetup.c]

User Global Definitions

init_sp = 0xc00

User Base Address Definitions

text = 0x8000 data =0Ox800
```

В то время, как файл карты памяти предоставляет программисту обобщенную информацию о том, в какой области памяти располагаются коды переменных и каждой подпрограммы, файл исполняемых микроконтроллером кодов interrupt.s19 имеет следующий вид:

```
S10E8000CF0C0016806787CE08018EAD
S110800B080127056A000820F6CE806BCD21
S111801808008E806C2706180A307020F516BA
S1078026802A20FE8A
S1050B1E80440D
S104806B0010
S110802AC6807B0086C6807B008DC6807BEF
S1118037008FC6FF7B000210EF20FE3F3DC607
S1118045807B008FF6080087C300017B0800D3
S1118053F60800C17A260CF600008741517B26
S109806100007908000B89
S10780677900163D45
S90380007C
```

Каждая строка файла в формате S19 содержит информацию о типе представленных в ней данных, начальный адрес данных с строке, сами данные и контрольную сумму для этой строки. Полная расшифровка формата S19 дана в руководстве пользователя для отладочной платы M68EVB912B32.

Итак, мы рассмотрели последовательность преобразования исходного файла на Си к файлу загрузочного исполняемого кода для микроконтроллера. На языке структурных диаграмм этот процесс представлен на рис. 3.4.

3.15. Заключение по главе 3

В этой главе мы кратко рассмотрели технику программирования встраиваемых систем на Си. Мы показали некоторые особенности записи программ на Си для платы отладки MC68HC912B32EVB. Безусловно, столь краткое изложение языка Си недостаточно для разработки собственных реальных встраиваемых приложений. Поэтому для расширения познаний в области программирования на Си рекомендуем Вам обратиться к специальной литературе, приведенной в разделе 3.16.

Мы также рассмотрели процесс преобразования исходного текста программы на Си в файл исполняемых кодов для МК выбранного типа. Мы показали Вам примеры всех промежуточных генерируемых файлов по ходу этого процесса. Мы также кратко остановились на аппаратных особенностях МК семейства 68HC12/HCS12 для целей отладки, рассмотрели возможности использования аппаратных и программных средств платы MC68HC912B32EVB для тестирования и отладки примеров следующих глав данной книги.

3.16. Что еще почитать?

1. Kemighan, B. W., and D. M. Ritchie. The C Programming Language, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1988.
2. Schildt, H. C: The Complete Reference. Osbome McGraw-Hill.
3. Harbison, Samuel P. III, and Guy Steele Jr. C A Reference Manual, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2002.
4. ItageCraft C Compiler and Development Environment for Motorola HC12, ImageCraft Creations Inc., Pal0 Alto, CA, 2002.

3.17. Вопросы и задания

Основные

1. Каково назначение символа «;» в программах на Си?
2. Сколько байт отводится для хранения переменной в формате integer?
3. Какое количество байт отводится для хранения переменной-указателя?
4. В какой области памяти МК семейства 68HC12 размещаются переменные со статическим классом хранения?
5. Расскажите о двух назначениях символа * в программах на Си.
6. Опишите процесс преобразования файла с исходным текстом программы на Си в файл исполняемых кодов программы.

Более сложные

1. Глобальные переменные обладают свойством доступности данных из любой функции программного модуля. Почему тогда все используемые в прог-

раммном модуле переменные не объявляют глобальными, ведь так программисту удобнее?
2. Создайте структуру для хранения записей телефонных абонентов. Каждая запись должна включать имя абонента, номер телефона, улицу и номер дома проживания, город, почтовый индекс.
3. В тексте программы на Си присутствует запись:

```
static int array[10];
```

Поясните, какие действия компилятора или управляющей программы соотносятся с этим выражением?
4. Поясните назначение макроопределения. Приведите примеры макроопределений в программах на Си.
5. В чем отличие макроопределения от функции?

Исследовательские

1. Напишите программу на Си, которая переключает состояние светодиодов из выключенного во включенное и наоборот каждые 5 с. Функциональная схема подключения светодиодов (в составе семисегментного индикатора) к МК семейства 68HC12 приведена на рис. 3.12.
2. Используя понятия структуры и указателя, напишите программу, которая выводит на экран дисплея информацию об абоненте телефонной компании, которая записана в формате, который Вы разработали в задании №2 раздела «более сложные вопросы».
3. Напишите программу для отображения на семимегментном индикаторе последовательности цифр от 0 до 9. Каждая цифра должна светиться 100 мс. Функциональная схема подключения семисегментного индикатора к МК семейства 68HC12 приведена на рис. 3.12. Семисегментный индикатор выполнен по схеме с общим катодом. Схема соединения светодиодов внутри корпуса индикатора и обозначения сегментов индикатора представлены на рис. 3.13. Аноды диодов подключаются к выходам логических буферных

Цифра шестнадцатеричной системы счисления	Кодовая комбинация	Цифра шестнадцатеричной системы счисления	Кодовая комбинация
0	0x3F	1	0x06
2	0x5B	3	0x4F
4	0x66	5	0x6D
6	0x7D	7	0x07
8	0x7F	9	0x6F
A	0x77	B	0x7F
C	0x39	D	0x3F
E	0x79	F	0x71

Табл. 3.9. *Кодовые комбинации для высвечивания цифр шестнадцатеричной системы счисления*

элементов (интегральная схема 74ALS244) через резисторы, которые служат ограничителями тока каждого сегмента (светодиода) индикатора. Входы интегральной схемы 74ALS244 подключены к выходам порта МК. Для формирования на индикаторе образов цифр и букв необходимо вывести под управлением программы на выводы порта МК кодовые комбинации, которые перечислены в табл. 3.9.

В процессе создания программы разработайте структуру программы, блок-схему алгоритма, псевдокод. Напишите функцию, которая получает в качестве параметра номер отображаемой цифры и формирует на выходе порта PORTB соответствующий код засветки. Напишите функцию отсчета задержки в 100 мс. Напишите основную функцию main.c., в которой последовательно перебираются и передаются для отображения все цифры.

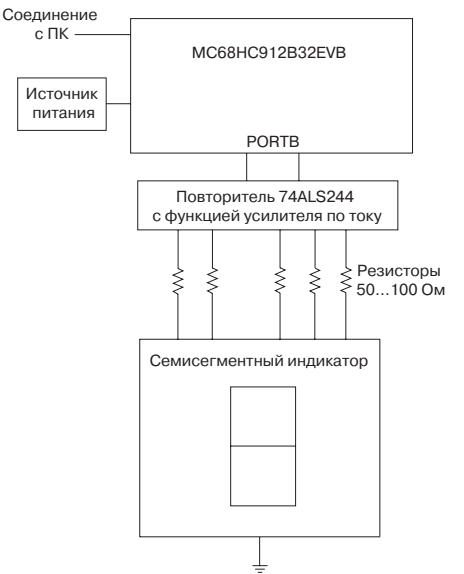


Рис. 3.12. *Схема подключения семисегментного индикатора к микроконтроллеру 68HC912B32*

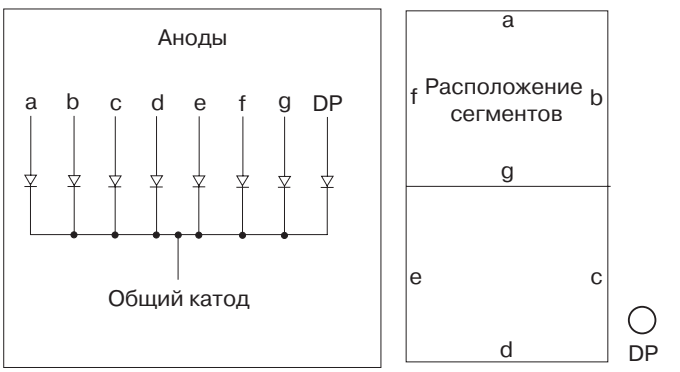


Рис. 3.13. *Семисегментный индикатор с общим катодом*

Глава 4

МИКРОКОНТРОЛЛЕРЫ 68HC12 И HCS12: АРХИТЕКТУРА И ПРОГРАММИРОВАНИЕ

ЦЕЛИ

ПОСЛЕ ИЗУЧЕНИЯ ГЛАВЫ ВЫ СМОЖЕТЕ:

- Описать структуру и определить основные отличительные особенности МК семейства 68HC12;
- Описать различные режимы работы МК 68HC12;
- Описать, какие действия выполняются над аппаратными средствами МК в состоянии сброса;
- Объяснить необходимость подсистемы прерывания в составе МК;
- Объяснить последовательность действий на аппаратном и программном уровне, которые выполняются при обработке запроса на прерывание в МК;
- Описать работу модуля тактирования в составе МК 68HC12;
- Описать структуру и режимы работы модуля таймера TIM, счетчика внешних событий и модуля меток реального времени, привести примеры программирования всех перечисленных подсистем реального времени;
- Разъяснить термины, которые используются при описании обмена данными в последовательном коде;
- Описать структуру аппаратных средств и привести примеры программирования контроллера асинхронного последовательного обмена SCI в составе МК 68HC12;
- Составить программу для обмена в последовательном коде с заданными параметрами для модуля SCI;
- Описать структуру аппаратных средств и привести примеры программирования контроллера синхронного последовательного обмена SPI в составе МК 68HC12;
- Разъяснить физический смысл процессов и терминов, связанных с аналого-цифровым преобразованием, таких, как квантование по времени и по уровню, кодирование информации, частота выборки, разрешающая способность, скорость потока данных оцифровки;

- Грамотно рассчитать параметры процесса аналого-цифрового преобразования для сигнала с заданной частотой и формой;
- Описать модуль аналого-цифрового преобразователя ATD в составе МК 68HC12;
- Разработать программу для выполнения нескольких преобразований модулем ATD по заданному сценарию;
- Подробно описать усовершенствования модуля АЦП в составе МК HCS12 по сравнению с модулем ATD в составе 68HC12;
- Составить программу и использовать модуль широтно-импульсного модулятора PWM в составе МК 68HC12B32 для управления электрическими двигателями.

4.1. Аппаратные средства микроконтроллеров семейства 68HC12....139
4.2. Аппаратные средства МК семейства HCS12 147
4.3. Режимы работы МК семейства 68HC12/HCS12 149
4.4. Назначение выводов МК 152
4.5. Регистры специальных функций МК....153
4.6. Порты ввода/вывода 157
4.7. Подсистема памяти МК B32....164
4.8. Подсистема памяти МК DP256....167
4.9. Состояния сброса и прерывания МК 168
4.10. Состояния сброса и прерывания в МК 68HC12 170
4.11. Процесс перехода к подпрограмме прерывания....183
4.12. Оформление подпрограммы прерывания на Си 187
4.13. Система тактирования....189
4.14. Подсистема реального времени – модуль таймера 191
4.15. Модуль меток реального времени 230
4.16. Модуль таймера ECT в составе МК MC68HC12BE32 и HCS12....233
4.17. Обмен информацией в последовательном коде: многофункциональный последовательный интерфейс 240
4.18. Контроллер асинхронного обмена SCI 245
4.19. Синхронный последовательный интерфейс SPI....260
4.20. Введение в теорию аналого-цифрового преобразования....274
4.21. Принцип действия АЦП....279
4.22. Подсистема аналого-цифрового преобразования МК 68HC12....282
4.23. Особенности модуля ATD в составе МК семейства HCS12 297
4.24. Подсистема широтно-импульсной модуляции....300
4.25. Ограничение энергии потребления....317
4.26. Советы по использованию платы отладки MC68EVB912B32....320
4.27. Заключение по главе 4 323
4.28. Что еще почитать? 323
4.29. Вопросы и задания 323

Знакомясь с оглавлением, Вы должны были заметить, что эта глава – самая длинная в книге. В ней мы с достаточной степенью подробности изучим структуру и режимы работы всех подсистем микроконтроллеров семейства 68HC12 и HCS12. Для определенности рассмотрение будем вести на примере двух моделей МК: MC68HC912B32 и MC9S12DP256. Далее с целью удобства восприятия будем называть эти модели просто B32 и DP256. Периферийные модули в составе МК 68HC12 и HCS12 очень похожи друг на друга. Поэтому мы сначала будем рассматривать модули МК 68HC12, а затем остановимся на отличиях конкретного модуля в составе семейства HCS12 от его прототипа в составе 68HC12. Изучению каждого периферийного модуля будет предшествовать краткая теоретическая справка, затем будут рассмотрены структура аппаратных средств и регистры специальных функций модуля. В завершение для каждого модуля приведены несколько примеров его программного обслуживания.

Очень важно, чтобы Вы достаточно глубоко поняли особенности подсистем в составе МК 68HC12 перед тем, как перейти к примерам практической реализации достаточно сложных систем на основе этих МК. Если Вы уже знакомы с МК семейства 68HC12, то у Вас возникнет желание пропустить главу 4. Однако мы советуем Вам все же ознакомиться с примерами программ управления из этой главы, поскольку последние используются в приложениях главы 7. Заметим также, что примеры этой главы могут использоваться Вами для получения навыков отладки программного обеспечения с использованием платы отладки M68EVB912B32 или каких-либо других отладочных средств.

4.1. Аппаратные средства микроконтроллеров семейства 68HC12

На рис. 4.1. представлена структура микроконтроллера MC68HC912B32 или в сокращенном виде B32. Мы уже использовали этот рисунок в гл. 1, однако обратились к нему снова, для более подробного рассмотрения технических характеристик МК B32.

Все МК семейства 68HC12 обладают 16-разрядным процессорным ядром. Семейство объединяет ряд моделей, в том числе ранее упомянутый МК 68HC812A4 (A4) и рассматриваемый в данной главе МК 68HC912B32 (B32). Отдельные модели в составе семейства различаются набором периферийных модулей, которые подключаются к внутренней межмодульной магистрали. Основные отличия между отдельными представителями семейства состоят в типе и объеме размещенной на кристалле резидентной памяти, количестве параллельных портов и контроллеров последовательных интерфейсов. В настоящей главе будет подробно рассмотрен МК модели 68HC912B32, который был выбран по причине наличия в нем всех типовых перифе-

рийных модулей. Подробное изучение предложенного МК позволит читателю с минимальными затратами адаптироваться к проектированию приложений на основе других моделей семейства 68HC12 и HCS12.

МК 68HC912B32 характеризуются следующими отличительными особенностями:

- ***Низкое энергопотребление.*** Микроконтроллеры семейства 68HC12 производятся на основе CMOS технологии (CMOS – Complementary Metal oxide semiconductor). Эта технология позволяет создать транзисторные структуры с относительно низкими потерями энергии при работе в ключе-

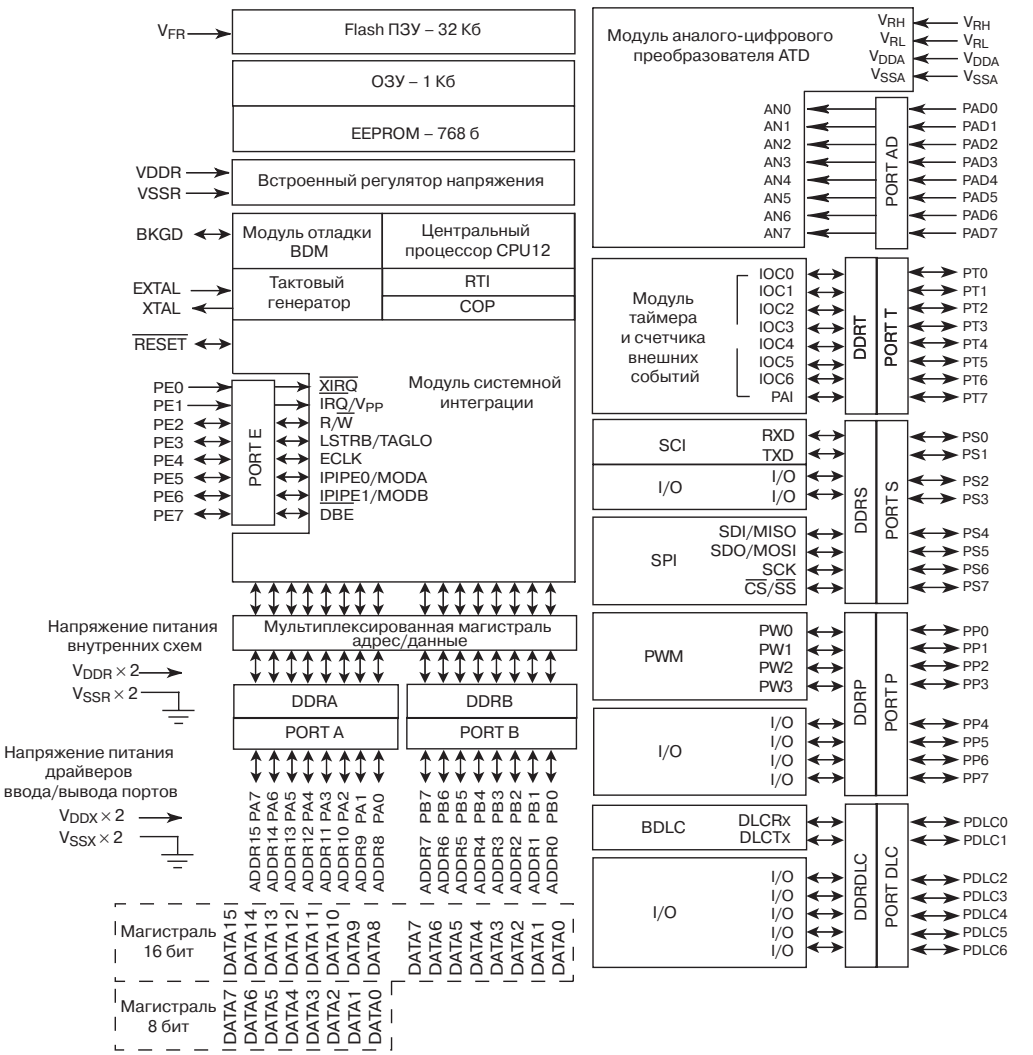


Рис. 4.1. *Структура микроконтроллера MC68HC912B32*

вых режимах. Поэтому МК семейства 68HC12 характеризуются малым потреблением энергии, что позволяет рекомендовать их для использования в изделиях с автономным питанием (от аккумуляторов или батареек). Однако не следует забывать, что потребляемая энергия для полупроводниковых CMOS структур прямопропорциональна частоте переключения. Поэтому для достижения оптимальных энергетических характеристик следует выбирать частоту тактирования центрального процессора микроконтроллеров 68HC12 минимально возможной для конкретного применения. Микроконтроллеры 68HC12 имеют специальные режимы пониженного энергопотребления, которые также позволяют оптимизировать энергетические характеристики проектируемого изделия.

- ***Высокопроизводительное 16-разрядное процессорное ядро.*** Центральный процессор семейства 68HC12 выполняет действия над 16-разрядными операндами, поэтому время выполнения алгоритмов над переменными, разрядность которых превышает байт, существенно сокращается по сравнению с 8-разрядными МК. Максимальная частота тактирования процессорного ядра f_{BUS} для микроконтроллеров семейства 68HC12 составляет 8 МГц. Однако частота внешнего генератора или кварцевого резонатора должна составлять 16 МГц, поскольку внутренние цепи микроконтроллера делят эту частоту на два для получения f_{BUS}.
- ***Резидентное ОЗУ объемом 1024 байта (1Кб).*** Встроенное в кристалл микроконтроллера ОЗУ используется для хранения промежуточных результатов вычислений. Число ячеек, равное 1К, достаточно для большинства прогнозируемых для семейства 68HC12 применений.
- ***Резидентная энергонезависимая память данных типа EEPROM объемом 768 б.*** Этот тип энергонезависимой памяти (энергонезависимая память сохраняет содержимое после отключения питания) обычно используют для сохранения изменяемых констант прикладной программы. Например, в области EEPROM могут храниться коды доступа к данной модели устройства, или на основе ячеек EEPROM могут быть организованы счетчики аварий исполнительного механизма, которым управляет микропроцессорный контроллер. Энергонезависимая память типа EEPROM позволяет выполнять операции записи и перезаписи содержимого ячеек памяти в течение сеанса работы микропроцессорного устройства под управлением прикладной программы, а также чтение ячеек памяти в произвольном порядке.
- ***Резидентная память программ типа Flash объемом 32 Кб.*** Этот тип памяти предназначен для хранения прикладной программы, которая функционально завершена, прошла отладку и тестирование на реальном объекте. Объем памяти программ МК B32 составляет 32 Кб, что позволяет разместить в ней достаточно большие программы. В процессе работы над примерами из нашей книги Вы почувствуете, какой алгоритм может быть реализован программным кодом объемом 32 Кб. Использование Flash памяти в качестве памяти программ позволяет реализовать технологию программирования в системе ISP (In System Programming). Эта технология обеспечивает выполнение операций стирания и записи новых кодов в резидентное ПЗУ программ микроконтроллера без демонтажа МК с платы конечного изделия. Учитывая, что наиболее надежным способом монтажа МК на плату встраиваемой системы в настоящее время является пайка, читатель

должен оценить, сколь полезна технология ISP. Если Вы собрались использовать технологию ISP на плате отладки MC68HC912B32EVB, то следует помнить, что программа монитора отладки D-Bug12, которая в том числе необходима для реализации этой технологии, располагается в области Flash памяти. Поскольку любой операции записи во Flash ПЗУ предшествует операция стирания Flash памяти, необходимо соблюдать осторожность, чтобы не стереть коды монитора отладки. Необходимые для этого сведения Вы найдете в разделе 7.8.

- ***Мультиплексированная шина адрес/данные для адресации внешней памяти и периферийных устройств.*** Число выводов корпуса, в котором размещается полупроводниковый кристалл микроконтроллера, ограничено. Причиной тому – стремление разработчика выпускать миниатюрные и относительно недорогие МК, в то время, как увеличение числа выводов корпуса увеличивает его размеры и стоимость. Одним из способов сокращения числа выводов корпуса МК при сохранении функций этих выводов является мультиплексирование линий магистралей адреса и данных для сопряжения МК с внешней памятью. При мультиплексировании одни и те же выводы МК на протяжении одного временного интервала используются для передачи информации об адресе внешней ячейки памяти, а в течение другого временного интервала – для обмена данными с этой ячейкой. В микроконтроллере B32 функции мультиплексированных во времени магистралей адрес/данные выполняют линии портов Port A и Port B (рис. 4.1).
- ***Многофункциональный таймер.*** Подсистема реального времени МК семейства 68HC12 включает несколько модулей, но основным является таймер с 16-разрядным счетчиком временной базы, программируемым делителем частоты тактирования и 8 каналами входного захвата IC (Input Capture) или выходного сравнения OC (Output Compare). Эти каналы могут быть сконфигурированы произвольно: любое число каналов из 8 настраивается на реализацию функции входного захвата IC, оставшиеся каналы – на функцию выходного сравнения OC. При этом возможны конфигурации, когда все каналы находятся в режиме IC или в режиме OC. Такая организация модуля таймера позволяет производить точные измерения временных характеристик входных сигналов МК, и генерировать многоканальные импульсные последовательности на его выходах.
- ***Независимый 16-разрядный счетчик внешних событий.*** Этот модуль также принадлежит к подсистеме реального времени. Он предназначен для подсчета так называемых внешних событий, каждое из которых представляется импульсом на одном из входов МК. Например, в главе 7 мы будем рассматривать пример системы управления скоростью вращения электрического двигателя. Этот двигатель оснащен оптическим датчиком скорости, который генерирует 60 импульсов на один оборот двигателя. С помощью счетчика внешних событий эти импульсы могут быть подсчитаны на определенном временном интервале, после чего МК рассчитает число оборотов двигателя в мин.
- ***Таймер меток реального времени.*** Этот модуль также относится к подсистеме реального времени. Он позволяет организовать прерывания основной программы МК для выполнения некоторых других «неотложных функ-

ций». Например, эта подсистема может быть использована для регулярного, один раз в 15 мин, опроса напряжения питающей систему батарейки с целью исключения потери информации при ее разряде. В главе 8 мы покажем использование таймера меток реального времени для обеспечения работы в микроконтроллере операционной системы реального времени.

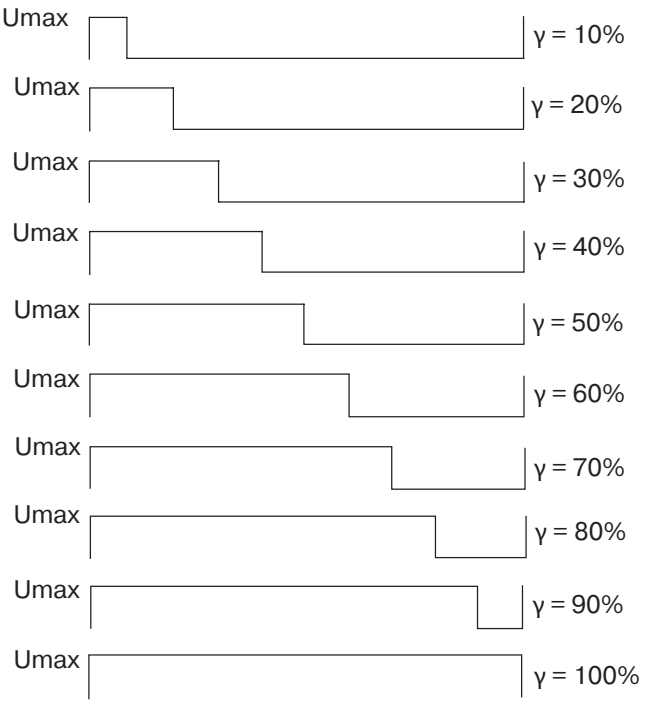
- ***Модули контроллеров последовательных интерфейсов SPI (Serial Peripheral Interface) и SCI (Serial Communication Interface).*** Микроконтроллеры семейства 68HC12 обладают достаточно мощными аппаратными средствами для обмена с другими устройствами в последовательном коде. Модуль контроллера SPI реализует обмен в синхронном режиме, в то время как модуль SCI предназначен для асинхронного последовательного обмена. Синхронный режим обмена в стандарте SPI характеризуется более высокими скоростями обмена по сравнению с стандартным асинхронным протоколом. Однако расстояние между взаимодействующими устройствами ограничено 20...30 см. Интерфейс в стандарте SPI часто используется для подключения к МК дополнительных интерфейсных компонентов, установленных на плате с МК. Например, МК семейства 68HC12 не имеют в своем составе цифро-аналоговых преобразователей (ЦАП). Поэтому система с МК может быть дополнена внешней ИС ЦАП, подключенной к микроконтроллеру с использованием встроенного модуля контроллера SPI. Интерфейс асинхронного обмена SCI часто используется для обмена данными между двумя и более контроллерами, т.к на его основе созданы интерфейсы, сигналы которых могут передаваться на значительные расстояния.
- ***Модуль аналого-цифрового преобразователя ATD (Analog-To-Digital convertion system`).*** Большинство сигналов в системах, которыми управляют микроконтроллеры, имеют аналоговую природу. Например, температура и давление воздуха окружающей среды изменяются плавно, а не скачкообразно. Для работы с аналоговыми сигналами в микропроцессорной системе, эти сигналы должны быть предварительно преобразованы в цифровую форму. В русскоязычной литературе говорят, что эти сигналы должны быть оцифрованы. Для этой цели в составе МК B32 имеется модуль аналого-цифрового преобразователя. Модуль имеет 8 входов для одновременного подключения восьми измеряемых аналоговых сигналов. Однако оцифровка этих сигналов будет производиться последовательно. Измеряемые аналоговые сигналы будут подключаться ко входу одного аналого-цифрового преобразователя (АЦП) посредством встроенного в модуль ATD мультиплексора. Оцифрованный сигнал представляется в 8-и или 10-разрядном прямом коде без знака. Два дополнительных бита кода представления результата позволяют увеличить чувствительность АЦП с 19.53 до 4.88 мВ.
- ***Модуль сторожевого таймера COP (Computer Operating Properly).*** Встраиваемая микропроцессорная система должна обладать высокой надежностью. Одним из путей повышения этой надежности является свойство самовосстановления системы при возникающих некатострофических отказах. Такими отказами являются нарушения исполнения прикладной программы, вызванные помехами, и не сопровождаемые отказом аппаратных средств МК. Для восстановления правильного хода исполнения прикладной программы используется механизм сторожевого таймера. Если программа исполняется без нарушений, то сторожевой таймер регулярно обнуляется (сбрасывается) под

управлением этой программы. Если же ход исполнения программы нарушен, то своевременный сброс может не произойти, и сторожевой таймер переполнится. Переполнение вызовет состояние внутреннего сброса МК, в результате исполнение прикладной программы будет начато заново, т.е. правильный ход исполнения программы будет восстановлен. В главе 5 мы обсудим возможные источники помех и шумов, которые могут вызвать сбой в исполнении программы. В главе 6 поговорим о дополнительных программных мерах, которые следует предпринять для предупреждения подобных отказов.

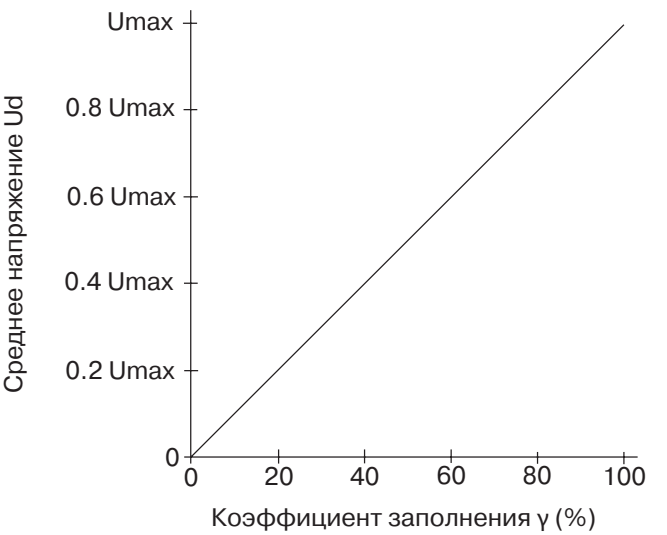
- ***Модуль широтно-импульсного модулятора PWM (Pulse Width Modulation).*** Широтно-импульсная модуляция (ШИМ) – один из способов формирования импульсного сигнала с регулируемыми временными характеристиками. Способ широтно-импульсной модуляции часто используется для регулирования скорости вращения двигателей постоянного тока, а также для управления электрическими двигателями других типов. Для генерации ШИМ-сигнала в МК B32 могут быть использованы аппаратные средства модуля многофункционального таймера TIM. Однако МК B32 оснащен специальным модулем ШИМ. Этот модуль позволяет генерировать четыре независимых импульсных последовательности с 8-разрядным разрешением для задания коэффициента заполнения, или две импульсные последовательности с 16-разрядным заданием коэффициента заполнения. Допускается комбинация этих режимов. Например, ШИМ сигнал используется для управления двигателем рулевого управления в игрушечных радиоупавляемых машинках. Для того, чтобы машинка повернула направо или налево, она должна получить импульсный сигнал, у которого частота следования импульсов постоянная, а длительность импульсов изменяется, как показано на рис. 4.2. Отношение длительности импульса к длительности периода сигнала называется коэффициентом заполнения. Машинка повернет налево, если коэффициент заполнения менее 50% , или направо, если коэффициент заполнения превышает 50%. Модуль PWM микроконтроллера B32 позволяет организовать импульсную последовательность с требуемыми значениями периода следования и коэффициента заполнения при использовании минимального числа команд прикладной программы.
- ***Модуль контроллера последовательного обмена BDLC (Byte Data Link Communication)*** поддерживает коммуникационный протокол SAE J1850, который является действующим стандартом бортовой информационной сети в автомобилях североамериканского производства.
- ***Модуль контроллера CAN интерфейса msCAN12 (Motorola Scalable Controller Area Network)*** содержит в себе набор аппаратных средств для поддержки коммуникационного протокола промышленных сетей в стандарте CAN 2.0 A/B. Этого модуля нет в составе МК модели B32, однако он присутствует во многих других моделях семейства HC12 и HCS12. Пример работы с этим модулем приведен в главе 9.

Для более полного восприятия структуры МК B32 мы дополним приведенные технические характеристики небольшим примером применения.

На рис. 4.3. представлена микропроцессорная система регулирования скорости электрического двигателя на основе МК семейства 68HC12. Желаемая скорость вращения двигателя (число оборотов в секунду – об/с) задается посредством шестнадцатикнопочной клавиатуры. МК 68HC12 осуществляет преобразование вводимых с клавиатуры кодов в ШИМ сигнал для управления двигателем. Информа-



а) Временные диаграммы ШИМ-сигналов
с различными коэффициентами заполнения



б) Регулировочная характеристика

Рис. 4.2. *Широтно-импульсная модуляция*

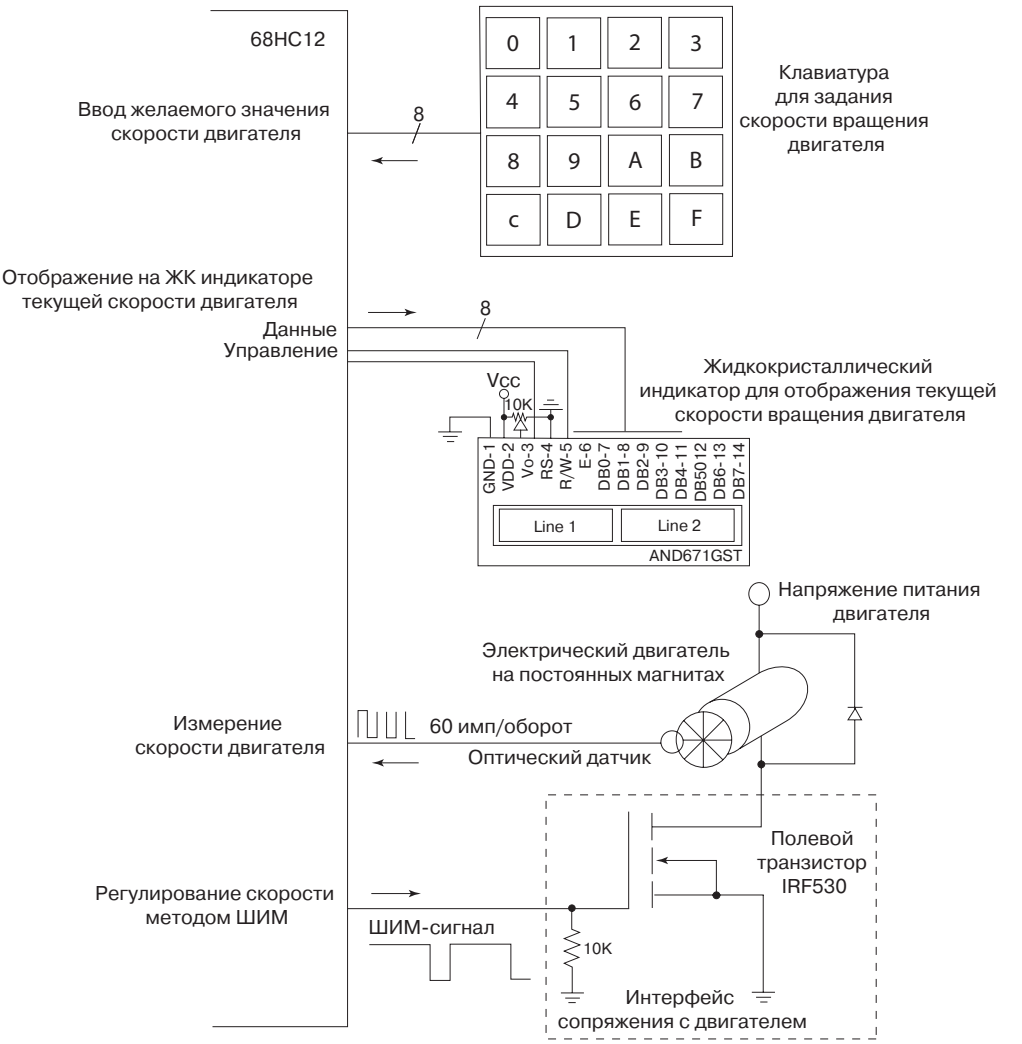


Рис. 4.3. *Система управления электрическим двигателем на основе МК 68HC12*

ция о скорости вращения двигателя снимается с выхода оптического импульсного датчика, установленного на валу двигателя. Измеренное значение скорости используется для вычисления коэффициента заполнения ШИМ сигнала, который управляет подключением силового напряжения к обмотке двигателя. Регулирование коэффициента заполнения ШИМ сигнала позволяет поддерживать скорость вращения двигателя на заданном уровне. Текущая скорость двигателя отображается на жидкокристаллическом (ЖК) индикаторе.

Вопрос: какие подсистемы МК семейства 68HC12 используются для реализации рассмотренной системы регулирования скорости вращения двигателя?

Ответ:

- Параллельный порт задействован для ввода сигналов с клавиатуры.
- Модуль ШИМ используется для генерации широтно-модулированного сигнала управления силовым ключом IRF530. Этот ключ коммутирует силовое напряжение к обмотке двигателя. Если в примере использовать МК семейства 68HC12, который не имеет в своем составе модуля ШИМ, например A4, то этот же сигнал может быть сгенерирован одним из каналов многофункционального таймера в режиме выходного сравнения ОС.
- Выход оптического импульсного датчика подключается на вход счетчика внешних событий PA. Внешний датчик и встроенный в МК счетчик вместе образуют цепь обратной связи системы управления скоростью вращения двигателя.
- Модуль меток реального времени RTI реализует периодические прерывания для считывания накопленного в счетчике внешних событий числа импульсов. Это значение используется для вычисления реальной скорости вращения двигателя.
- Параллельный порт и несколько дополнительных линий другого порта используются для вывода информации на ЖК индикатор.

4.2. Аппаратные средства МК семейства HCS12

Новое семейство МК HCS12 унаследовало архитектуру процессорного ядра и большинства периферийных модулей от своего предшественника, семейства 68HC12. Каковы основные отличия МК нового семейства HCS12 от изученных представителей 68HC12?

- Напряжение питания большинства моделей МК семейства HCS12 равно 5,0 В, что позволяет обеспечить электромагнитную совместимость в автомобильных и общепромышленных применениях.
- Повышенна производительность процессорного ядра. Частота тактирования центрального процессора и межмодульных магистралей МК семейства HCS12 составляет 25 МГц против 8 МГц у HC12;
- Увеличен объем резидентной памяти. Объем встроенного в МК семейства HCS12 ОЗУ достигает 12 Кб, объем Flash ПЗУ – 512 Кб. Кроме того, в составе большинства моделей МК имеется значительная область EEPROM (до 4 Кб) для хранения перепрограммируемых констант пользователя;
- Большое число интегрированных на кристалл разнообразных контроллеров последовательных интерфейсов, т.к. МК семейства предназначены для работы в качестве интеллектуальных узлов распределенных систем управления.

Семейство HCS12 объединяет ряд моделей МК с одинаковым процессорным ядром CPU HCS12. Отдельные представители семейства различаются объемом встроенной памяти и количеством и типом интегрированных на кристалл МК периферийных модулей. Однако каждый МК из семейства HCS12 имеет в своем составе следующие функциональные модули:

- Память трех типов: FLASH память программ, энергонезависимая память EEPROM для хранения изменяемых констант пользователя и статическое ОЗУ для размещения промежуточных переменных прикладной программы управления;
- Многофункциональный 16-разрядный таймер с 8 каланами IC/OC;

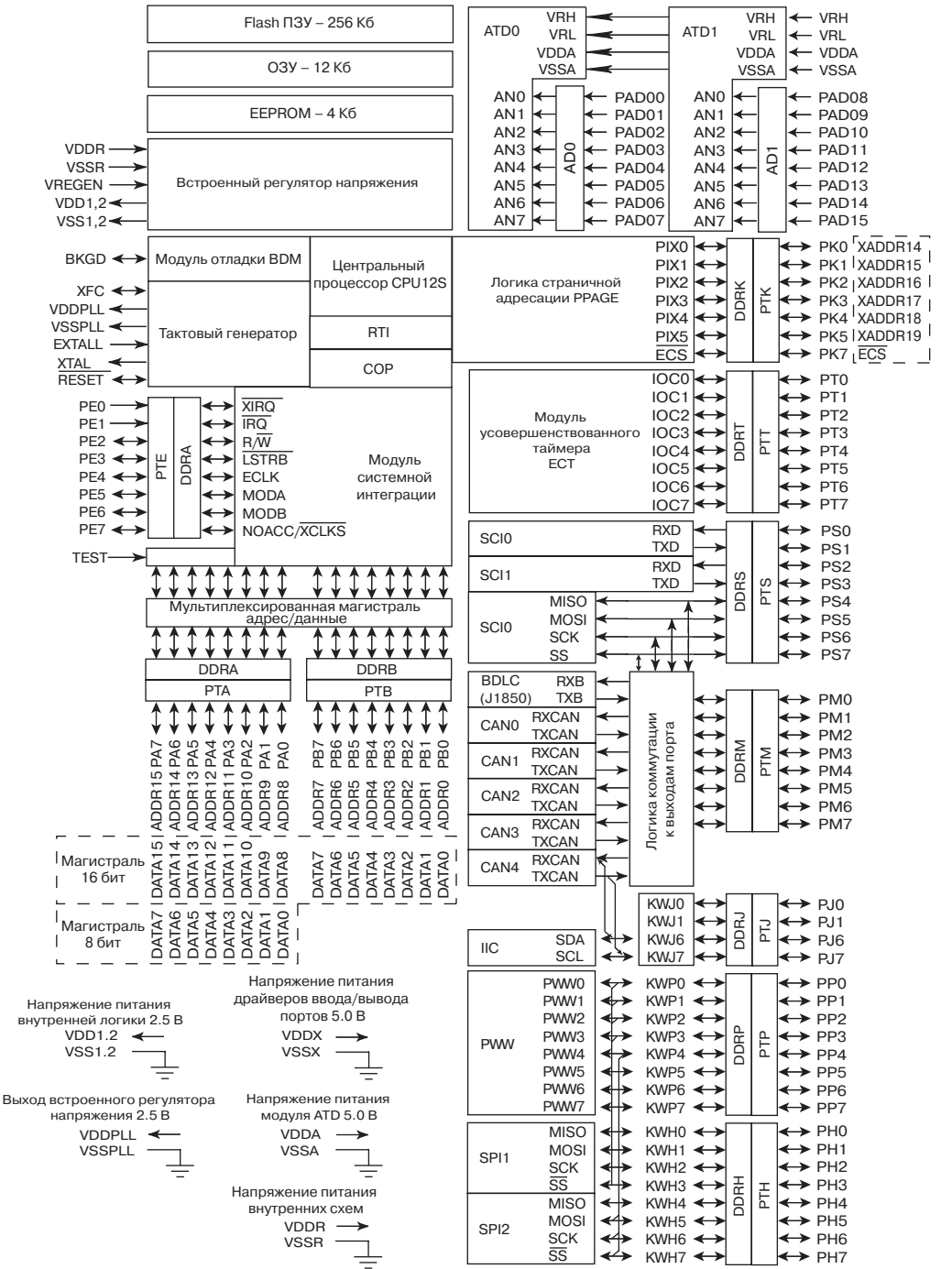


Рис. 4.4. *Структура микроконтроллера MC9S12DP256B*

- Многоканальный аналого-цифровой преобразователь;
- Контроллеры последовательного обмена нескольких стандартов;
- Модуль ШИМ общего назначения, ряд моделей оснащен специализированным модулем ШИМ для управления автономными вентильными преобразователями.

Структурная схема одного из представителей семейства HCS12 – микроконтроллера MC9S12DP256B представлена на рис. 4.4. Обратите внимание, что большая часть модулей этого МК уже рассматривалась Вами в составе МК MC68HC912B32 (рис. 4.1). Однако на кристалле MC9S12DP256B размещены уже два 8-канальных АЦП, добавлены 5 модулей контроллеров CAN и новый модуль PPAGE для аппаратной поддержки режима страничной адресации внешней памяти. Также претерпел изменения модуль таймера ECT, который стал именоваться «усовершенствованным таймером с функцией фиксации» (Enhanced Capture Timer).

Мы надеемся, что читатель получил общее представление о МК семейства 68HC12/HCS12, и следует перейти к подробному изучению их технических особенностей. Далее на протяжении этой главы мы рассмотрим аппаратную реализацию и регистровые модели отдельных модулей в составе МК семейства 68HC12/HCS12. Также будут рассмотрены примеры программного обслуживания каждого модуля. В последующих главах мы объединим полученные навыки программирования периферии МК при создании микропроцессорных устройств различного назначения.

4.3. Режимы работы МК семейства 68HC12/ HCS12

Микроконтроллеры семейства 68HC12/HCS12 функционируют в одном из восьми режимов, которые делят на две группы: рабочие режимы и специальные режимы. Рабочие режимы позволяют создать различную аппаратную реализацию встраиваемого контроллера, в то время как специальные режимы работы предназначены для проведения тестовых испытаний и диагностики МК в процессе производства. Поэтому инженерам по применению микроконтроллеров важно изучить лишь группу рабочих режимов.

BKGD MODB MODA	Режим работы	POTRA	PORTB
000	Специальный однокристальный	POTRA	PORTB
001	Специальный расширенный с 8-разрядной внешней шиной	ADDR [15…8] DATA [7…0]	ADDR [7…0]
010	Специальный периферийный	ADDR, DATA	ADDR, DATA
011	Специальный расширенный с 16-разрядной внешней шиной	ADDR, DATA	ADDR, DATA
100	Нормальный однокристальный	POTRA	PORTB
101	Нормальный расширенный с 8-разрядной внешней шиной	ADDR [15…8] DATA [7…0]	ADDR [7…0]
110	Резервный (периферийный)	–	–
111	Нормальный расширенный с 16-разрядной внешней шиной	ADDR, DATA	ADDR, DATA

Рис. 4.5. *Режимы работы микроконтроллеров семейства 68HC12*

Каждый режим из группы рабочих задает собственное распределение адресного пространства МК и конфигурацию магистралей для подключения внешней памяти. Режим работы МК назначается посредством комбинации логических сигналов на входах BKGD, MODB, MODA микроконтроллера в состоянии начального запуска МК. Состояние начального запуска именуют также состоянием сброса (Reset). Сразу после выхода из состояния сброса МК запоминает кодовую комбинацию на перечисленных входах и переходит в соответствующий режим работы. Полный перечень режимов работы МК 68HC12/HCS12 представлен на рис. 4.5. Там же указаны альтернативные функции линий портов PORT A и PORT B, которые они приобретают в каждом из режимов работы.

4.3.1. Рабочие режимы

В большинстве проектируемых устройств Вы будете использовать МК 68HC12/HCS12 в одном из трех рабочих режимов:

- Однокристальном или автономном режиме;
- Расширенном режиме с 16-разрядной внешней шиной;
- Расширенном режиме с 8-разрядной внешней шиной.

Расширенные режимы работы предоставляют возможность подключения к МК внешней памяти и внешних периферийных ИС с использованием параллельных магистралей адреса и данных. Сигналы магистралей формируются на линиях портов PORTA и PORTB, поэтому использование соответствующих выводов МК в качестве линий ввода/вывода общего назначения в расширенных режимах работы становится невозможным.

Краткое описание рабочих режимов:

- ***Однокристальный режим работы (BKGD: 1, MODB: 0, MODA: 0)*** обеспечивает функционирование МК с использованием только внутренней памяти. Поэтому коды прикладной программы управления и ее переменные должны размещаться только во внутреннем ПЗУ и ОЗУ МК. Порты PORTA и PORTB используются в качестве обычных двунаправленных портов ввода/вывода. Подключение внешних периферийных ИС должно производиться с использованием последовательных интерфейсов или с программной поддержкой временной диаграммы обмена на линиях портов ввода/вывода.
- ***Расширенный режим с 16-разрядной системной шиной (BKGD: 1, MODB: 1, MODA: 1)*** обеспечивает функционирование МК с использованием как внутренней, так и внешней памяти. Для подключения внешней памяти предназначена 16-разрядная мультиплексированная магистраль адрес/данные ADDR15-0/DATA15-0. При этом старший байт мультиплексированной во времени магистрали ADDR15-8/DATA15-8 формируется на линиях PORTA, младший байт ADDR7-0/DATA7-0 – на линиях PORTB.
- ***Расширенный режим с 8-разрядной системной шиной (BKGD: 1, MODB: 0, MODA: 1)*** также реализует работу МК с использованием внутренней и внешней памяти. Но для подключения внешней памяти предназначены 16-разрядная магистраль адреса ADDR15-0 и 8-разрядная магистраль данных DATA7-0. Старший байт магистрали адреса ADDR15-8 выво-

дится на PORTA, младший байт ADDR7-0 – на PORTB. Двунаправленная 8-разрядная магистраль данных DATA7-0 использует линии порта PORTA в мультиплексированном со старшими разрядами магистрали адреса режиме. В обоих расширенных режимах некоторые линии порта PORTE используются для передачи сигналов управления обменом по шине.

4.3.2. Режимы работы отладочной платы M68EVB912B32

В данном параграфе мы рассмотрим режимы отладки, которыми может воспользоваться начинающий разработчик микропроцессорных систем, используя для своих экспериментов специальную плату отладки M68912B32EVB. Установленный на плате МК MC68HC912B32 работает в однокристальном режиме. Возможен перевод МК в расширенный режим работы. Для этого на плате имеются переключатели и разъемы системной шины для подключения внешних элементов. Однако в процессе выполнения учебных задач эта операция вряд ли будет необходима. Объем учебных примеров таков, что программы могут быть размещены во внутренней памяти МК отладочной платы.

Плата отладки M68912B32EVB позволяет реализовать четыре различных режима отладки, каждый из которых назначается посредством определенной конфигурации переключателей режимов:

- ***Режим EVB.*** В этом режиме реализуется работа под управлением резидентного монитора отладки D-Bug12. Аппаратные средства отладочной платы выполнены таким образом, что сразу после включения запустится программа, которая размещается во Flash ПЗУ МК. В заводском исполнении в эту область памяти помещается программа монитора отладки D-Bug12. Если же пользователь вместо программы монитора записал коды программы пользователя, то начнет исполняться последняя. Однако чтобы не потерять коды программы монитора, неопытному пользователю не следует перепрограммировать область Flash ПЗУ. Записанный в памяти МК монитор отладки D-Bug12 содержит в себе простую, но полнофункциональную среду отладки, которая позволяет загрузить в память МК небольшие программы и протестировать их работоспособность.
- ***Режим JUMP-EE.*** В этом режиме исполняется программа, загруженная с адреса $D000 в область памяти EEPROM. Поскольку область памяти EEPROM ограничена 768 байтами, то исполняемый в реальном времени фрагмент программы должен быть очень коротким. Такой режим пригоден для лабораторных испытаний функционирования в реальном времени отдельных фрагментов разрабатываемой прикладной программы.
- ***Режим POD.*** В этом режиме аппаратные средства отладочной платы M68912B32EVB используются в качестве интерфейса между последовательным портом персонального компьютера и отладочным портом модуля отладки BDM другого микроконтроллера, установленного на другой отладочной плате (можно использовать вторую плату M68912B32EVB, как показано в гл.3), или на плате собственной разработки. При этом этот другой МК будет находиться в режиме внутрисхемной отладки под управлением встроенного в МК модуля BDM.

В этом внутрисхемной отладки возможно реализовать все этапы отладки, начиная с записи программы во Flash память МК (программирование) и заканчивая испытаниями прикладной программы в реальном времени на объекте управления. Для взаимодействия с отлаживаемым МК следует использовать один из пакетов программного обеспечения, рассмотренных в гл. 3.

- ***Режим BOOTLOAD.*** Этот режим предназначен для занесения программы во Flash или EEPROM память микроконтроллера MC68HC912B32, установленного на отладочной плате.

4.4. Назначение выводов МК

Так же, как и любая другая современная ИС, один и тот же микроконтроллер семейства 68HC12/HCS12 может быть доступен в нескольких типах корпусов. Мы рассмотрим МК MC68HC912B32 в 80-и выводном корпусе типа QFP (Quart Flat Pack). Назначение выводов корпуса приведено на рис. 4.6. При инсталляции МК в конкретное устройство и при проектировании аппаратных средств, все множество выводов МК может быть разделено на три группы:

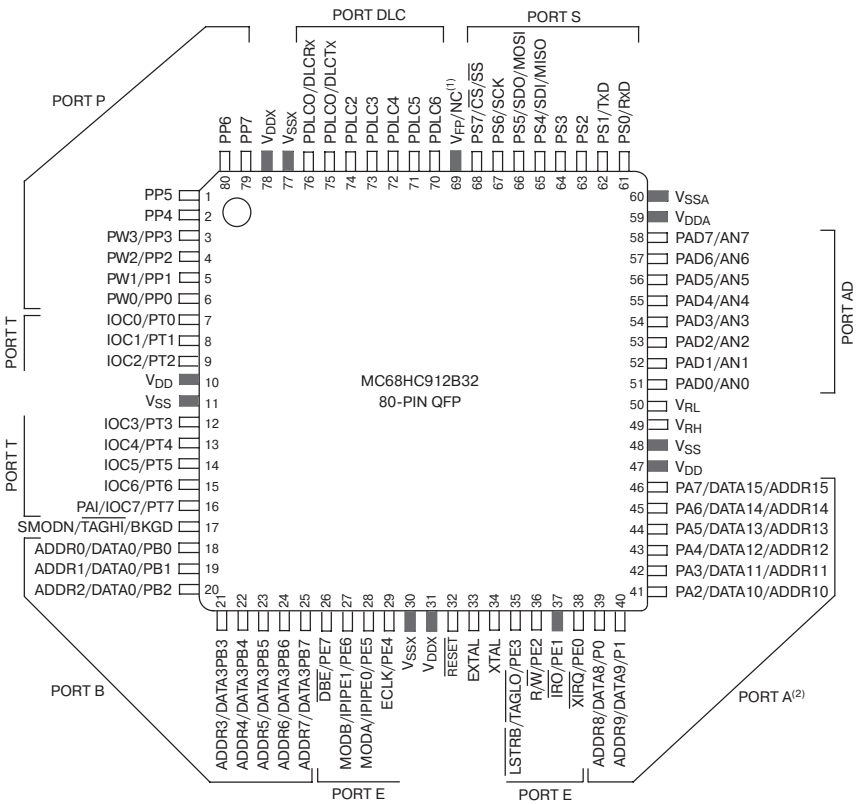


Рис. 4.6. *Цоколевка корпуса микроконтроллера MC68HC912B32*

- Выводы для подключения напряжения питания и опорного напряжения модуля АЦП. Имена выводов этой группы начинаются с буквы V. Например: V_{DD}, V_{SS} и т.д.
- Выводы портов ввода/вывода. Имена выводов этой группы начинаются с буквы P. Например: PA0, PB5 и т.д. Следующие за буквой P символы обозначают имя и номер линии порта: PORTA, линия 0 и PORTB, линия 5.
- Дополнительные выводы. Эти выводы используются для подключения дополнительных логических и нелогических сигналов, которые обеспечивают функционирование аппаратных средств микроконтроллера. Например, выводы XTAL и EXTAL используются для активизации системы тактирования МК.

Многие из выводов МК могут обладать второй, и даже третьей, так называемой альтернативной функцией. О том, как активизировать эти дополнительные функции, мы поговорим во второй части данной главы.

4.5. Регистры специальных функций МК

Представьте себе пульт управления большим технологическим комплексом, состоящим из многих технологических установок. Это может быть завод по производству молочных продуктов или цех раскройки одежды. Каждая технологическая установка этого комплекса, которая выполняет свойственную только ей функцию, может быть запущена в работу или остановлена посредством переключателей на панели управления. Пульт управления позволяет также увидеть текущее состояние каждой технологической установки посредством лампочек на его панели.

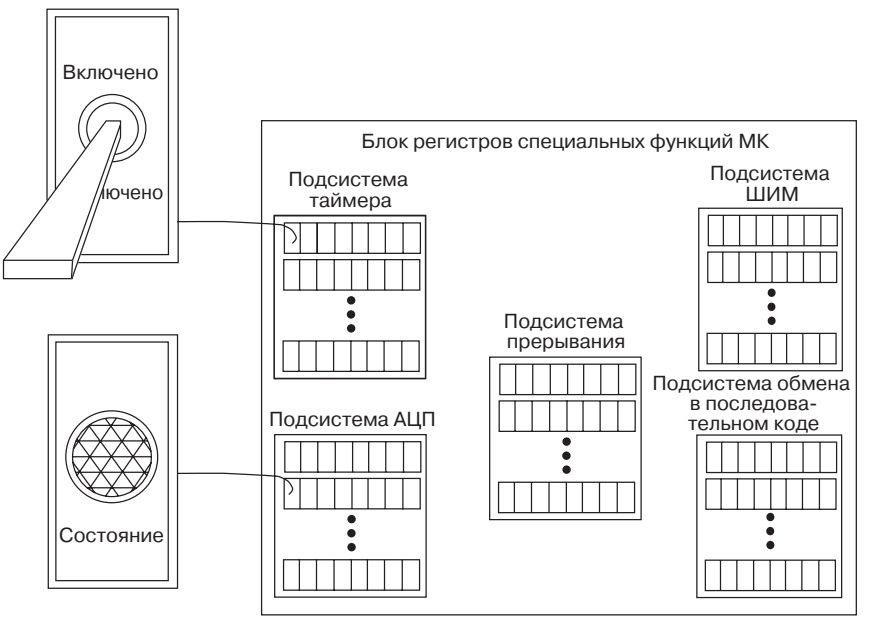


Рис. 4.7. *«Пульт» управления МК семейства HC12*

Блок регистров специальных функций микроконтроллера по реализуемым функциям очень похож на пульт управления. Каждый регистр осуществляет управление или отображает состояние того периферийного модуля МК, к которому он отнесен в техническом описании. Часть битов регистров специальных функций может быть сопоставлена с переключателями: установка бита в 1 разрешает реализацию какой либо функции периферийного модуля, сброс в 0 – прекращает исполнение этой функции. Другая часть битов регистров аналогична индикаторам состояния: установка 1 свидетельствует о завершении выполнения назначенной функции, пребывание в 0 свидетельствует о том, что процесс еще не завершился.

Регистры специальных функций МК семейства 68HC12/HCS12 объединены в блок, который размещается в адресном пространстве памяти МК. Доступ к каждому из 512 регистров блока осуществляется теми же командами центрального процессора, что и к ячейкам памяти. Никакого различия с точки зрения программного обслуживания между регистрами специальных функций и ячейками ОЗУ не существует. Исключение составляют лишь регистры конфигурации и другие, так называемые защищенные регистры, состояние которых невозможно изменить в произвольный момент исполнения прикладной программы.

В момент начального запуска, когда МК находится в состоянии сброса, в каждый регистр специальных функций записывается определенное в техническом описании значение. Начальные состояния регистров специальных функций обычно таковы, что функции соответствующего периферийного модуля отключены. Это очень удобно как с точки зрения потребления энергии микроконтроллером, так и с точки зрения программиста, составляющего прикладную программу управления. Если при запуске большинство периферийных модулей не работает, то МК потребляет минимальную энергию. Далее будут активизированы только те модули, которые потребуются для выполнения управляющего алгоритма. Остальные модули останутся не задействованными, что обеспечит минимизацию энергии потребления в процессе функционировании конечного устройства. При составлении программы управления МК сначала предстает перед разработчиком в минимальной конфигурации, когда большинство периферийных модулей как-бы не существует. Таким МК легко управлять, поэтому ошибок в прикладной программе будет меньше. По мере реализации программы управления разработчик активизирует работу только тех модулей, которые ему потребуются. Поэтому ему не придется думать о программном обслуживании не задействованных при реализации алгоритма периферийных модулей.

Каждый регистр специальных функций имеет свой собственный адрес в карте памяти МК. При программировании на ассемблере изменение значений регистров специальных функций и считывание их состояния может быть произведено командами загрузки и пересылки данных, такими как LDAA, STAA, MOV, а также командами битовых операций BSET, BCLR. Для того чтобы избавить программиста от запоминания физических адресов регистров специальных функций, следует использовать псевдокоманду операции присваивания EQU языка Ассемблер. Эта псевдокоманда назначит символьному имени регистра, которое одинаково для всех моделей МК семейства 68HC12/HCS12, абсолютный адрес. И при записи исходного текста программы можно будет использовать только символьные имена регистров специальных функций. Использование символьных имен также полезно при переносе прикладной программы с одной модели МК на другую. В этом случае следует заменить только абсолютные адреса в псевдокомандах, в то время, как исходный текст программы останется неизменным.

При написании программы на Си с последующим использованием пакета ICC12 компании ImageCraft (см. гл. 3) для этих же целей следует использовать оператор присваивания «=». При программировании на Си установление соответствия между символьными именами регистров специальных функций и их абсолютными адресами в памяти МК наиболее часто производится с использованием директивы препроцессора #define. Для того чтобы освободить программиста от непроизводительного труда, каждая интегрированная среда разработки ПО для МК снабжена файлами заголовков (header file), которые содержат в себе полный набор директив #define для всех регистров специальных функций конкретной модели микроконтроллера. Этот файл включается в исходный текст прикладной программы директивой «include». После этого программист может обращаться к любому регистру специальных функций МК, называя символьное имя регистра. Более того, некоторые заголовочные файлы содержат определения для наиболее часто употребимых битов в составе регистров специальных функций МК, что позволяет устанавливать и сбрасывать эти биты в ходе программы, также используя их символьные имена. Применяя заголовочные файлы при программировании, необходимо помнить, что используемое в тексте программы символьное имя какого-либо регистра должно в точности совпадать с именем, назначенным для него в заголовочном файле. Однако последнее не всегда совпадает с именем, приведенном в техническом описании МК.

Рассмотрим пример записи в заголовочном файле:

```
#define ATDCTL2 *(unsigned char volatile *)  (IO_BASE + 0x62)
```

Разберем, что означает эта запись. Как было отмечено в гл. 3, директива #define используется препроцессором компилятора языка Си для присвоения численных значений символьным константам программы, или, как в нашем случае, определения абсолютного адреса для ячейки памяти или регистра с названным именем. После записи приведенной выше строки препроцессор будет заменять имя ATDCTL2 на указанное в директиве абсолютное значение.

Каково это значение? Оно определено в круглых скобках как сумма символьной константы IO_BASE и шестнадцатеричного числа 0x62. Константа IO_BASE – это начальный адрес блока регистров специальных функций в выбранной модели МК. Для МК B32 начальный адрес блока регистров равен 0x0000. Тогда адрес регистра ATDCTL2 будет равен 0x0062. Использованная форма записи адреса (IO_BASE + 0x62) не случайна, поскольку аппаратные средства 68HC12 позволяют программно изменять начальный адрес блока регистров (см. 4.5.1), сохраняя при этом порядок расположения регистров в памяти.

Каково назначение остальных символов рассматриваемой записи? Переменной ATDCTL2 назначается формат представления данных unsigned char, т.е 8-разрядный без знака. Определение volatile, которое следует за объявление формата, информирует компилятор о том, что изменение значения переменной ATDCTL2 может происходить не только в результате действия программы, но и на аппаратном уровне. Когда переменная сопровождается определением volatile, компилятор не использует по отношению к ней оптимизацию. Знак * после определения volatile указывает на то, что в следующих скобках заключено значение адреса переменной, а не значение этой переменной.

Пример.

Регистр ATDCTL2, расположенный в памяти МК по адресу 0x0062, управляет работой модуля аналого-цифрового преобразователя ATD. Бит 7 этого регистра разрешает (1) или запрещает (0) работу модуля. Этот бит, обозначенный в техническом описании как ATPU (ATD Powerup Bit), устанавливается в 0 в состоянии сброса МК. Для включения модуля ATD в работу необходимо программно установить этот бит в 1. На языке Ассемблера это действие может быть произведено следующим фрагментом программы:

```
;-------------------------------------------------------------------
;MAIN PROGRAMM
;-------------------------------------------------------------------

ATDCTL2       EQU    $0062         ;определить адрес регистра в МК
ATD_INI       EQU    $80           ;определить значение слова
                                   ;инициализации для включения АЦП

              LDAA   #ATD_INI      ;загрузить слово инициализации в ACC
              STAA   ATDCTL2       ;записать значение инициализации в
                                   ;регистр управления АЦП
;-------------------------------------------------------------------
```

Первые две строки программы содержат директивы присвоения EQU языка Ассемблер, которые информируют программу о том, что вместо имени ATDCTL2 в кодах программы следует использовать значение $0062, а вместо ATD_INI – $80 или 10000000 в двоичном коде. Команда LDAA использует непосредственную адресацию, в результате ее выполнения число $80 загрузится в аккумулятор ACC. Команда STAA с прямой адресацией пересылает содержимое ACC в ячейку с адресом ATDCTL2. В результате, код в регистре управления ATDCTL2 будет равен 10000000, т.е. установленный в 1 бит 7 вызовет включение модуля АЦП.

На языке Си аналогичное действие выполняет следующий программный фрагмент:

```
/*-------------------------------------------------------------------*/
/*MAIN PROGRAMM
/*-------------------------------------------------------------------*/

#include<912b32.h>

void main(void)
{
unsigned char ATD_INI = 0x80;
ATDCTL2 = ATD_INI;
}
/*-------------------------------------------------------------------*/
```

Как видите, Вам не пришлось указывать в программе физический адрес регистра ATDCTL2, т.к. он уже определен в заголовочном файле 912b32.h.

4.5.1. Виртуальный адрес блока регистров

Все регистры специальных функций МК семейства 68HC12/HCS12 объединены в блок, который занимает 512 ячеек памяти в адресном пространстве МК. Регистры

располагаются в памяти, начиная с некоторого начального адреса, который принято в программном коде обозначать как IO_BASE. Адреса регистров следуют последовательно друг за другом, поэтому адрес каждого регистра можно представить, как (IO_BASE + код смещения). Численное значение кода смещения каждого регистра определено в карте памяти конкретной модели МК в составе семейства 68HC12/HCS12 и может быть получено из технического описания этого МК. Так в примере предыдущего параграфа код смещения регистра ATDCTL2 в составе МК B32 был равен $0062.

После выхода МК из состояния сброса блок регистров специальных функций располагается в адресном пространстве МК, начиная с адреса IO_BASE = $0000. Такое расположение блока наиболее часто сохраняется при использовании МК семейства 68HC12/HCS12 в однокристальном режиме работы, поскольку для обращения к регистрам могут быть использованы команды с прямой адресацией, которые имеют двухбайтовый формат и экономно расходуют память программ МК. Однако аппаратные средства МК семейства 68HC12/HCS12 позволяют переопределить значение базового адреса блока регистров IO_BASE = $0000 на любое другое кратное 2 Кб в пределах адресного пространства в 64Кб. Для этого достаточно записать соответствующее значение в один из регистров специальных функций МК с именем INITRG. После изменения базового адреса блок регистров будет «располагаться» в первых 512 ячейках выбранной области памяти объемом 2 Кб. Причем порядок расположения регистров, а значит и значения кодов смещения, в пределах этой области памяти останутся неизменными. Так если блок регистров назначен в область памяти с базовым адресом $1000, то регистр ATDCTL2 будет иметь новый виртуальный (т.е. кажущийся) адрес, равный $1000 + $0062. Заметим, что при использовании виртуальной адресации регистров программисту придется только переопределить базовый адрес IO_BASE , в то время как указанные в заголовочном файле коды смещения останутся прежними.

Зачем нужна система виртуальной адресации? Дело в том, что при работе МК в одном из расширенных режимов расположение блока регистров с адреса $0000 может вызвать неудобства при проектировании схемы подключения внешней памяти. А изменение базового адреса значительно упростит эту схемотехнику.

4.6. Порты ввода/вывода

Все МК семейства 68HC12 имеют некоторое количество линий ввода/вывода данных. Линии объединены в 8-разрядные параллельные порты данных: Port A, Port B, Port E, PORT AD и т.д.

За редким исключением, все линии ввода/вывода - двунаправленные. Направление передачи линий ввода/вывода настраивается программно путем записи управляющего слова в регистр направления передачи соответствующего порта. Возможно изменение направления передачи в ходе выполнения программы посредством перепрограммирования этих регистров. Сигнал сброса устанавливает все двунаправленные линии в режим ввода. Следует особо подчеркнуть, что направление передачи каждой линии может быть выбрано разработчиком произвольно, независимо от других линий, принадлежащих к одному и тому же порту ввода/вывода. Исключение составляют лишь линии однонаправленной передачи, которые изначально специализированы на ввод или на вывод (см. 4.6.1).

Часть линий ввода/вывода имеют так называемую альтернативную функцию, т.е. обеспечивают связь встроенных периферийных модулей МК с «внешним миром». Так линии порта PORT AD используются для подключения к встроенному АЦП измеряемых напряжений, линии порта PORT S служат входами и выходами контроллеров последовательного обмена. Если соответствующий периферийный модуль МК не используется, то его выводы можно задействовать как обычные линии ввода/вывода.

Если линии порта двунаправленные, то для его обслуживания такого порта предусмотрены два типа регистров:

PORTx – регистр данных порта x, где x – имя порта ввода/вывода;

DDRx – регистр направления передачи порта x.

Например, порт PORT A обслуживается регистрами PORTA и DDRA, а порт PORT B – регистрами PORTB и DDRB.

Если порт имеет схемотехнику с программно подключаемым «подтягивающим» резистром (R pull-up), то для обслуживания такого порта предусмотрен дополнительный регистр входного сопротивления порта.

Ниже приведен фрагмент текста программы, которая конфигурирует PORT T для вывода данных, а затем записывает в порт число S62. Для того, чтобы все линии порта PORT T стали линиями вывода, необходимо записать в регистр направления передачи DDRT код $FF.

```
/*-------------------------------------------------------------------*/
/* MAIN PROGRAM:
/*-------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>

void main{void)
{
unsigned char DDRT_INI = 0xFF

DDRT = DDRT_INI;          //установить порт PORTT на вывод
PORTT = 0x62
}
/*-------------------------------------------------------------------*/
```

4.6.1. Спецификация портов ввода/вывода

Подсистема параллельного ввода/вывода МК B32 состоит из 8 портов, причем линии многих портов обладают альтернативной функцией. Мы рассмотрим эти порты в порядке их расположения на рис. 4.1. по часовой стрелке.

- PORT A. В однокристальном режиме работы – 8-разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRA. В расширенном режиме работы на линиях порта формируются сигналы старшего байта мультиплексированной магистрали адрес/данные ADDR15-8/DATA15-8. В расширенном режиме с 8-разрядной шиной линии порта представляют собой мультиплексированную магистраль ADDR15-8/DATA7-0.
- PORT B. В однокристальном режиме работы – 8-разрядный порт вво-

да/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRB. В расширенном режиме работы на линиях порта формируются сигналы младшего байта мультиплексированной магистрали адрес/данные ADDR7-0/DATA7-0. В расширенном режиме с 8-разрядной шиной линии порта представляют собой немультиплексированную магистраль ADDR7-0.

- PORT E. 8-разрядный порт ввода/вывода общего назначения. Две линии порта PE0 и PE1 – однонаправленные и работают только на ввод. Остальные линии порта – двунаправленные, направление передачи линий PE2-PE7 определяется соответствующими битами регистра DDRE. Все линии порта имеют альтернативную функцию. Линии PE1 и PE0 могут использоваться как входы внешних прерываний $\overline{IRQ}$ и $\overline{XIRQ}$. Остальные линии служат для задания режимов работы МК (MOD A и MOD B) и в качестве сигналов управления внешней шиной при работе МК в расширенно режиме.
- PORT AD. Однонаправленный 8-разрядный порт ввода. Все линии имеют альтернативную функцию. Если работа модуля аналого-цифрового преобразователя ATD разрешена, то линии порта используются для подключения измеряемых аналоговых сигналов.
- PORT T. Двунаправленный 8-разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRT. Альтернативная функция линий порта PORT T – обслуживание модуля таймера. Если работа таймера разрешена, то линии используются в качестве входов входного захвата IC или выходов выходного сравнения OC.
- PORT S. Двунаправленный 8-разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRS. Альтернативная функция линий порта PORT S – обслуживание модулей последовательного обмена SCI и SPI.
- PORT P. Двунаправленный 8-разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRP. Четыре линии порта PORT P могут использоваться в качестве выходов модуля генератора ШИМ-сигнала (модульPWM), если работа последнего программно разрешена.
- PORT DLC. Двунаправленный 8-разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRDLC. Альтернативная функция двух линий порта PORT DLC – обслуживание модуля последовательного обмена в стандарте BDLC.

Регистры управления портами

В МК семейства 68HC12/HCS12 каждый двунаправленный порт ввода/вывода общего назначения обслуживается двумя регистрами специальных функций. Это регистр данных порта и регистр направления передачи DDRx (вместо буквы «x» следует подставить буквенное обозначение порта). Если линии порта общего назначения настроены на ввод, то операция чтения регистра данных возвращает состояние выводов корпуса МК, с которыми связан порт. Если порт настроен на вывод, то операция записи в регистр данных устанавливает на вы-

водах корпуса МК, связанных с портом, соответствующие логические уровни. Регистр DDRx определяет направления передачи каждой линии порта независимо от других линий этого же порта. Если какой либо бит регистра DDRx равен 0, то соответствующая линия настраивается на ввод, если 1 – то на вывод. Возможны решения, при которых часть линий одного и того же порта настроена на ввод, а часть на вывод. Например, при значении DDRx=10110010 линии D6, D3, D2 и D0 развернуты на ввод, а линии D7, D5, D4 и D1 – на вывод. В состоянии сброса МК все биты регистров направления передачи DDRx сбрасываются, поэтому сразу после включения питания все линии портов МК конфигурированы как входы с высоким входным сопротивлением.

Часть портов ввода/вывода обслуживается дополнительными регистрами управления:

- PUCR (Pull-Up Control Register) – регистр разрешения схемотехники подтягивающих резисторов. Формат регистра представлен на рис. 4.8. Если соответствующие биты регистра установлены, то в портах PORT A, PORT B и Port E при конфигурировании какой либо линии порта на ввод автоматически подключается встроенный подтягивающий к напряжению питания резистор. Если же эта линия настраивается на вывод, то встроенный резистор автоматически отключается. В расширенных режимах работы МК, когда названные порты используются для формирования сигналов внешних магистралей адреса, данных и управления, встроенные резисторы также автоматически отключаются.
- RDRIV (Reduced DRIVe Register) – регистр выбора режима работы выходных каскадов с пониженными выходными токами. Формат регистра также представлен на рис. 4.8. Как следует из рис. 4.8, в МК B32 этот режим также

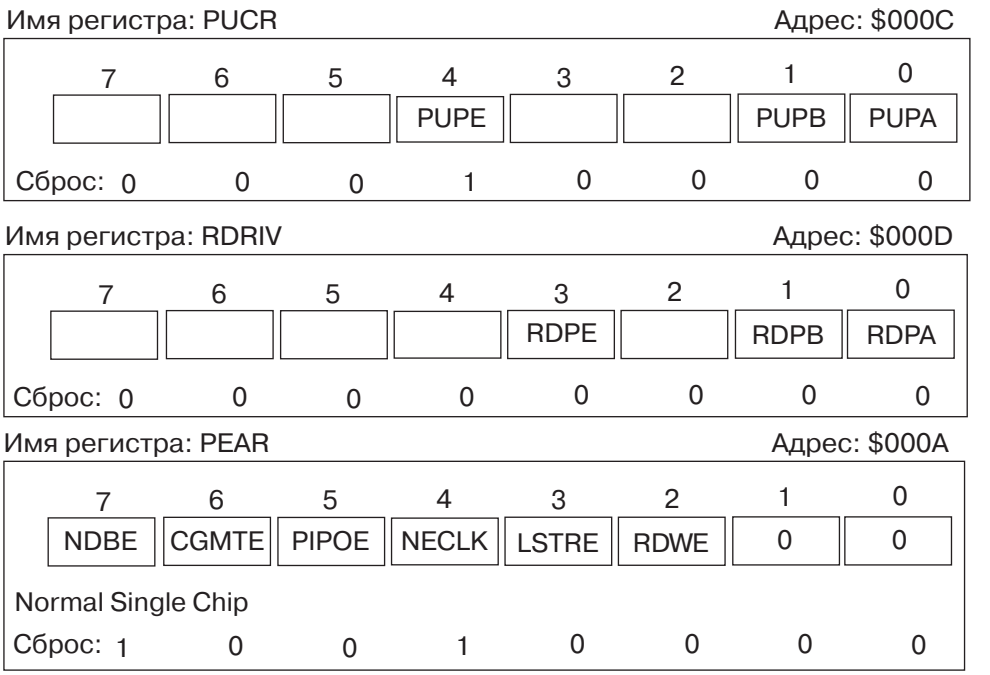


Рис. 4.8. *Формат регистров PUCR, RDRIV, PEAR*

доступен только для портов PORT A, PORT B и PORT E. В МК иных моделей этой функцией могут обладать также и другие порты. Если функция для порта разрешена установкой разряда RDPx в 1, то при конфигурировании какой-либо линии порта на вывод ее выходной ток снижается с 0,6 мА до 0,3 мА.

- PEAR (Port E Assignment Register) – регистр выбора назначения линий порта Port E. Отдельные биты этого регистра (рис. 4.8) позволяют назначить линии альтернативную функцию или функцию линии ввода/вывода общего назначегния.

Вопросы для самопроверки

1. Сколько портов ввода/вывода в МК B32?
   Ответ: МК B32 имеет в своем составе восемь 8-разрядных портов (A, B, E, AD, T, S, P, DLC).
2. Какие альтернативные функции реализуют порты AD, T, S, P, DLC?
   Ответ: Линии порта AD служат аналоговыми входами встроенного АЦП, порт T используется модулей таймера, на выходах порта P формируются ШИМ-сигналы, порт S обслуживает контроллера последовательного ввода/вывода SCI и SPI.
3. Каково назначение регистра направления передачи порта?
   Ответ: Каждый разряд этого порта определяет направление передачи соответствующей линии порта.
4. В какое состояние устанавливается регистр направления передачи во время сброса МК? Ответ: Сбрасывается, т.е. устанавливается в 0. При этом все линии портов конфигурируются на ввод.

Пример применения

В этом примере мы подключили в порту PORTA группу зеленых и красных светодиодов. Схема подключения показана на рис. 4.9. Мы обсудим эту схему подробно в гл.5. А сейчас лишь договоримся, что если на линии порта установлена 1, то будет гореть зеленый светодиод, если логический 0 – то красный светодиод. А если линия порта переведена в состояние ввода, т.е. она представляется для цепи светодиодов нагрузкой с высоким входным сопротивлением, то оба светодиода окажутся погашенными. На рис. 4.10 приведена блок-схема алгоритма программы, которая зажигает на 30 мс зеленым цветом светодиоды с четными номерами и одновременно красным цветом светодиоды с нечетными номерами. Следующие 30 мс светодиоды «меняются цветами», далее этот процесс продолжается до бесконечности. Ниже приведен текст программы на языке Си, который реализует этот алгоритм.

```
/*-------------------------------------------------------------------*/
/* MAIN PROGRAM: Эта программа «зажигает» на выходах порта PORTA   */
/*30 мс зеленым цветом горят светодиоды на выходах порта с четными   */
/*номерами, красным цветом - светодиоды на выходах порта с нечетными  */
/* номерами. Следующие 30 мс на месте зеленых горят красные, и наоборот */
/*-------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>

/*используемые функции*/
void delay_100us(void)
```

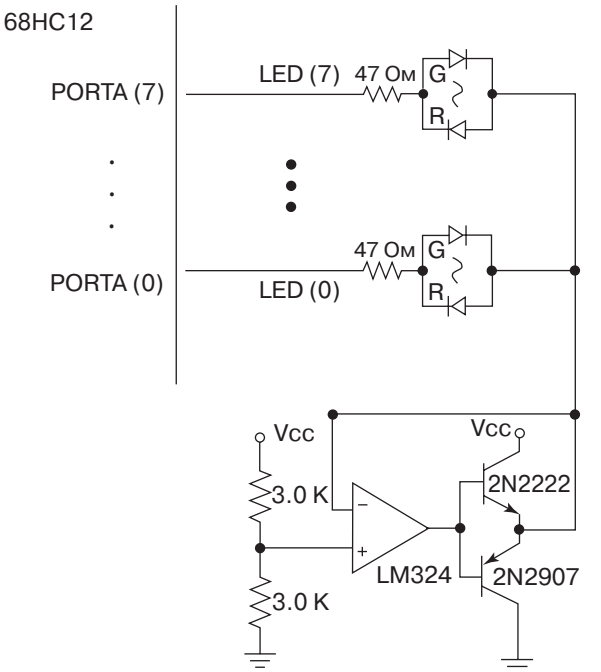


Рис. 4.9. *Схема отображения состояния порта PORTA*

Схема содержит 8 идентичных светодиодных индикаторов для логических выходов с тремя состояниями. Индикатор каждого разряда состоит из двух светодиодов: зеленого и красного. Если на выходе порта PORTA[i] логическая 1,то светится зеленый светодиод, если логический 0 – то красный. Если линия порта PORTA[i] настроена на ввод, то ни один из светодиодов этого разряда порта не светится.

```
void delay_30ms(void)

void main{void)
{
DDRA = 0xFF          //установить порт PORTA на вывод

while(1)
       {
       PORTA = 0x55;
                     delay_30ms();
       PORTA = 0xAA;
                     delay_30ms();
       }
}
/*-------------------------------------------------------------------*/
/* Функция delay_30ms формирует задержку в 30мс мкс, частота тактирования*/
/* межмодульных магистралей МК составляет 8 МГц                      */
/*-------------------------------------------------------------------*/
void delay_30ms(void)
{
int i;
```

Рис. 4.10. *Блок-схема алгоритма управления светодиодами*

```
for (i=0; i<=299; i++)
    delay_100us();
}

/*---------------------------------------------------------------------*/
/* Функция delay_100us формирует задержку в 100 мкс, частота тактирования */
/* межмодульных магистралей МК составляет 8 МГц                        */
/*---------------------------------------------------------------------*/
void delay_100us(void)
{
int j;
for (j=0; j<50; j++)
      {
      asm(«nop»\n) ;
      }
}
/*---------------------------------------------------------------------*/
```

Обратите внимание, что функция задержки на 30 мс использует вложенную функцию задержки на 100 мкс. В приведенном тексте программы для формиро-

вания задержки на 100 мкс используются 50 циклов повторения операторов функции delay_100us. Выбор числа повторений производился из предположения, что данная программа будет исполняться микроконтроллером, частота внутренней шины которого составляет 8 МГц. По результатам рассмотрения файла в формате *.lst было установлено, что команды ассемблера, соответствующие одному повторению цикла функции delay_100us , реализуются за 16 машинных циклов. При частоте шины в 8 МГц для формирования временного интервала в 100 мкс потребуется 800 машинных циклов. Поэтому число повторений цикла функции delay_100us должно составлять 800/16 = 50. Если бы эта программа исполнялась бы микроконтроллером DP256, частота внутренней шины которого составляет 25 МГц, то число циклов функции delay_100us должно было бы быть увеличено до 156.

4.7. Подсистема памяти МК B32

Подсистема памяти МК семейства 68HC12/HCS12 включает четыре различных модуля памяти: энергонезависимая Flash память программ, энергонезависимая EEPROM память данных, статическое ОЗУ и блок регистров специальных функций для управления режимами работы периферийных модулей. Расположение различных модулей памяти в адресном пространстве МК принято отражать на так называемой карте памяти. Карта памяти МК B32 представлена на рис. 4.11. Указанные на ней адреса будут действительны при выходе МК из состояния сброса. В ходе исполнения прикладной программы адресное пространство для каждого модуля памяти может быть изменено (см. 4.5.1.). Тогда для обращения к ячейкам памяти будут использоваться не указанные на рис. 4.11 физические адреса, а измененные виртуальные адреса.

Микроконтроллер B32 предназначен для использования преимущественно в однокристальном режиме работы. Он содержит в себе 32Кб ПЗУ программ, 768 байт памяти типа EEPROM, 1Кб статического ОЗУ и 512 регистров управления.

Строго говоря, аббревиатура EEPROM (Electrically Erasable Programmable ROM) обозначает энергонезависимую память с электрическим программированием и электрическим стиранием. Поэтому и резидентная память программ, выполненная на основе технологии FLASH, и энергонезависимая память данных должны быть характеризованы как EEPROM. Однако энергонезависимая память программ и энергонезависимая память данных отличаются по своим свойствам не только на уровне технологии изготовления, но и на уровне разработчика встраиваемых систем. Память типа Flash допускает выполнение операции стирания только над некоторым множеством ячеек, что неудобно при необходимости замены только одного байта информации. Энергонезависимая память данных позволяет стереть и потом запрограммировать один байт информации. Однако ячейки памяти с подобным свойством занимают значительную площадь полупроводникового кристалла МК, поэтому на их основе не может быть выполнена память программ большого объема. Во избежание путаницы у российских разработчиков принято использовать аббревиатуру EEPROM только для памяти с побайтным стиранием и побайтным программированием. А память со стиранием блоками обозначают как Flash, хотя и эта память по своим свойствам относится

$0000 $01FF	Регистры специальных функций
$0800 $0BFF	ОЗУ 1 Кб ● Код/данные пользователя ($0800-$09FF) ● Резервная область для D-Bug12 ($0A00-$0BFF)
$0D00 $0FFF	EEPROM 768 байт ● Код/данные пользователя
$8000 $FFFF	FLASH ПЗУ программ 32 Кб ● код D-Bug12 ($8000–$F67F) ● Область, доступная пользователю ($F6C0–$F6FF) ● Настройка функций D-Bug12 ($F680–$F6BF) ● Код запуска D-Bug12 ($F700–$F77F) ● Вектора прерывания ($F780–$F7FF) ● Расширение загрузчика ($F800–$FBFF) ● Загрузчик EEPROM ($FC00–$FFBF) ● Вектора сброса и прерывания ($FFC0–$FFFF)

Рис. 4.11. *Карта памяти МК B32 в составе отладочной платы MC68HC912B32EVB*

к EEPROM.

Используя МК от Freescale Semiconductor, в частности семейство 68HC12/HCS12 следует знать, что гарантированное число циклов стирания/программирования для МК HCS12 составляет 10000, а для МК 68HC12 – всего 100. Именно поэтому во многих отладочных средствах на основе МК 68HC12 рекомендуется промежуточные версии программы записывать и исполнять из ОЗУ. Поскольку резидентное ОЗУ у микроконтроллеров обладает недостаточным объемом для размещения программы, то многие отладочные платформы используют расширенный режим работы МК с подключением внешнего ОЗУ. В МК семейства HCS12 для целей отладки обычно используется внутреннее Flash ПЗУ программ, т.к. 10000 циклов перезаписи обычно остаточно для внесения всех исправлений в процессе отладки.

Пример применения

В процессе эксплуатации память типа EEPROM часто используют для создания счетчиков аварийных ситуаций на объекте. Возможные аварии предварительно классифицируются, в системе устанавливаются датчики, которые позволяют микроконтроллеру отнести возникшую аварийную ситуацию к тому или иному типу. Если тип аварии диагностирован, то МК увеличивает соответствующий счетчик и запоминает его новое состояние в энергонезависимой памяти данных типа EEPROM. Такое решение позволяет сохранить информацию об авариях даже

при отключении системы питания.

4.7.1. Карта памяти МК B32

Карта памяти МК определяет, по каким адресам в конкретной модели МК расположены блоки памяти. Процессорное ядро HC12 позволяет линейно адресовать 64К слов памяти. Поскольку МК 68HC12 используют однобайтовые ячейки памяти, то получается, что при линейной адресации в МК семейства 68HC12 адресуют 64 Кб памяти. Определение «линейная адресация» означает, что в любой момент времени без применения дополнительных команд МК может обратиться к ячейке памяти с любым адресом из диапазона $0000...$FFFF.

Карта памяти МК B32 привелдена на рис. 4.11. Резидентная память (т.е. память, расположенная на кристалле МК) включает четыре блока памяти: 512 байт регистров специальных функций для управления периферийными модулями, 1 Кб оперативного запоминающего устройства для хранения прормежуточных результатов вычислений, 768 байт энергонезависимой памяти типа EEPROM с побайтным стиранием и побайстным программирования для хранения уставок программы пользователя, 32 Кб энергонезависимой Flash памяти для размещения прикладной программы пользователя.

На рис. 4.11 карта памяти учитывает особенности размещения резидентной программы отладки D-Bug12 при работе МК в составе платы отладки MC68HC912B32EVB:

- $8000...$F600 – код программы отладчика D-Bug12;
- $F680...$F6BF – область пользователя;
- $F6C0...$F6FF – область D-Bug12;
- $F700...$F77F – код запуска D-Bug12;
- $F780...$F7FF – таблица векторов для режима отладки;
- $F800...$FBFF – зарезервированная разработчиком область;
- $FC00...$FFBF – код программы загрузчика в EEPROM;
- $FFC0...$FFFF – вектора сброса и прерывания.

Из представленного распределения адресного пространства МК в составе платы отладки видно, что код программы отладчика занимает практически всю область Flash ПЗУ, которая в реальных проектах предназначается для прикладной программы управления. А где же предполагается размещать отлаживаемую программу? Ответ на этот вопрос Вы найдете в параграфе 4.3.1.

4.7.2. Изменение адресов в карте памяти МК

Внимательно проанализировав распределение адресного пространства в карте памяти МК, можно заметить, что часть доступного адресного пространства не используется резидентной памятью микроконтроллера. Так в МК B32 (рис. 4.11) в диапазоне адресов $01FF...$0800, $0BFF...$0D00 и $0FFF...$8000 память отсутствует. Именно это незанятое адресное пространство может быть использовано для подключения внешней памяти в расширенных режимах работы МК. В процессе подключения внешней памяти может оказаться, что отдельные блоки внутренней памяти МК желательно «переместить» в пределах адресного пространства МК. Тогда схемотехника подключения внешней памяти упростится. Для назначения новых, виртуальных адресов блоков резидентной памяти МК предназначены три регистра специальных функций:

- INITRG – регистр базового адреса блока регистров специальных функций;

- INITRM – регистр базового адреса блока ОЗУ;
- INITEE – регистр базового адреса блока EEPROM.

В состоянии сброса МК эти регистры указывают на реальные, физические адреса перечисленных блоков памяти. Назначение виртуальных адресов блоков обычно происходит на начальном этапе выполнения программы в секции инициализации.

4.8. Подсистема памяти МК DP256

Карта памяти для МК DP256 приведена на рис. 4.12. Расположение различных модулей памяти в адресном пространстве МК чрезвычайно похоже на рассмотренный ранее МК B32. Основное отличие подсистемы памяти DP256 от B32 – это наличие системы страничной адресации, которая позволяет обращаться к более чем 64 Кб памяти с использованием 16-разрядной магистрали адреса. Объем резидентной памяти МК DP256 составляет 256 Кб. Модули резидентного ОЗУ, EEPROM и регистры специальных функций доступны, как и ранее, с использованием обычной адресации в пределах 64 Кб (см. рис. 4.12). Основная часть ПЗУ программ разделена на страницы по 16 Кб, причем каждая из страниц может отображаться на одном и том же адресном пространстве $8000-$BFFF. Выбор конкретной страницы осуществляется занесением под управле-

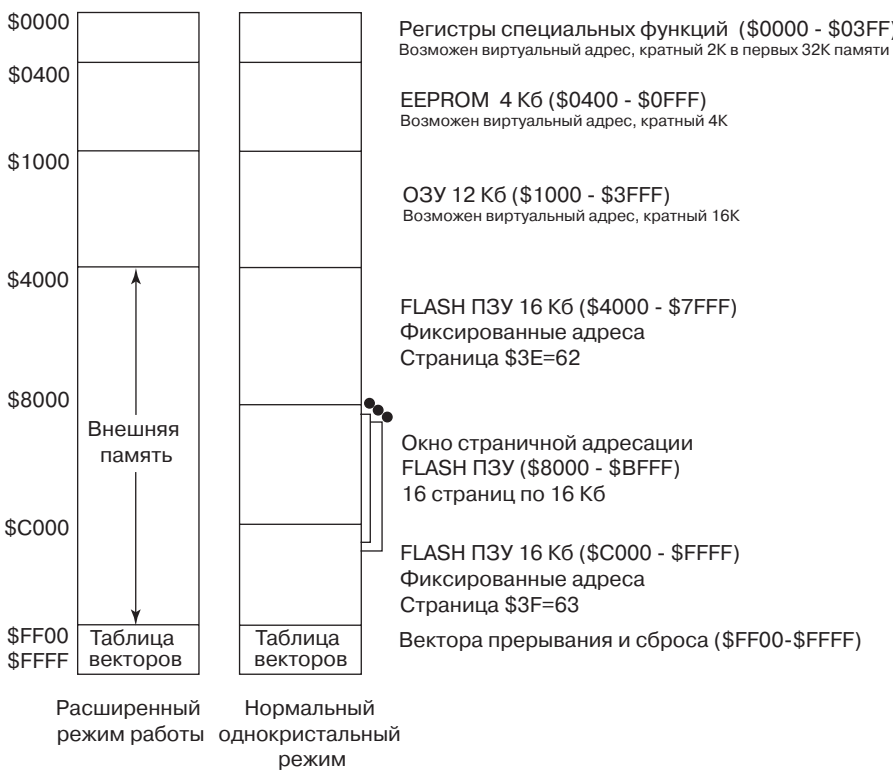


Рис. 4.12. *Карта памяти МК семейства HCS12 DP256*

нием программы ее кода в специальный регистр. Поэтому частое переключение между страницами программной памяти замедляет исполнение программы.

Вопросы для самопроверки

1. Каков объем Flash памяти программ МК B32?
   Ответ: 32 Кб.
2. Как занести программу пользователя во Flash память МК B32, установленного на отладочной плате M68EVB912B32?
   Ответ: Для этого может быть использована программа загрузчика, которая поместит программу пользователя в область EEPROM, начиная с адреса $D000, или в область Flash ПЗУ начиная с адреса $8000. Однако в последнем случае программа монитора отладки D-Bug12 будет потеряна.
3. Сколько раз можно перепрограммировать резидентное Flash ПЗУ программ МК?
   Ответ: 100 раз в МК семейства 68HC12 и 10000 раз в МК семейства HCS12.
4. Сколько раз можно перезаписать данные в энергонезависимой памяти данных типа EEPROM?
   Ответ: гарантированное число циклов стирания/программирования резидентной энергонезависимой памяти данных равно 10000. На практике это число значительно больше.

4.9. Состояния сброса и прерывания МК

В процессе исполнения прикладной программы МК реализует монотонную многократно повторяющуюся последовательность действий:

- Выборку кода команды из памяти программ в регистр команды центрального процессора;
- Дешифрацию кода команды;
- Выборку из памяти следующих байтов команды;
- Исполнение команды;
- Сохранение в памяти результатов исполнения команды.

Если исполняется линейная последовательность команд, то содержимое счетчика PC центрального процессора постоянно увеличивается на 1, обеспечивая выборку из памяти следующих команд прикладной программы. Линейная последовательность исполняемых команд может быть изменена под управлением самой программы, например инструкциями «jmp» или «branch». При этом в счетчик команд под управлением программы будет записано новое число, и начнется исполнение следующего линейного фрагмента программы из другого сегмента памяти программ. Несмотря на явные различия механизмов формирования следующего за исполнением текущей операции значения счетчика команд PC, в обоих рассмотренных случаях это следующее значение PC определяется ходом вычислительного процесса и предсказывается программистом в ходе написания прикладной программы.

В противоположность только что рассмотренному полностью предсказуемому потоку событий, который формируется самой микропроцессорной системой, существует еще поток внешних событий, очередность и моменты возникновения которых не синхронизированы с исполнением центральным процессором тех или иных команд прикладной программы. Однако МК должен реагировать на эти со-

бытия, для чего необходимо изменить последовательность исполнения операторов программы в произвольный, непредсказуемый с точки зрения устройства управления центральным процессором момент времени. Такое изменение реализуется принудительной записью нового значения в счетчик команд PC под управлением специальных аппаратных средств микроконтроллера, которые реагируют на внешние события. Включение в работу механизма принудительного изменения текущего значения счетчика команд нельзя считать аварийным состоянием микропроцессорной системы. Это лишь специальное состояние, которое позволяет организовать эффективное распределение ресурса одного центрального процессора для обслуживания нескольких устройств, генерирующих в реальном времени несвязанные между собой внешние события.

Встроенный в МК механизм реагирования на внешние события авторы данной книги именуют исключениями, поскольку внешние события нарушают нормальную, назначенную программистом последовательность исполнения команд. По способу обработки микроконтроллером исключения подразделяются на прерывания и сброс. В русскоязычной литературе термин «исключение» обычно не используется, и говорят просто о состоянии прерывания или о состоянии сброса микроконтроллера (примечание переводчика). Сохраняя оригинальный стиль авторов, далее в книге будем использовать термин «исключение».

4.9.1. Реакция МК на внешние события

Рассматривая далее технические особенности подсистемы прерывания МК семейства 68HC12/HCS12, мы должны обсудить общие для всех микропроцессорных систем алгоритмы обработки прерываний:

- Каждое событие, на которое микропроцессорная система должна реагировать с использованием механизма прерывания, называется запросом на прерывание. Последовательность команд, которая должна быть исполнена при возникновении запроса на прерывание, называется подпрограммой прерывания ISR (Interrupt Service Routing). При возникновении запроса на прерывание текущая исполняемая программа, которую в русскоязычной литературе называют фоновой, должна быть приостановлена для выполнения подпрограммы прерывания ISR. По завершении последней исполнение фоновой программы должно быть продолжено.
- В момент приостанова исполнения фоновой программы, содержимое всех регистров центрального процессора должно быть сохранено в специальной области ОЗУ, которая называется «стек». В составе центрального процессора обязательно имеется регистр «указатель стека SP», который содержит адрес области памяти, в которой сохранили значения остальных регистров центрального процессора. По завершении исполнения подпрограммы прерывания этот адрес будет использован для восстановления значений регистров центрального процессора из стека, чтобы далее продолжить исполнение фоновой программы.
- Аппаратный флаг, который был установлен внешним событием для генерации запроса на прерывание, должен быть обязательно сброшен до завершения исполнения подпрограммы прерывания ISR. Если этого не будет сделано, то МК снова перейдет к исполнению подпрограммы прерывания.

Таким образом, реакция МК на один и тот же запрос может получиться многократной, что не предусматривается алгоритмом управления.

- После завершения выполнения подпрограммы прерывания, содержимое счетчика команд и всех регистров центрального процессора восстанавливается из стека. В результате, исполнение фоновой программы возобновляется с того оператора, на котором она была приостановлена.
- Сразу после начального запуска МК должен выполнить процедуры инициализации (начальной установки), которые позволят настроить подсистему прерывания микроконтроллера необходимым образом. И лишь после этого можно разрешать обработку прерываний в микроконтроллере.

Отметив эти общие свойства подсистемы прерывания, обратимся к более подробному рассмотрению последней в МК семейства 68HC12/HCS12.

4.10. Состояния сброса и прерывания в МК 68HC12

МК семейства 68HC12/HCS12 обладают мощной системой обработки исключений. По способу реакции микроконтроллера на возмущающие события исключения делятся на прерывание и на сброс. В русскоязычной литературе используют следующие словосочетания с терминами «прерывание» и «сброс» (прим. переводчика):

- МК реализует прерывание или МК находится в состоянии прерывания;

Рис. 4.13. *Классификация исключений 68HC12*

- МК находится в состоянии сброса или в состоянии начального запуска.

Прерывания в свою очередь делятся на маскируемые и немаскируемые. Полный список источников исключений в МК семейства 68HC12/HCS12 с указанием способа обработки исключения по каждому источнику события приведен на рис. 4.13. В следующих параграфах мы обсудим особенности работы внутренних ресурсов МК в процессе обработки каждого упомянутого на рис. 4.13 исключения.

4.10.1. Состояние сброса МК

Микроконтроллер семейства 68HC12/HCS12 переходит в состояние сброса по внешнему сигналу или в при наступлении определенных внутренних событий. В состоянии сброса программный счетчик и часть битов регистра состояния центрального процессора, а также определенные в техническом описании регистры специальных функций периферийных модулей устанавливаются в начальное состояние. Это состояние однозначно определяет аппаратную конфигурацию микроконтроллера, с которого он начнет работу после включения питания. Поэтому второе название состояния сброса – состояние начального запуска. Состояние сброса МК использует также для восстановления работоспособного состояния после обнаружения внутренней аварийной ситуации.

Различают четыре источника событий, которые переводят МК в состояние сброса:

- ***Внешний сброс (External reset).*** Все МК семейства 68HC12/HCS12 имеют специальный вывод корпуса RESET для подачи сигнала внешнего сброса. Активный уровень сигнала – логический 0. Пока на входе RESET удерживается низкий уровень сигнала, МК будет находиться в состоянии сброса. После перевода линии RESET в состояние логической 1 МК перейдет в активный режим работы по истечении задержки, которая составляет 4096 периодов системной магистрали МК.
- ***Внутренний сброс по нарастанию напряжения питания (Power-on reset – POR).*** Нарастание напряжения на входе V_{DD} микроконтроллера вызывает состояние сброса. Таким образом реализуется начальный запуск МК с однозначно определенной аппаратной конфигурацией и с известным начальным адресом запускаемой на исполнение программы.
- ***Внутренний сброс по сторожевому таймеру (Computer Operating Properly reset – COP).*** Логика работы сторожевого таймера позволяет микроконтроллеру выявлять перемежающиеся ошибки в исполнении прикладной программы, которые могут возникнуть в результате электромагнитных помех или при колебаниях напряжения питания микропроцессорной системы. В процессе отладки работа сторожевого таймера запрещена. Работа модуля сторожевого таймера разрешается в конечном варианте прикладной программы, который используется при работе МК в системе. Сторожевой таймер – это счетчик, коэффициент счета которого настраивается пользователем при инициализации системы. Счетчик начинает счет внутренних тактовых импульсов в момент начала исполнения программы. Если счетчик переполнится, то МК перейдет в состояние сброса. Правильно исполняемая прикладная программа, в которой очередность исполнения операторов совпадает с предусмотренной програм-

мистом очередностью, должна постоянно сбрасывать сторожевой таймер. Тогда внутреннего сброса от него случаться не будет. Для сброса сторожевого таймера в МК семейства 68HC12/HCS12 необходимо в регистр COPRST записать сначала код $55, а затем код $AA. При создании конечного кода прикладной программы разработчик должен разместить операции записи приведенной последовательности кодов так, чтобы исполнение программы по любому возможному пути обеспечивало бы выполнение команд сброса через меньшие интервалы времени, чем период переполнения сторожевого таймера.

- ***Внутренний сброс по отклонению частоты тактовых импульсов МК (Clock Monitor reset).*** МК переводится в состояние сброса, когда модуль встроенного генератора тактирования обнаруживает выход частоты тактирования за заданные пределы или просто останов системы тактирования.

После перехода в состояние сброса в счетчик команд центрального процессора

Имя регистра: COPCTL Адрес: $0016

7	6	5	4	3	2	1	0
CME	FCME	FCM	FCOP	DISR	CR2	CR1	CR0
Сброс: 0	0	0	0	0	0	0	1
(Нормальный режим работы)							
Сброс: 0	0	0	0	1	0	0	1
(специальный режим работы)							

Имя регистра: COPRST Адрес: $0017

7	6	5	4	3	2	1	0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Сброс: 0	0	0	0	0	0	0	0

Период переполнения сторожевого таймера

CR[2:0]	Делитель	Период переполнения сторожевого таймера	
		4 МГц	8 МГц
000	Не используется	нет	нет
001	2^{13}	2,048 мс	1,024 мс
010	2^{15}	8,192 мс	4,096 мс
011	2^{17}	32,768 мс	16,384 мс
100	2^{19}	131,072 мс	65,536 мс
101	2^{21}	524,288 мс	262,144 мс
110	2^{22}	1,048 с	524,288 с
111	2^{23}	2,097 с	1,048576 с

Рис. 4.14. *Формат регистров COPCTL, COPRST*

автоматически загружается так называемый вектор сброса. Вектор сброса – это адрес ячейки памяти, в которой расположен первый оператор исполняемой программы. После загрузки начального адреса программа запускается на исполнение. Два источника сброса: сторожевой таймер и монитор частоты тактирования – обладают собственными векторами сброса (см. рис. 4.17).

Регистры сторожевого таймера и монитора тактирования

Два регистра специальных функций используются МК семейства 68HC12/HCS12 для управления генерацией сигналов сброса от сторожевого таймера и монитора тактирования:

- COPCTL – регистр управления сторожевого таймера.
- COPRST – регистр сброса сторожевого таймера.

Формат обоих регистров представлен на рис. 4.14. Отдельные биты регистра COPCTL разрешают или запрещают сброс от сторожевого таймера и от монитора питания, задают период переполнения сторожевого таймера, а также позволяют генерировать сигнал сброса под управлением программы.

Назначение битов регистра COPCTL:

- CME – разрешает (CME=1) или запрещает (CME=0) работу монитора тактирования.
- FCME – разрешает работу монитора тактирования (FCME=1) независимо от значения бита CME.
- FCM – установка под управлением программы этого бита в 1 генерирует внутренний сброс по монитору тактирования.
- FCOP – установка под управлением программы этого бита в 1 генерирует внутренний сброс по переполнению сторожевого таймера.
- DISR – запрещает (при DISR=1) или запрещает (при DISR=0) перезапуск МК по установленным битам FCM и FCOP.
- CR2...CR0 - задают коэффициент счета сторожевого таймера. Численные значения периода переполнения сторожевого таймера для некоторых частот системной шины МК приведены на рис. 4.14.

Регистр COPRST предназначен для сброса сторожевого таймера. В этот регистр под управлением программы должна быть записана последовательность кодов: сначала $55, затем $AA. И сторожевой таймер будет сброшен.

4.10.2. Прерывания

Мы рассмотрели один из способов реализации исключений в микроконтроллерах – это сброс МК. Другой способ принудительного изменения содержимого программного счетчика центрального процессора – это прерывания, которые в МК семейства 68HC12/HCS12 делятся на маскируемые и немаскируемые.

Программно-логическая модель центрального процессора содержит регистр признаков CCR, формат которого представлен на рис. 4.15. Особенностью всех семейств МК от компании Freescale Semicinductor является наличие в регистре признаков не только флагов результатов операции, но и дополнительных битов управления подсистемой прерывания. В составе регистра признаков МК 68HC12/HCS12 – два таких бита. Бит I – глобальная маска прерываний – используется для управления маскируемыми прерываниями. Бит X – бит запрета немаскируемых прерываний – управляет немаскируемыми прерываниями. Оба этих бита устанавливаются в

1 в состоянии сброса МК. Обратите внимание, в МК 68HC12/HCS12 установка битов I и X запрещает соответствующие прерывания.

Немаскируемые прерывания

В соответствие со своим названием немаскируемые прерывания не могут быть отключены пользователем. Однако в предыдущем абзаце было упомянуто, что установка бита X в 1 запрещает немаскируемые прерывания. Значение бита X действительно равно 1 в состоянии сброса МК. Однако далее он может быть установлен в 0 под управлением программы инициализации, разрешая тем самым немаскируемые прерывания. Далее этот бит не может быть изменен под управлением программы, и в этом его отличие от бита глобальной маски прерываний I.

Три типа немаскируемых прерываний реализуются в МК 68HC12/HCS12:

- ***Прерывание по внешнему запросу*** $\overline{XIRQ}$**.** Все МК 68HC12/HCS12 имеют вывод внешнего немаскируемого прерывания $\overline{XIRQ}$. Активный уровень сигнала для генерации запроса на прерывание – логический 0.
- ***Прерывание по несуществующему коду команды.*** Каждая инструкция языка ассемблер МК имеет собственный код. В МК 68HC12/HCS12 коды операций могут быть однобайтовыми и двухбайтовыми. Но не все теоретически возможные коды использованы для кодирования реальных команд процессорного ядра CPU12. Если на этапе выборки кода команды из памяти произошло считывание несуществующего кода команды, то генерируется запрос на немаскируемое прерывание.
- ***Программное прерывание – инструкция SWI.*** Система команд МК 68HC12/HCS12 имеет инструкцию программного прерывания, которая

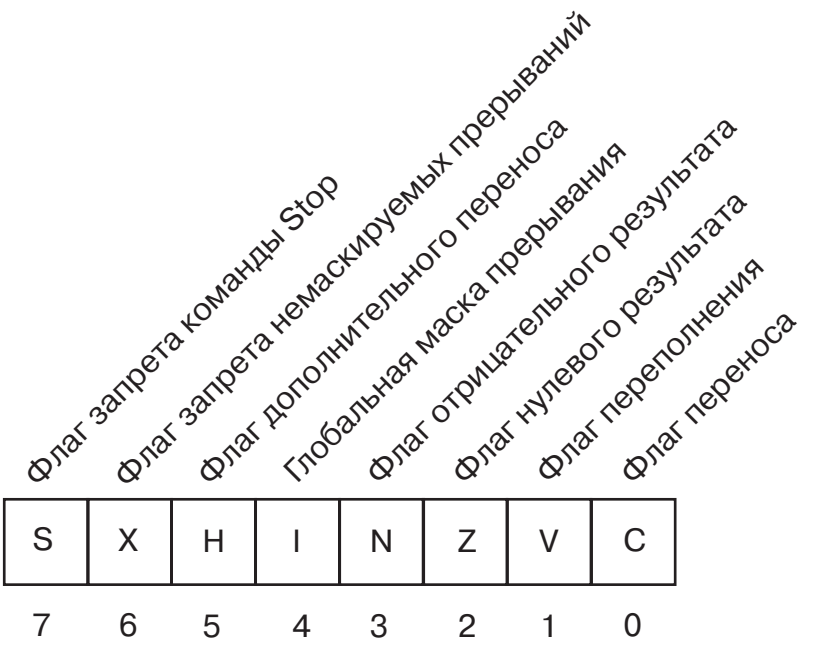


Рис. 4.15. *Формат регистра состояния CCR*

позволяет перейти к исполнению подпрограммы прерывания из прикладной программы.

Маскируемые прерывания

- ***Прерывание по внешнему запросу*** $\overline{IRQ}$ **.** Все МК 68HC12/HCS12 имеют вывод внешнего маскируемого прерывания $\overline{IRQ}$. Активный уровень сигнала для генерации запроса на прерывание – логический 0. В некоторых приложениях требуется принять запросы от нескольких внешних источников сигналов. Для таких случаев следует использовать дополнительный логический элемент, который объединяет запросы от всех источников по логике ИЛИ (рис. 4.16). Если запрос на вход $\overline{IRQ}$ поступил, и МК перешел к выполнению подпрограммы прерывания, то в этой подпрограмме следует опросить линии порта для того, чтобы установить, какой из источников вызвал прерывание. Обработка нескольких объединенных по ИЛИ запросов с программным поиском установившего запрос источника называется поллингом.
- ***Прерывание по таймеру меток реального времени RTI.*** Таймер меток реального времени генерирует последовательность равноотстоящих во времени запросов на прерывание. Период повторения запросов настраивается программистом. Эти прерывания могут быть использованы для регулярного выполнения микроконтроллером некоторой задачи. Например, для измерения напряжения аккумуляторной батареи каждые три мин, чтобы сигнализировать о необходимости ее замены. Мы рассмотрим особенности прерываний RTI в главе 7 на примере управления скоростью вращения электрическим двигателем.
- ***Прерывание по событию канала захвата/сравнения (IC/OC) таймера.*** Восемь одинаковых блоков в составе модуля таймера, которые именуют «каналами», предназначены для контроля за уровнем сигнала на входе канала или для изменения в строго определенный момент времени логического уровня на выходе канала. Заданное программистом изменение входного или выходного сигнала канала рассматривается как событие, которое генерирует запрос на прерывание. Например, если канал настроен на слежение за перепадом входного сигнала из 1 в 0, то когда такое изменение произойдет, будет выставлен запрос на прерывание.
- ***Прерывание по переполнению таймера.*** Основным блоком модуля таймера является 16-разрядный счетчик временной базы. Этот счетчик невозможно остановить. Также невозможно изменить его коэффициент счета, который составляет $2^{16} = 65536$. Поэтому регулярно счетчик временной базы изменяет свой код с $FFFF на $0000. Такое изменение кода называют переполнением счетчика. В момент переполнения по желанию программиста может генерироваться запрос на прерывание, в то время как счетчик продолжает считать дальше. Такие прерывания особенно удобны при необходимости измерения очень больших временных интервалов. Для этого в микроконтроллере производят подсчет, сколько переполнений счетчика произошло за этот временной интервал, и, зная период счета счетчика, определяют длительность исследуемого временного интервала.
- ***Прерывание по переполнению счетчика внешних событий.*** Когда счетчик внешних событий переполняется, то может генерироваться запрос на прерывание точно так же, как и для счетчика временной базы.
- ***Прерывание по событию на входе счетчика внешних событий.*** Этот запрос на прерывание формируется, если сигнал на входе счетчика внешних событий

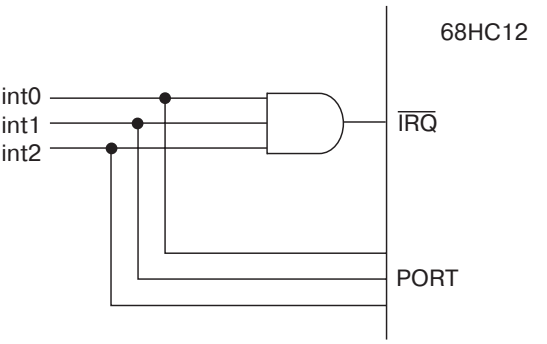


Рис. 4.16. *Внешняя цепь для подключения сигналов нескольких внешних запросов на прерывание*

изменил свое значение. Характер изменения, т.е. перепад из 0 в 1, или из 1 в 0, или любое изменение логического уровня, определяется программистом.

- ***Прерывание от модулей контроллеров последовательного ввода/вывода SCI и SPI.*** Каждый модуль последовательного ввода/вывода формирует целую группу запросов на прерывание: при завершении передачи слова, при приеме слова, при обнаружении различного рода нарушений в протоколе передачи информации.
- ***Прерывание от модуля АЦП.*** Модуль аналого-цифрового преобразователя формирует запрос на прерывание, когда процесс оцифровки очередного сигнала завершен, и двоичный код сигнала может быть считан в память МК.
- ***Прерывание при выходе МК из энергосберегающих режимов.*** Это прерывание позволяет вывести МК из состояния STOP или WAIT, в котором он находился с целью снижения потребляемой энергии. Такие прерывания очень полезны при объединении нескольких МК в информационную сеть. Мы рассмотрим этот тип прерывания более подробно в следующем параграфе.

Вопросы для самопроверки

1. Каковы различия между прерыванием и сбросом?

   Ответ: И прерывание, и сброс МК вызывают принудительный останов выполнения текущей прикладной программы. Поэтому эти два состояния МК характеризуют общим термином «исключение». Сброс МК обычно вызывается нарушениями в работе узлов МК или ошибках при исполнении прикладной программы. Когда такие неисправности возникают, состояние сброса переводит МК в некоторое начальное состояние, из которого и аппаратные средства и программа могут начать правильное функционирование. Таким образом, возникшая не фатальная неисправность будет устранена. В отличие от сброса, запросы на прерывание генерируются при штатной работе микропроцессорной системы. Они используются для организации работы системы в реальном времени. Запросы могут генерироваться встроенными в МК периферийными мо-

дулями или внешними сигналами. В отличие от сброса, при обслуживании запроса на прерывание выполняется специальная подпрограмма прерывания.

2. Каково различие между маскируемыми и немаскируемыми прерываниями?
   Ответ: Немаскируемое прерывание не может быть запрещено программистом. Напротив, маскируемое прерывание может быть многократно разрешено и запрещено в ходе исполнения прикладной программы.
3. Каково различие между прерываниями по входам $\overline{XIRQ}$ и $\overline{IRQ}$?
   Ответ: И $\overline{XIRQ}$, и $\overline{IRQ}$ являются входами МК для подключения внешних источников запросов на прерывания. Но прерывания по входу $\overline{IRQ}$ – маскируемые, а по входу $\overline{XIRQ}$ – немаскируемые.
4. Как организовать подсистему прерывания с несколькими внешними запросами для МК семейства 68HC12/HCS12, используя лишь один вход внешнего прерывания $\overline{IRQ}$.
   Ответ: Вход внешнего прерывания $\overline{IRQ}$ обладает низким активным уровнем сигнала. Это означает, что если сигнал на этом входе изменил свое значение с 1 на 0, то в МК генерируется запрос на прерывание. Множество сигналов запросов на прерывание следует объединить по И. Кроме того, каждый из этих сигналов следует подключить к отдельному входу порта МК. Тогда, если все сигналы запросов находятся в неактивном состоянии, на выходе элемента формируется И высокий логический уровень, и вход $\overline{IRQ}$ также в находится в неактивном состоянии. Если же на одном из входов запросов устанавливается низкий логический уровень, то на входе $\overline{IRQ}$ формируется активный уровень запроса, в результате по истечении некоторого времени МК переходит к выполнению подпрограммы прерывания. В этой подпрограмме МК опрашивает последовательно все линии порта, к которым подключены сигналы внешних запросов с целью определения, какой из сигналов вызвал прерывание. Далее МК переходит на подпрограмму обслуживания именно этого запроса. Схема подключения сигналов запросов к МК по предложенному способу приведена на рис. 4.16.

4.10.3. Вектора исключений

При переходе микроконтроллера в состояние прерывания или сброса должна реализовываться некоторая внутренняя последовательность действий, которая приведет к изменению текущего состояния счетчика команд центрального процессора, т.е. к исключению. Последнее вызовет исполнение подпрограммы прерывания или программного фрагмента начального запуска МК. Причем и подпрограмма прерывания, и программный фрагмент начального запуска должны быть разными и соответствовать тому событию, которое вызвало конкретное исключение. Другими словами, МК должен обладать аппаратными средствами, которые позволят ему начать программу с адреса, который определяется источником исключения.

В МК семейства 68HC12/HCS12 начальные адреса подпрограмм прерывания и начального запуска располагаются в специальной области памяти, которая называется таблицей векторов прерывания. Таблица векторов прерывания размещается в последних 128 ячейках резидентной линейно адресуемой Flash памяти программ. Это означает, что независимо от того, каким реальным объе-

Адреса вектора	Источник исключения	Глобальная маска в регистре CCR	Биты разрешения прерывания в регистрах специальных функций		Значение регистра HPRIO для установления наивысшего уровня приоритета
			Регистр	Бит	
$FFFE, $FFFF	Внешний сброс	нет	нет	нет	–
$FFFC, $FFFD	Сброс от системы тактирования	нет	COPCTL	CME, FCME	–
$FFFA, $FFFB	COP	нет	нет	COP rate selected	–
$FFF8, $FFF9	Сброс по несуществующему коду команды	нет	нет	нет	–
$FFF6, $FFF7	Программное прерывание SWI	нет	нет	нет	–
$FFF4, $FFF5	Внешнее прерывание XIRQ	X	нет	нет	–
$FFF2, $FFF3	Внешнее прерывание IRQ	I	INTCR	IRQEN	$F2
$FFF0, $FFF1	Метки реального времени RTI	I	RTICTL	RTIE	$F0
$FFEE, $FFEF	Канал 0 таймера	I	TMSK1	C0I	$EE
$FFEC, $FFED	Канал 1 таймера	I	TMSK1	C1I	$EC
$FFEA, $FFEB	Канал 2 таймера	I	TMSK1	C2I	$EA
$FFE8, $FFE9	Канал 3 таймера	I	TMSK1	C3I	$E8
$FFE6, $FFE7	Канал 4 таймера	I	TMSK1	C4I	$E6
$FFE4, $FFE5	Канал 5 таймера	I	TMSK1	C5I	$E4
$FFE2, $FFE3	Канал 6 таймера	I	TMSK1	C6I	$E2
$FFE0, $FFE1	Канал 7 таймера	I	TMSK1	C7I	$E0
$FFDE, $FFDF	Переполнение счетчика таймера	I	TMSK1	TOI	$DE
$FFDC, $FFDD	Переполнение счетчика внешних событий	I	PACTL	PAOVI	$DC

Рис. 4.17. *Таблица векторов сброса и прерывания для МК MC68HC912B32*

$FFDA, $FFDB	Событие на входе счетчика внешних событий	I	PACTL	PAI	$DA
$FFD8, $FFD9	Контроллер SPI	I	SP0CR1	SPIE	$D8
$FFD6, $FFD7	Контроллер SCI	I	SP0CR2	TIE,TCIE,RIE,ILIE	$D6
$FFD4, $FFD5	зарезервирован	I	–	–	$D4
$FFD2, $FFD3	Модуль ATD	I	ATDCTL2	ASCIE	$D2
$FFD0, $FFD1	Модуль BDLC	I	BCR1	IE	$D0
$FF80, $FFC1	зарезервирован	I	–	–	$80-$C0
$FFC2, $FFC9	зарезервирован	I	–	–	$C2-$C8
$FFCA, $FFCB	Переполнение счетчика внешних событий B	I	PBCTL	PBOVI	$CA
$FFCC, $FFCD	Переполнение счетчика	I	MCCTL	MCZI	$CC
$FFCE, $FFCF	зарезервирован	I	–	–	$CE

Рис. 4.17. *Таблица векторов сброса и прерывания для МК MC68HC912B32*

мом резидентного ПЗУ обладает конкретная модель МК, имеет или не имеет этот МК страничную адресацию памяти программ, таблица векторов прерывания будет помещена в адресном пространстве $FF80...$FFFF. Причем, не все 128 ячеек памяти могут быть заняты векторами исключений. Объем таблицы векторов определяется числом источников прерываний и сброса в конкретной модели МК.

На рис. 4.17 приведен формат таблицы векторов прерываний для МК B32. В первой колонке приведены адреса двух ячеек памяти, в которых должен располагаться двухбайтовый адрес начала подпрограммы прерывания или сброса. Во второй колонке указан источник события исключения, подпрограмма обслуживания которого должна начинаться с адреса, записанного в соответствующих ячейках памяти. Если, например, на входе внешнего запроса $\overline{IRQ}$ будет установлен низкий логический уровень, и МК перейдет к обслуживанию прерывания по этому запросу, то аппаратные средства МК считают 16-разрядный код из ячеек памяти с адресами $FFF2, $FFF3 и передадут его в счетчик адреса PC центрального процессора. Так начнет исполняться подпрограмма прерывания по внешнему запросу $\overline{IRQ}$. В третьей колонке таблицы рис. 4.17 указан бит глобальной маски для каждого источника исключения. Обратите внимание, что все источники сброса и программное прерывание по команде SWI невозможно запретить под управлением программы. Остальные прерывания могут быть запрещены либо X–, либо I-битом в регистре признаков CCR центрального процессора. Причем установленный в 1 бит I запрещает сразу все прерывания, напротив которых он упомянут в таблице. Поэтому этот бит и носит название глобальной маски прерывания. В колонках 4 и 5 приведены имена регистров специальных функ-

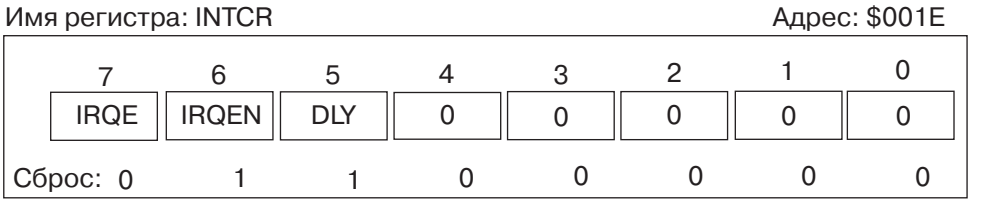


Рис. 4.18. *Формат регистра INTCR*

ций и битов этих регистров, которые являются битами разрешения для соответствующих запросов на прерывание. Эти биты позволяют разрешить или запретить прерывание только от одного источника запроса. Для того чтобы запрос на прерывание поступил в центральный процессор, необходимо, чтобы глобальная маска прерывания I была сброшена, и бит разрешения соответствующего прерывания был установлен в 1. Например, с помощью таблицы рис. 4.17 можно получить справку о том, что прерывание по внешнему запросу $\overline{IRQ}$ будет разрешено, если бит разрешения внешнего прерывания IRQEN в регистре управления прерываниями INTCR будет установлен в 1, а бит глобальной маски прерывания I в 0. Формат регистра управления внешним прерыванием INTCR приведен на рис. 4.18. Последняя колонка таблицы рис. 4.17 содержит шестнадцатеричный код, который необходимо загрузить в регистр HPRIO для того, чтобы назначить соответствующему запросу наивысший уровень приоритета.

Внимательный читатель должен был заметить, что адреса ячеек памяти, в которых располагаются вектора исключений, находятся в защищенной области памяти. Эту область памяти невозможно стереть и затем занести в нее новые вектора. Для того чтобы программист в процессе отладки все-таки имел возможность использования подсистемы прерывания с произвольными векторами входа в подпрограммы, в отладочном режиме работы МК семейства 68HC12/HCS12 используют дополнительную таблицу векторов, которая располагается в незащищенной области памяти. Соответствие адресов таблиц векторов прерывания в отладочном и пользовательском режиме работы приведено на рис. 4.19. На рис. 4.19 представлены альтернативные адреса размещения векторов прерываний для МК MC68HC912B32. В других моделях МК альтернативная таблица может располагаться в области ОЗУ, поскольку она предназначена только для целей отладки.

4.10.4. Система приоритетов для исключений

Среди представленного множества событий, которые могут вызвать исключения, ряд событий обладают большей значимостью для системы, чем другие. В микропроцессорной технике значимость события характеризуют термином «приоритет». События большей значимости обладают более высоким приоритетом. Рассматривая множество исключений для 68HC12/HCS12, можно отметить, что все немаскируемые источники исключений по умолчанию наделены более высоким приоритетом, нежели маскируемые. Поэтому, если в один и тот же момент времени поступят два запроса, то первым будет обслужен немаскируемый запрос, который обладает более высоким приоритетом. Затем будет выполнена

Адреса вектора	Источник исключения	Адрес передачи управления
$FFC0 – $FFCF	*зарезервирован*	$F7C0 – $F7CF
$FFD0	Модуль BDLC	$F7D0
$FFD2	Модуль ATD	$F7D2
$FFD4	зарезервирован	$F7D4
$FFD6	Контроллер SCI	$F7D6
$FFD8	Контроллер SPI	$F7D8
$FFDA	Событие на входе счетчика внешних событий	$F7DA
$FFDC	Переполнение счетчика внешних событий	$F7DC
$FFDE	Переполнение счетчика таймера	$F7DE
$FFE0	Канал 7 таймера	$F7E0
$FFE2	Канал 6 таймера	$F7E2
$FFE4	Канал 5 таймера	$F7E4
$FFE6	Канал 4 таймера	$F7E6
$FFE8	Канал 3 таймера	$F7E8
$FFEA	Канал 2 таймера	$F7EA
$FFEC	Канал 1 таймера	$F7EC
$FFEE	Канал 0 таймера	$F7EE
$FFF0	Метки реального времени RTI	$F7F0
$FFF2	Внешнее прерывание IRQ	$F7F2
$FFF4	Внешнее прерывание XIRQ	$F7F4
$FFF6	Программное прерывание SWI	$F7F6
$FFF8	Сброс по несуществующему коду команды	$F7F8
$FFFA	COP	$F7FA
$FFFC	Сброс от системы тактирования	$F7FC
$FFFE	Внешний сброс	$F7FE

Рис. 4.19. *Таблица адресов, которым передается управление исключениями в пользовательском и отладочном режимах работы МК*

подпрограмма обслуживания запроса с низшим приоритетом, в нашем случае этот запрос маскируемый.

Немаскируемые запросы исключений также ранжируются по приоритету. В приведенном ниже списке приоритет событий снижается с увеличением номера записи:

1. Внешний сброс по входу $\overline{RESET}$ или сброс по нарастанию напряжения питания POR;
2. Сброс по монитору системы тактирования;
3. Сброс по переполнению сторожевого таймера COP;
4. Программное прерывание SWI;
5. Немаскируемое прерывание по входу $\overline{XIRQ}$.

Очередность приоритетов маскируемых прерываний отображена на рис. 4.17. Чем выше строка источника события в таблице векторов, тем выше его приоритет. Однако приоритет маскируемого прерывания может быть повышен посредством записи соответствующей комбинации битов в регистр уровня приоритета HPRIO. Код, который следует записать в регистр HPRIO для назначения выбранному источнику запросов наивысшего приоритета среди маскируемых прерываний, приведен в последней колонке таблицы рис. 4.17. Следует помнить, что изменение кода в регистре HPRIO возможно только при единичном значении глобальной маски прерываний I, т.е. когда маскируемые прерывания запрещены. После повышения уровня прерывания какого-либо запроса остальные источни-

ки запросов сохраняют приведенное в таблице векторов прерываний рис. 4.17 распределение приоритетов.

Вопросы для самопроверки

1. Какой код должен быть записан в регистр HPRIO, чтобы приоритет прерывания от аналого-цифрового преобразователя стал наивысшим среди маскируемых прерываний?
   Ответ: Код $D2 следует записать в регистр уровня приоритета HPRIO.
2. Приведите запись выражения на Си, которое реализует действие вопроса 1.
   Ответ: HPRIO = 0xD2
3. Как изменятся приоритеты немаскируемых и маскируемых исключений после изменения приоритета прерывания от модуля АЦП в результате действий вопроса 1 или 2?
   Ответ: Приоритеты немаскируемых исключений останутся без изменения. Приоритеты все маскируемых прерываний, кроме АЦП, понизятся на один уровень, но при этом по отношению друг к другу их приоритеты не изменятся.
4. Какие действия должен предпринять программист, чтобы после начального запуска МК присвоить входу $\overline{IRQ}$ наивысший приоритет среди маскируемых прерываний?
   Ответ: Никаких. В соответствие с таблицей приоритетов на рис. 4.17 вход $\overline{IRQ}$ автоматически имеет наивысший приоритет среди маскируемых запросов.

4.10.5. Регистры подсистемы прерывания

Два регистра специальных функций используются для задания режимов подсистемы прерывания:

- INTCR – регистр управления внешним прерыванием по входу $\overline{IRQ}$;
- HPRIO – регистр уровня приоритета.

Формат регистра INTCR приведен на рис. 4.18. Для того чтобы разрешить прерывание по входу $\overline{IRQ}$, необходимо командой CLI сбросить глобальную маску прерывания I в регистре признаков CCR центрального процессора и установить в 1 бит разрешения внешнего прерывания IRQEN в регистре INTCR (бит 6). Бит 7 регистра INTCR, именуемый IRQE, конфигурирует линию $\overline{IRQ}$ на прием запроса при низком уровне сигнала (при IRQE = 0) или только по перепаду сигнала с 1 на 0 (при IRQE = 1). В первом случае, если активный низкий уровень на входе $\overline{IRQ}$ установлен и поддерживается внешним устройством после завершения исполнения подпрограммы прерывания, МК воспримет такую ситуацию как новый запрос и снова перейдет к исполнению подпрограммы прерывания. Так будет продолжаться до тех пор, пока на входе внешнего прерывания $\overline{IRQ}$ не установится логическая 1. Во втором случае, когда внешнее событие фиксируется только по перепаду сигнала $\overline{IRQ}$ из

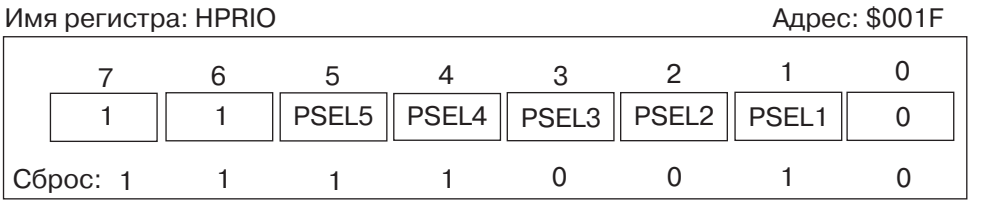


Рис. 4.20. *Формат регистра HPRIO*

1 в 0, в аналогичных условиях будет распознан только один запрос на прерывание, соответственно и подпрограмма прерывания будет исполнена только один раз.

Регистр уровня приоритета HPRIO предназначается для изменения уровня приоритета одного из источников маскируемых прерываний. Формат регистра HPRIO приведен на рис. 4.20. Если в регистр записать число, которое указано в правой колонке таблицы рис. 4.17, то соответствующий запрос получит наивысший приоритет среди группы маскируемых прерываний. В состоянии сброса МК регистр HPRIO устанавливается в $F2, что соответствует наивысшему уровню приоритета для запроса от внешнего устройства по входу $\overline{IRQ}$, что и отражено в таблице векторов рис. 4.17.

4.11. Процесс перехода к подпрограмме прерывания

На рис. 4.12 показана последовательность действий МК семейства 68HC12/HCS12 по обработке запроса на прерывание, начиная с момента поступления запроса и заканчивая возвратом из подпрограммы прерывания к продолжению исполнения основной программы. При изучении механизма перехода к подпрограмме прерывания и возврата из нее Вам следует обратить внимание на то, какие действия совершаются аппаратными средствами МК, т.е. автоматически, а какие требуют программной поддержки, т.е. должны сопровождаться написанием специального программного кода.

Первый шаг при разработке прикладной программы, которая будет содержать подпрограммы прерывания, – создание в памяти таблицы векторов прерываний. Для этого в ячейки памяти с указанными в техническом описании МК адресами должны быть помещены адреса начала соответствующих подпрограмм прерывания. При написании исходного текста программы начальные адреса подпрограмм прерывания могут быть указаны в абсолютных значениях, т.е. в численном виде. Однако это неудобно при отладке программы, когда в результате исправления ошибок эти адреса будут изменяться. Поэтому более грамотно указывать в таблице векторов прерывания символьные имена подпрограмм прерывания. При программировании на ассемблере для этой цели обычно используется псевдокоманда Ассемблера «DW».

Второй шаг при разработке программы с прерываниями – инициализация начального значения указателя стека SP. Область стека используется микроконтроллером для хранения значений регистров центрального процессора во время исполнения подпрограммы прерывания. В МК семейства 68HC12/HCS12 при пересылке данных в память указатель стека сначала уменьшается на единицу, и только затем содержимое какого-либо регистра загружается в память по новому адресу из SP. Поэтому начальное значение указателя стека должно быть равно увеличенному на единицу адресу последней ячейки в области стека. При программировании на Си программист обязан указать диапазон адресов области стека в опциях конфигурирования компилятора. Старший адрес этого диапазона будет загружен в указатель стека при исполнении файла Start.c.

Третий шаг – разрешить прерывания по выбранным источникам запросов. Это действие выполняется в два этапа. Сначала следует установить биты разрешения прерывания от каждого выбранного источника в соответствующем регистре специальных функций. Например, для прерывания от АЦП, должен быть установлен бит IRQEN в регистре INTCR. На втором этапе необходимо сбросить глобальную маску

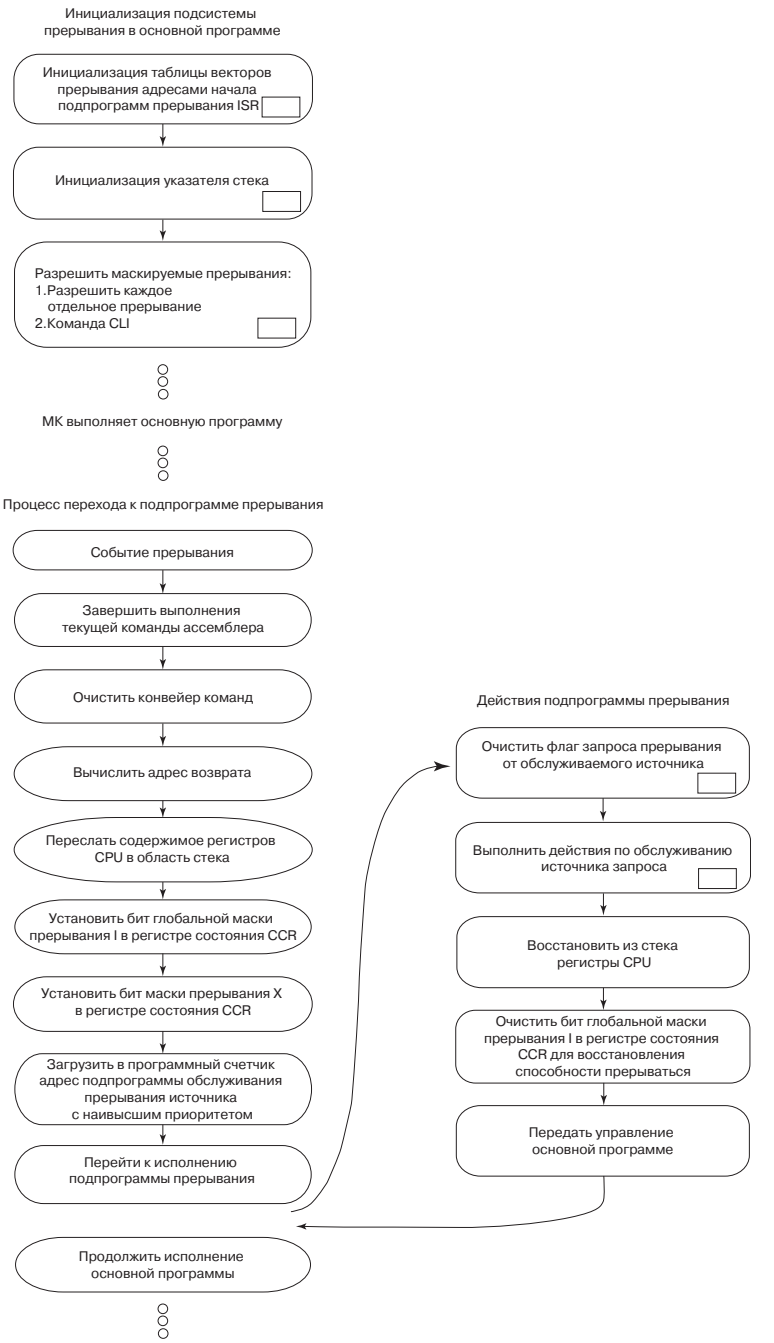


Рис. 4.21. *Последовательность действий МК при обслуживании запроса на прерывание*

прерывания I в регистре признаков CCR. При программировании на ассемблере следует использовать команду «CLI».

После того, как указатель стека инициализирован и прерывания разрешены, пишется текст основной программы, которая будет являться фоновой программой. Поступление запроса на прерывание во время исполнения фоновой программы вызывает в микроконтроллере определенную последовательность действий (рис. 4.21), которую принято называть процессом перехода к подпрограмме прерывания.

Наиболее вероятно, что запрос поступит в момент, когда центральный процессор уже начал, но еще не завершил исполнение очередной команды. В большинстве случаев исполнение текущей команды будет завершено, в счетчике PC будет сформирован адрес следующей команды основной программы, и только затем МК начнет процедуру перехода к подпрограмме прерывания. В отличие о других, МК семейства 68HC12/HCS12 обладают рядом специфических команд, время исполнения которых значительно превышает время исполнения традиционных команд ассемблера. К таким командам относятся инструкции нечеткой логики REV, REVW, WAV, которые могут потребовать для своего выполнения несколько сот циклов центрального процессора. Поэтому, при возникновении запроса, исполнение этих команд будет прервано с сохранением промежуточных результатов. Завершение исполнения инструкции произойдет после выполнения подпрограммы прерывания.

Центральный процессор МК семейства 68HC12/HCS12 оснащен двухступенчатым конвейером команд. Это означает, что во время исполнения текущей команды регистры конвейера команд хранят выбранные из памяти коды двух следующих команд. Когда запрос на прерывание поступает, нормальный порядок исполнения команд фоновой программы нарушается. Поэтому хранящиеся в конвейере команд коды становятся бесполезными, и конвейер автоматически очищается. После этого в счетчике команд автоматически восстанавливается адрес первой неисполненной команды фоновой программы, к которой МК должен будет вернуться после прерывания. Далее содержимое программного счетчика PC и других регистров центрального процессора сохраняется в стеке. Последовательность загрузки в стек следующая (рис. 4.22): старший байт счетчика команд PCH, младший байт счетчика команд PCL, старший YH и младший YL байты индексного регистра Y, старший XH и младший XL байты индексного регистра X, аккумулятор B, аккумулятор A, регистр признаков CCR. Важно, что это сохранение выполняется микроконтроллером автоматически, никаких команд в подпрограмме прерывания для этого не требуется.

После того, как регистры будут сохранены в стеке, МК автоматически устанавливает глобальную маску прерывания I в 1, запрещая тем самым другие маскируемые прерывания. Бит маски внешнего прерывания по входу $\overline{XIRQ}$ также устанавливается в 1.

Поэтому подпрограмма прерывания не может быть прервана даже немаскируемым внешним прерыванием. Далее в программный счетчик центрального процессора автоматически загружается адрес начала подпрограммы прерывания. Этот адрес извлекается из ячеек памяти таблицы векторов прерывания. Выбор необходимого адреса внутри таблицы векторов осуществляется аппаратными средствами микроконтроллера. Этот адрес соответствует источнику обрабатываемого прерывания.

По завершении исполнения подпрограммы прерывания, МК восстанавливает регистры центрального процессора из стека, включая программный счетчик PC, сбрасывает биты масок I и X в регистре CCR. Таким образом, в МК восстанавливается состояние центрального процессора, которое было до исполнения подпрограммы прерывания. И процесс выполнения фоновой программы возобновляется. В исходном тексте подпрограммы прерывания для ее корректного завершения записывается команда возврата из прерывания

RTI. Именно эта команда реализует действия по восстановлению регистров из стека и сбросу масок прерывания. Обратите внимание, в начале подпрограммы прерывания для выгрузки регистров в стек и установки масок I и X не требуется команд. А для выполнения обратных действий программист должен воспользоваться командой RTI. При написании программы на Си подпрограмма прерывания оформляется как функция. При этом если объявлено, что функция является подпрограммой прерывания, то команда ассемблера RTI подставляется в исходный текст автоматически. Мы рассмотрим процесс составления исходного текста программы с прерываниями на Си в следующем параграфе.

Вопросы для самопроверки

1. В приведенном списке перечислены действия МК в процессе перехода к подпрограмме прерывания. Укажите, какие действия выполняются аппаратными средствами МК, а какие требуют написания соответствующего кода в управляющей программе.
   - Инициализация таблицы векторов прерываний (программист);
   - Инициализация указателя стека (программист);
   - Разрешение прерываний в системе (программист);
   - Завершение выполнения текущей команды (МК);
   - Обнуление регистров конвейера команд (МК);
   - Вычисление адреса возврата в основную программу (МК);
   - Запоминание состояния регистров в стеке (МК);
   - Установка в 1 бита I (МК);
   - Загрузка из памяти вектора прерывания с максимальным приоритетом (МК);
   - Передача управления подпрограмме прерывания (МК);
   - Сброс сигнала запроса обслуживаемого прерывания (программист или МК, в зависимости от источника прерывания);
   - Восстановление регистров центрального процессора из стека (программист командой RTI);
   - Сброс маски I (МК);
   - Возврат к исполнению основной программы (МК).

2. Каким образом МК семейства 68HC12/HCS12 определяет приоритет обслуживаемого запроса на прерывание?

Адрес ячейки памяти	Содержимое ячейки стека
Указатель стека – 2	Старший байт адреса возврата; младший байт адреса возврата
Указатель стека – 4	Старший байт регистра Y; младший байт регистра Y
Указатель стека – 6	Старший байт регистра X; младший байт регистра X
Указатель стека – 8	Аккумулятор B; Аккумулятор A
Указатель стека – 9	Регистр состояния CCR

Рис. 4.22. *Последовательность загрузки регистров центрального процессора в стек*

Ответ: Последовательность расположения источников запросов в таблице векторов прерывания определяет их приоритет.

3. Часто в основной программе, которая содержит подпрограммы прерывания, одной из первых команд является команда SEI. Объясните почему?
   Ответ: Простановка команды запрета маскируемых прерываний SEI в начале программы – своеобразный «хороший тон» при программировании. Эта команда запрещает прерывания во время инициализации периферии МК, т.е. когда МК находится в процессе создания своей внутренней структуры для решения конкретной задачи. Когда инициализация будет завершена, программист разрешит прерывания командой CLI.
4. Зачем МК семейства 68HC12/HCS12 очищает конвейер команд при прерываниях?
   Ответ: При переходе на подпрограмму прерывания инструкции, коды которых хранятся в регистрах конвейера команд, будут отложены для исполнения. Поэтому их коды не сохраняются, и конвейер команд начинает выборку следующих за первой команд подпрограммы прерывания.

4.12. Оформление подпрограммы прерывания на Си

В данном параграфе мы рассмотрим основные особенности формирования исходного текста программы с прерываниями на Си. Конкретные примеры последуют позднее при рассмотрении конкретных устройств управления. На протяжении всех этих примеров мы будем использовать синтаксис компилятора ImageCraft ICC12. При использовании других компиляторов Вы встретитесь с теми же особенностями программирования. Возможно, что их реализация будет сопровождаться некоторыми другими правилами записи (синтаксиса) исходного текста программы.

При программировании на Си Вы должны обязательно реализовать следующие этапы записи исходного текста:

1. При написании программы обработки прерывания на Си, имя подпрограммы обработки прерывания должно быть объявлено с использованием специальной директивы препроцессора. В компиляторе ImageCraft ICC12 для этой цели следует использовать директиву #pragma:

   #pragma interrupt_handler <name>

   В поле <name> следует записать имя подпрограммы прерывания, которое Вы будете далее использовать в тексте программы. Приведенная запись информирует компилятор о том, что функция с названным именем является подпрограммой прерывания.
2. Далее по тексту подпрограмма прерывания оформляется как обычная функция. Компилятор в процессе перевода исходного текста этой функции на Си в инструкции ассемблера автоматически подставит в конце подпрограммы команду возврата из прерывания RTI, потому что эта функция была объявлена подпрограммой прерывания (директива #pragma на этапе 1).
3. Для правильного функционирования МК в процессе прерывания необходимо инициализировать указатель стека. Его значение должно быть равно старше-

му адресу области оперативной памяти МК, увеличенному на единицу. Поскольку диапазоны памяти пользователя (как постоянной, так и оперативной) являются необходимыми установками в конфигурации компилятора, то функция инициализации указателя стека выполняется компилятором автоматически. Поэтому программист не должен записывать какой либо текст в программе для инициализации указателя стека. Зато следует проверить карту памяти в установках компилятора, которая обязательно должна совпадать с реальной проектируемой системой.

4. Подсистема прерывания будет функционировать корректно, если для нее сформирована таблица векторов прерывания. Мы уже обсуждали, что таблица векторов прерывания в МК B32 находится в области Flash памяти, которая защищена от перезаписи информации. Для того, чтобы пользователь имел возможность записать собственную таблицу векторов сброса и прерывания, в эту нестираемую область памяти записаны фиксированные вектора, которые передают управление по известным адресам в области перезаписываемой EEPROM памяти (см. рис. 4.19). По этим адресам программист должен вписать команду безусловного перехода JMP с адресом соответствующей подпрограммы обработки прерывания.
5. Каждый маскируемый источник запроса на прерывание должен быть разрешен установкой соответствующего бита в регистре управления периферийного модуля. Мы рассмотрим, как это записать на Си в последующих примерах.
6. После установки всех индивидуальных битов на разрешение прерывания, необходимо сбросить глобальную маску прерывания I. На ассемблере для этого используют команду CLI. При программировании на Си мы также воспользуемся этой командой, посредством следующих макросов:

```
#define CLI ( )    asm(«cli\n»);    //разрешить маскируемые прерывания
#define SEI ( )  asm(«sei\n»);     //запретить маскируемые прерывания
```

Далее по тексту программы, если необходимо разрешить прерывания, то следует ввести CLI ( ).

Ниже приведен пример инициализации подсистемы прерывания. Полную запись исходного текста программы с прерываниями мы рассмотрим после обсуждения модуля таймера.

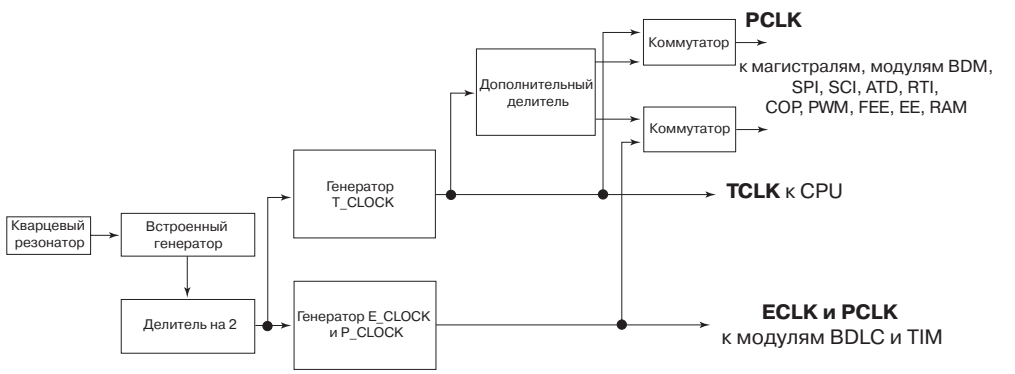


Рис. 4.23. *Структура модуля тактирования CGM*

```
//объявление функции в модуле
void toggle_isr(void);

//директива #pragma для указания, что функция является подпрограммой
//обслуживания прерывания

#pragma interrupt_handler toggle_isr

//инициализация соответствующего вектора в таблице векторов прерываний
#pragma abs_address: 0xF7EA            //B32 RAM-based vector address
void (*Timer_Channel_2_interrupt_vector[]) ()={toggle_isr};
#pragma end_abs_address
```

4.13. Система тактирования

Микроконтроллеры семейства 68HC12/HCS12 имеют в своем составе модуль генератора CGM (Clock Generation Module), который генерирует импульсные последовательности для тактирования центрального процессора, межмодульных магистралей, периферийных модулей в составе МК, а также внешние периферийные интегральные схемы. Структурная схема модуля CGM представлена на рис. 4.23.

Микроконтроллеры семейства 68HC12/HCS12 используют три внутренних сигнала тактирования: TCLK, ECLK и PCLK. Эти сигналы образуются путем деления эталонной импульсной последовательности внутреннего генератора с внешним кварцевым резонатором. Сигнал TCLK предназначен для тактирования центрального процессора, импульсные последовательности ECLK и PCLK используются для тактирования межмодульных магистралей и различных периферийных модулей (рис. 4.23). Модуль тактирования CGM микроконтроллера B32 оснащен также дополнительным делителем, который позволяет существенно снизить частоту одной из импульсных последовательностей тактирования. Низкая частота тактирования таймерных модулей в некоторых применениях позволяет значительно упростить управляющую программу.

4.13.1. Система тактирования отладочной платы MC68HC912B32EVB

Микроконтроллер в составе платы отладки MC68HC912B32EVB тактируется от внешнего кварцевого резонатора с частотой 16 МГц. В модуле генератора CGM эта частота делится на 2, образуя импульсную последовательность для тактирования центрального процессора и межмодульных магистралей микроконтроллера с частотой 8 МГц. Эта же импульсная последовательность используется для тактирования всех периферийных модулей микроконтроллера: таймера, контроллеров последовательного обмена, АЦП. Многие периферийные модули обладают собственным делителем частоты. Этот делитель позволяет создать собственную внутреннюю частоту тактирования модуля, которая может не коррелироваться с частотами других модулей. Это удобно при проектировании.

Кварцевый резонатор может быть отключен от входа МК посредством переключателей на плате MC68HC912B32EVB. Вместо кварцевого резонатора для тактирования МК может быть использован внешний генератор или керамический резона-

тор. Последний обладает меньшей, чем кварцевый резонатор, стабильностью частоты, однако он является более дешевым компонентом. Мы не будем вносить каких-либо изменений в схемотехнику платы MC68HC912B32EVB. Поэтому далее на протяжении всей книги частота внутренней шины и центрального процессора МК будет составлять 8 МГц.

Частота внутренней шины МК может быть понижена дополнительным делителем модуля CGM (рис. 4.23). Его коэффициент деления назначается под управлением программы и может составлять 1, 2, 4, ...128. Для выбора желаемого коэффициента деления необходимо записать соответствующий таблице рис. 4.24 код в разряды SLDV2...SLDV0 регистра управления дополнительным делителем SLOW. Формат этого регистра приведен на рис. 4.24.

Вы можете заинтересоваться, а зачем понижать частоту тактирования МК? Ведь тогда снизится его вычислительная производительность. Дело в том, что энергия потребления МК пропорциональна частоте его тактирования. Поэтому для достаточно медленных приложений, которые не требуют большой вычислительной производительности, частоту тактирования можно снизить ради экономии энергопотребления устройства управления. Понижение частоты тактирования возможно также для формирования таймерами необходимых временных интервалов. Например, если снизить частоту тактирования до 62,5 КГц (SLDV2...SLDV0 = 111), то период переполнения счетчика временной базы будет составлять около 1 с.

Имя регистра: SLOW Адрес: $00E0

	7	6	5	4	3	2	1	0
	0	0	0	0	0	SLDV2	SLDV1	SLDV0
Сброс:	0	0	0	0	0	0	0	0

Выбор пониженной частоты межмодульных магистралей МК

SLDV[2:0]	Коэффициент деления 2^X	При частоте внешнего кварцевого резонатора		
		16 МГц	8 МГц	4 МГц
000	1	8 МГц	4 МГц	2 МГц
001	2	4 МГц	2 МГц	1 МГц
010	4	2 МГц	1 МГц	500 кГц
011	8	1 МГц	500 кГц	250 кГц
100	16	500 кГц	250 кГц	125 кГц
101	32	250 кГц	125 кГц	62,5 кГц
110	64	125 кГц	62,5 кГц	31,2 кГц
111	128	62,5 кГц	31,2 кГц	15,6 кГц

Рис. 4.24. *Формат регистра SLOW*

4.14. Подсистема реального времени – модуль таймера

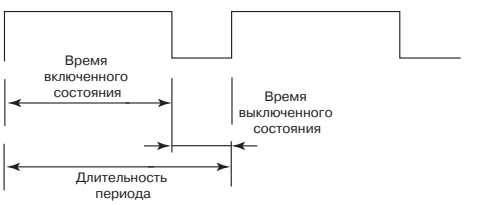
Подсистема реального времени МК семейства 68HC12/HCS12 включает основной таймерный модуль, который имеет две модификации – TIM и ECT, и отдельный таймер меток реального времени.

Структура модуля таймера TIM (Timer Interface Module) ориентирована на реализацию трех основных функций:

- ***Входного захвата (IC – Input Capture).*** Функция входного захвата позволяет производить измерения временных параметров сразу нескольких импульсных сигналов на входах МК. Подсистема входного захвата может быть настроена на измерение длительности единичного или нулевого состояния на входе порта (рис. 4.25, а), а также периода, коэффициента заполнения или частоты периодического импульсного сигнала (рис. 4.25, б).
- ***Выходного сравнения (OC – Output Compare).*** Функция выходного сравнения позволяет МК генерировать на нескольких выходах импульсные последовательности с заданными временными характеристиками, такими, как период и коэффициент заполнения для повторяющихся сигналов, длительность единичного или нулевого состояния для неповторяющихся сигналов.
- ***Счетчика внешних событий (PA – Pulse Accumulator).*** Основная функция этого счетчика – подсчет импульсов (внешних событий) на одном из входов МК. Он также может быть использован для измерения временных параметров внешнего импульсного сигнала большой длительности.

Для реализации функций входного захвата/выходного сравнения (IC/OC) модуль таймера TIM использует восемь идентичных аппаратных блоков, которые принято называть каналами. Каждый из каналов посредством программных установок настраивается на реализацию режима входного захвата или выходного сравнения независимо от режима ра-

Высокий активный уровень сигнала



Коэффициент заполнения

$$\text{Частота [Гц]} = \frac{1}{\text{длительность периода}} \qquad \gamma = \frac{\text{Время включенного состояния} \times 100\%}{\text{длительность периода}}$$

Рис. 4.25. *Временные характеристики ШИМ сигнала*

боты других каналов модуля таймера. Каждый из каналов использует общий счетчик временной базы для фиксации моментов наступления событий. Параллельная работа всех восьми каналов с одним счетчиком временной базы не вносит погрешностей в формируемые или измеряемые временные интервалы, поскольку фиксация этих интервалов реализуется на аппаратном уровне с последующим программным обслуживанием каналов по прерываниям. Каждый канал связан с одной из линий порта PORT T. Счетчик событий PA в составе модуля TIM также связан с линией 7 порта PORT T. Поэтому линии 0...6 порта PORT T в подсистеме таймера могут использоваться или как входы IC, или как выходы OC, в то время как линия 7 порта PORT T кроме этих двух функций IC/OC может также использоваться как вход тактовых импульсов для счетчика внешних событий.

4.14.1. Структура модуля таймера

Структурная схема модуля таймера представлена на рис. 4.26. При ее первичном рассмотрении она кажется запутанной и сложной. Для облегчения понимания мы выделим в составе таймера три блока: блок счетчика временной базы (1), восемь ка-

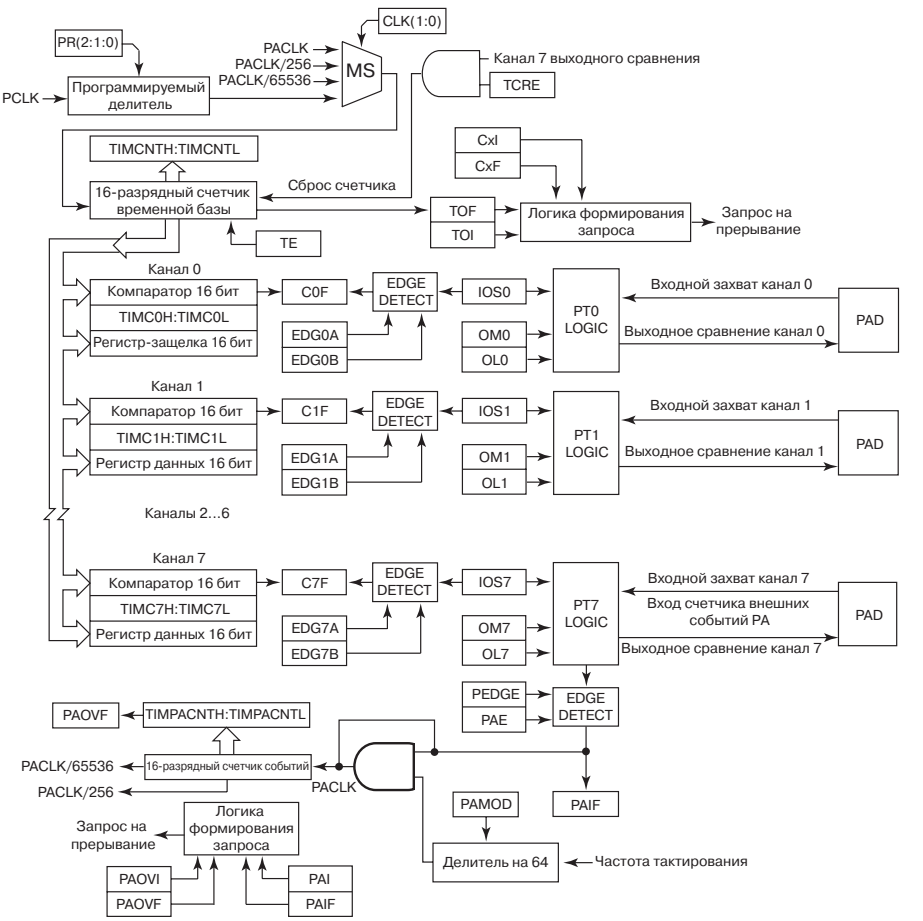


Рис. 4.26. *Структура модуля таймера*

налов с функциями IC/OC (2), блок счетчика внешних событий (3). Далее мы рассмотрим каждый из этих блоков отдельно, и для каждого из них приведем более понятные структурные схемы.

4.14.2. Счетчик временной базы

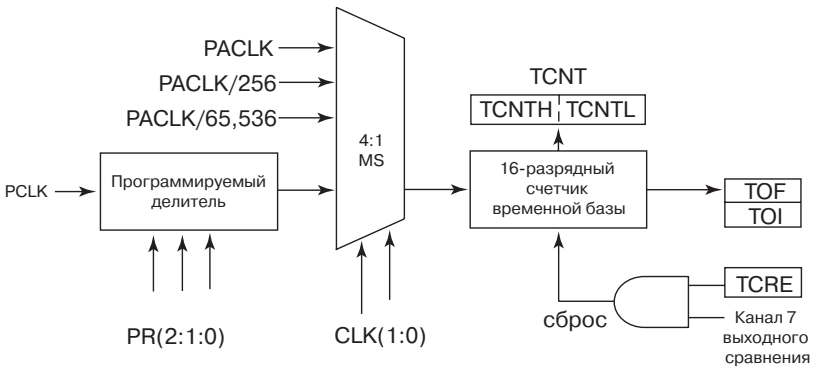
Основным блоком модуля таймера TIM является 16-разрядный счетчик временной базы TCNT, структурная схема которого представлена на рис. 4.27. Текущий код счетчика используется всеми каналами захвата/сравнения в качестве отсчета момента реального времени. Именно поэтому этот счетчик и называют счетчиком временной базы. Этот счетчик также называют свободно считающим счетчиком. Определение «свободно считающий» отражает следующую особенность работы счетчика. Если работа модуля таймера разрешена, то счетчик временной базы производит непрерывный счет, начиная с минимального кода $0000 до максимального кода $FFFF. При поступлении следующего тактового импульса код счетчика изменяется с $FFFF на $0000. Далее счет продолжается в порядке нарастания кода. Невозможно остановить счетчик под управлением программы, так же как и изменить коэффициент счета счетчика, равный 2^{16}. Текущее состояние счетчика отображается в двух 8-разрядных регистрах: TCNTH – старший байт счетчика, TCNTL – младший байт счетчика. В карте памяти МК эти регистры располагаются по следующим адресам: $0084 – TCNTH, $0085 – TCNTL. Вместе оба этих регистра составляют 16-разрядный регистр текущего состояния счетчика временной базы TCNT. Имя TCNT обычно объявляется в заголовочном файле.

Особенности счетчика временной базы

Счетчик временной базы тактируется импульсной последовательностью с выхода мультиплексора MUX. Ко входам мультиплексора подключены четыре источника тактирования: основная импульсная последовательность с выхода делителя частоты и дополнительные сигналы PACLK, PACLK/256, PACLK/65536. На вход программируемого делителя частоты подключен сигнал PCLK с выхода подсистемы тактирования. Частота следования импульсов на линии PCLK равна частоте тактирования межмодульных магистралей МК. Коэффициент деления программируемого делителя частоты определяется разрядами PR2...PR0 регистра масок таймера 2 (TMSK2). Формат регистра TMSK2 приведен на рис. 4.27. Таблица соответствия численного значения коэффициента деления двоичной комбинации разрядов PR2...PR0 представлена на рис. 4.28.

Величина коэффициента деления определяет длительность периода переполнения счетчика временной базы. Поскольку разрядность счетчика равна 16, то коэффициент счета этого двоичного счетчика равен 2^{16} или 65356. Минимальный период переполнения счетчика составляет 8169 мс (2^{16} импульсов $\times$ 1/8 МГц), поскольку максимальная частота импульсной последовательности PCLK, равная частоте тактирования межмодульных магистралей, составляет 8 МГц. Однако при задании максимального коэффициента деления, равного 32 (см. таблицу рис. 4.28), период переполнения счетчика составит уже 262424 мс (2^{16} импульсов $\times$ 32/8 МГц).

В некоторых приложениях необходимо формировать временные интервалы, значительно превышающие приведенные расчетные значения. Этого можно достигнуть путем снижения частоты тактирования счетчика временной базы.



Имя регистра: TCSR — Адрес: $0086

7	6	5	4	3	2	1	0
TEN	TSWAI	TSBCK	TFFCA	0	0	0	0
Сброс: 0	0	0	0	0	0	0	0

Имя регистра: TFLG2 — Адрес: $008F

7	6	5	4	3	2	1	0
TOF	0	0	0	0	0	0	0
Сброс: 0	0	0	0	0	0	0	0

Имя регистра: TMSK2 — Адрес: $008D

7	6	5	4	3	2	1	0
TOI	0	PUPT	RDPT	TCRE	PR2	PR1	PR0
Сброс: 0	0	0	0	0	0	0	0

Имя регистра: PACTL — Адрес: $00A0

7	6	5	4	3	2	1	0
0	PAEN	PAMOD	PEDGE	CLK1	CLK0	PAOVI	PAI
Сброс: 0	0	0	0	0	0	0	0

Имя регистра: TCNTH — Адрес: $0084

7	6	5	4	3	2	1	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Сброс: 0	0	0	0	0	0	0	0

Имя регистра: TCNTL — Адрес: $0085

7	6	5	4	3	2	1	0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.27 *Структура и регистры управления счетчика временной базы модуля таймера*

PR[2:1:0]	Коэффициент деления
000	1
001	2
010	4
011	8
100	16
101	32
110	*зарезервирован*
111	*зарезервирован*

Рис. 4.28. *Выбор коэффициента программируемого делителя для счетчика временной базы*

Однако уменьшение частоты импульсной последовательности PCLK связано со снижением производительности процессорного ядра. Поэтому для счетчика временной базы предусмотрены три альтернативных источника тактирования PACLK, PACLK/256, PACLK/65536. Выбор одного из четырех источников тактирования осуществляется двухразрядным кодом CLK1:CLK0 в регистре управления счетчиком внешних событий PACL. Формат регистра PACL представлен на рис. 4.27.

Флаг переполнения счетчика

Если длительность измеряемых или формируемых микроконтроллером временных интервалов превышает период переполнения счетчика временной базы, то возникает необходимость в подсчете числа периодов переполнения этого счетчика. При достижении максимального кода $FFFF счетчик не останавливается, он продолжает считать дальше. Поэтому при поступлении следующего тактового импульса в счетчике установится код $0000. Такое изменение кода называется событием переполнения счетчика и фиксируется установкой бита TOF в регистре управления счетчиком TFLG2 (рис. 4.27). Триггер TOF называют триггером переполнения счетчика. Этот триггер может быть считан под управлением программы, или, если прерывания по событию переполнения счетчика разрешены, то установленный в 1 триггер TOF генерирует запрос на прерывание. Прерывание по переполнению счетчика временной базы имеет свой собственный вектор и собственный бит разрешения прерывания TOI в регистре TMSK2 (рис. 4.27).

Если код счетчика временной базы изменился с $FFFF на $0000, то триггер TOF устанавливается в 1. При использовании этого флага для вызова прерываний следует позаботиться о том, чтобы флаг был сброшен под управлением подпрограммы прерывания до наступления следующего переполнения таймера. Для сброса (установки в 0) флага TOF следует записать 1 в бит 7 регистра TMSK2, т.е. выполнить любую команду установки в 1 этого флага. Вышесказанное не ошибка! Большинство флагов регистров специальных функций МК семейства 68HC12/HCS12, генерирующих разнообразные запросы на прерывание, сбрасываются в 0 посредством записи 1 в находящийся в состоянии 1 бит флага. Мы обсудим ниже варианты программного кода, которые могут быть использованы для сброса различных флагов, в том числе и флага переполнения счетчика временной базы.

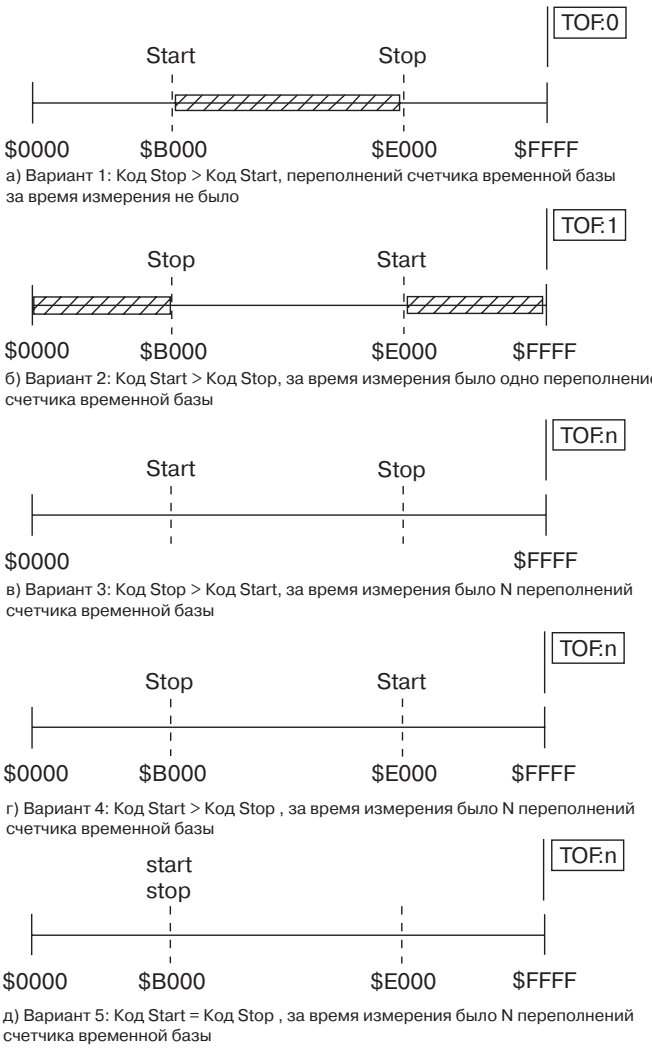


Рис. 4.29. *Диаграммы, поясняющие преобразования кодов для расчета длительности измеряемого временного интервала*

Если в прикладной задаче необходимо измерить временной интервал, значительно превышающий по длительности период переполнения счетчика временной базы, то следует организовать дополнительный счетчик в одной из ячеек памяти. Содержимое этого счетчика будет инкрементироваться подпрограммой прерывания при каждом переполнении счетчика.

Определение длительности временных интервалов

Во многих приложениях возникает необходимость определения длительности временного интервала между двумя изменениями сигнала на одной из линий порта. Эта задача может быть решена двумя способами.

При первом способе счетчик временной базы сбрасывается в момент первого изменения сигнала. Обнуление счетчика можно произвести в подпрограмме прерывания по событию входного захвата, которое соответствует первому изменению исследуемого сигнала. В этой же подпрограмме подсистема входного захвата перепрограммируется на детектирование второго события в исследуемом сигнале. Счетчик временной базы продолжает наращивать код. В момент второго изменения сигнала, которое фиксирует подсистема входного захвата, текущий код счетчика запоминается в специальном двухбайтовом регистре. Если в процессе слежения за сигналом переполнений счетчика временной базы не было, то запомненный код и есть искомая длительность временного интервала, выраженная в числе периодов частоты тактирования счетчика временной базы.

При втором способе счетчик временной базы считает непрерывно, он никогда принудительно не обнуляется. В момент первого изменения сигнала подсистема входного захвата запоминает текущее значение кода счетчика временной базы в регистре и генерирует запрос на прерывание. В подпрограмме прерывания по событию входного захвата этот двухбайтовый код программно считывается и запоминается в оперативной памяти МК под именем Start. В этой же подпрограмме подсистема входного захвата перенастраивается на детектирование второго изменения сигнала. Когда это событие происходит, подсистема входного захвата опять запоминает новое текущее значение кода счетчика временной базы и генерирует запрос на прерывание. В подпрограмме прерывания новый код счетчика запоминается под именем Stop. Если в процессе слежения за сигналом переполнений счетчика временной базы не было, то искомая длительность временного интервала, выраженная в числе периодов частоты тактирования счетчика временной базы, определяется как (Stop – Start).

Опытный разработчик всегда использует второй способ. По отношению к первому способу он обладает двумя преимуществами:

- Необходимое в первом способе обнуление счетчика может вызвать нарушение правильной работы других каналов таймера, которые также используют для отсчета временных интервалов код счетчика временной базы. Второй способ не нарушает естественный порядок счета счетчика временной базы, и, следовательно, создает «комфортные» условия для работы оставшихся семи каналов модуля таймера.
- Второй способ обладает большей точностью. При первом способе момент первого изменения сигнала отмечается нулевым кодом счетчика, который будет установлен только после перехода к подпрограмме прерывания. Этот процесс может занять от 10 до 20 тактов f_{BUS}. При втором способе аппаратные средства подсистемы входного захвата фиксируют первое изменение сигнала аппаратно, и ошибка не будет составлять более одного такта f_{BUS}.

Рассмотрим более подробно вычисление реальной длительности измеряемого сигнала по второму способу. На рис. 4.29 показаны пять возможных ситуаций, в которых подсистемой входного захвата зафиксированы совершенно одинаковые коды начала и конца измеряемого временного интервала. В первом случае (рис. 4.29,а) код Stop превышает код Start, и переполнений счетчика временной базы не было. Тогда очевидно, что измеряемая длительность временного интервала TIME = Stop – Start. Во втором случае (рис. 4.29,б) код Stop меньше, чем код Start, и между изменениями сигнала было всего одно переполнение счетчика временной базы:

$$TIME = (2^{16} - Start) + Stop = 2^{16} + (Stop - Start)$$

Рассмотрев остальные случаи (рис. 4.29, в,г,д), можно убедиться, что в каждом из случаев расчет искомого временного интервала следует вести по формуле:

$$TIME = 2^{16} \times n + (Stop - Start),$$

где n – число переполнений счетчика временной базы, которое случилось между двумя событиями фиксации изменения сигнала подсистемой входного захвата.

Код TIME отражает длительность временного интервала в периодах частоты тактирования счетчика временной базы. Во многих прикладных задачах вычисление реальной длительности в миллисекундах или секундах не производится. Если же это необходимо, то МК должен выполнить дополнительную операцию умножения:

$$t_{IZM} = TIME \times (TCNTclock),$$

где $TCNTclock$ – период частоты тактирования счетчика временной базы.

Сброс счетчика временной базы

Как было отмечено выше, сброс счетчика временной базы крайне нежелателен, поскольку он нарушает естественный порядок счета и может привести к ошибкам в работе подсистем входного захвата или выходного сравнения IC/OC, которые в момент сброса реализуют предназначенные им функции с использованием изменяющегося кода счетчика временной базы. Однако если сброс все-таки необходим, то его можно реализовать следующим образом:

- Установить в 1 бит разрешения сброса счетчика TCRE (бит 3) в регистре управления TMSK2;
- Установить в $0000 регистр данных канала 7 модуля таймера.

Комбинация этих двух состояний будет удерживать счетчик временной базы в нулевом состоянии.

Вопросы для самопроверки

1. Какова частота тактирования МК на отладочной плате MC68HC912B32EVB?
   Ответ: Микроконтроллер B32, установленный на плате MC68HC912B32EVB тактируется от кварцевого резонатора с частотой 16 МГц. Эта частота делится внутренними средствами МК на 2, поэтому частота межмодульных магистралей составляет 8 МГц.
2. С какой целью Вам может понадобиться тактировать МК на с иной, отличной от f_{BUS} = 8 МГц частотой?
   Ответ: Энергия потребления МК в процессе работы пропорциональна частоте тактирования. Поэтому снижение частоты тактирования целесообразно с точки зрения уменьшения энергетических потерь. Ряд применений, связанных с электромеханическими нагрузками, не требует предельного быстродействия МК, поэтому частота тактирования может быть снижена.
3. Подсистема входного захвата зафиксировала два различных события в моменты времени, соответствующие кодам $0105 и $EC20 счетчика временной базы. Чему равен интервал времени между этими событиями? Интервал следует указать в единицах счета счетчика, число должно быть представлено в десятичном коде.
   Ответ: $EC20 – $0105 = $EB1B = 60187 тактов счетчика

4. Если в предыдущем примере частота на входе программируемого делителя составляет 2 МГц, и биты PR2...PR0 регистра TMSK2 установлены в 000, то каков интервал времени между событиями в мс?
   Ответ: Интервал составляет: 60187 × 1/(2МГц) = 30093,5 мс.
5. Повторите расчет предыдущего вопроса, но при измененных значениях битов PR2...PR0 = 100.
   Ответ: В предыдущем случае коэффициент деления программируемого делителя был равен 1. При указанных в данном примере значениях битов PR2...PR0 коэффициент деления составляет 16. Следовательно, интервал времени между событиями составит 30093,5 мс × 16 = 481496 мс.
6. Назовите три основных режима работы модуля таймера TIM?
   Ответ: режим входного захвата IC, режим выходного сравнения OC и режим счетчика внешних событий PA.
7. Почему необходимо следить за флагом переполнения счетчика временной базы?
   Ответ: При использовании модуля таймера для измерения временных интервалов необходимо знать, сколько раз в процессе измерения переполнился счетчик. В противном случае подсчитанный код будет неверным.
8. Почему разработчик должен быть крайне осторожен при обнулении счетчика временной базы таймера?
   Ответ: Сама по себе операция обнуления не представляет сложности. Однако к работающему счетчику временной базы могут быть «привязаны» другие каналы IC/OC. И изменение естественного порядка счета счетчика может привести к неправильной работе этих каналов. Поскольку каналов в модуле таймера восемь, то достаточно трудно предугадать все возможные комбинации их функционирования в реальной задаче. Поэтому во избежание сбоев в работе целесообразно отказаться от обнуления счетчика временной базы.

4.14.3. Регистры для управления счетчиком временной базы

В данном параграфе мы рассмотрим регистры специальных функций модуля таймера, которые используются для управления и определения текущего состояния счетчика временной базы.

Регистр управления модулем таймера

Регистр управления модулем таймера TSCR (Timer System Control Register) располагается в памяти МК по адресу $0086. Формат регистра представлен на рис. 4.30. Старший бит регистра TEN разрешает (при TEN=1) или запрещает (при TEN=0) функционирование модуля таймера. Этот бит используется разработчиками для включения или отключения модуля таймера в процессе работы устройства. Отключение модуля полезно с точки зрения снижения потребления энергии, если функции таймера не используются в алгоритме управления. Обратите внимание, что после сброса МК модуль таймера выключен, т.к. бит TEN автоматически устанавливается в 0 в состоянии сброса МК.

Бит TFFCA управляет механизмом сброса флагов событий модуля таймера. К этим флагам относятся флаг переполнения таймера TOF, флаги событий в каналах захвата/сравнения IC/OC и флаг переполнения счетчика внешних событий. Если бит TFFCA установлен в 0, то для сброса перечисленных флагов следует использовать обычную процедуру, когда в бит установленного флага под управлением прог-

Имя регистра: TSCR Адрес: $0086

7	6	5	4	3	2	1	0
TEN	TSWAI	TSBCK	TFFCA	0	0	0	0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.30. *Формат регистра TCSR*

раммы записывается 1 (см. раздел 4.14.2, флаг переполнения счетчика). Попытка записи 0 в бит установленного флага оставит флаг без изменения. Если бит TFFCA установлен в 1, то вводится в действие дополнительный механизм сброса рассматриваемых флагов событий:

- Флаг переполнения счетчика временной базы TOF в регистре TFLG2 будет сброшен автоматически при выполнении операции чтения регистра текущего кода счетчика TCNT;
- Флаг события канала захвата/сравнения CnF (n – номер канала) в регистре TFLG1 будет сброшен автоматически при выполнении операции чтения или записи в регистр данных этого канала;
- Флаги PAONF и PAIF счетчика внешних событий в регистре PAFLG будут сброшены автоматически при чтении или записи в регистр PACNT.

Приведенный ниже программный фрагмент демонстрирует код для разрешения работы модуля таймера.

```
/*--------------------------------------------------------------------*/
/*MAIN PROGRAMM                                                       */
/*--------------------------------------------------------------------*/

#include<912b32.h>

void main(void)
{
unsigned char TSCR_MASK = 0x80;  /*разрешение работы модуля таймера*/
TSCR = TSCR_MASK;
}
/*--------------------------------------------------------------------*/
```

Регистр счетчика временной базы

Если работа модуля таймера разрешена, то счетчик временной базы изменяет свое состояние в порядке естественного счета. Число разрядов счетчика временной базы равно 16, поэтому коэффициент счета счетчика составляет 2^{16}. Изменение коэффициента счета не предусмотрено. Счетчик временной базы невозможно остановить, однако его состояние в любой момент времени может быть считано под управлением программы из регистра текущего состояния TCNT (Timer CouNTer register). Регистр TCNT – 16-разрядный, в памяти МК он занимает две ячейки. По адресу $0084 располагается старший байт регистра TCNTH, по адресу $0085 – младший байт регистра TCNTL. Формат регистра представлен на рис. 4.31.

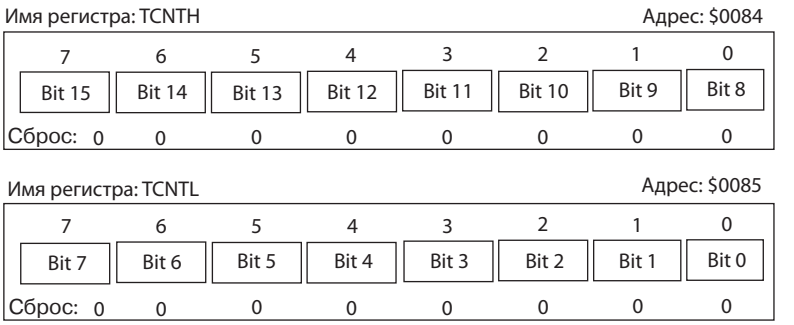


Рис. 4.31. *Формат регистра TCNT*

Считывание регистра текущего состояния счетчика временной базы следует производить только в 16-разрядном формате, поскольку побайтное чтение может дать неверный результат. Дело в том, что счетчик временной базы может тактироваться максимальной частотой, которая равна f_{BUS}. Каждая операция чтения занимает несколько тактов f_{BUS}, следовательно, две последовательных операции чтения приведут к тому, что старший и младший байты счетчика будут считаны при разных состояниях этого счетчика и вместе составят неверное его состояние (если режим «чтение на лету» не задействован). Приведенный ниже программный фрагмент демонстрирует, как правильно считать состояние счетчика с использованием двухбайтовых переменных.

```
/*-------------------------------------------------------------------*/
/*MAIN PROGRAMM                                                      */
/*-------------------------------------------------------------------*/

#include<912b32.h>

void main(void)
{
unsigned int      start_time;          /*время начала процесса*/
unsigned int      stop_time;           /*время окончания процесса*/


start_time=TCNT;



stop_time=TCNT;

}
/*-------------------------------------------------------------------*/
```

Регистр масок таймера 2

Регистр масок таймера TMSK2 (Timer MaSK register 2) располагается в памяти МК по адресу $008D. Формат регистра представлен на рис. 4.32. В данном параграфе мы рассмотрим лишь некоторые биты этого регистра. Бит TCRE разрешает сброс счетчика временной базы таймера (см. раздел 4.14.2, сброс счетчика временной базы). Биты PR2:PR1:PR0 устанавливают коэффициент деле-

Имя регистра: TMSK2 Адрес: $008D

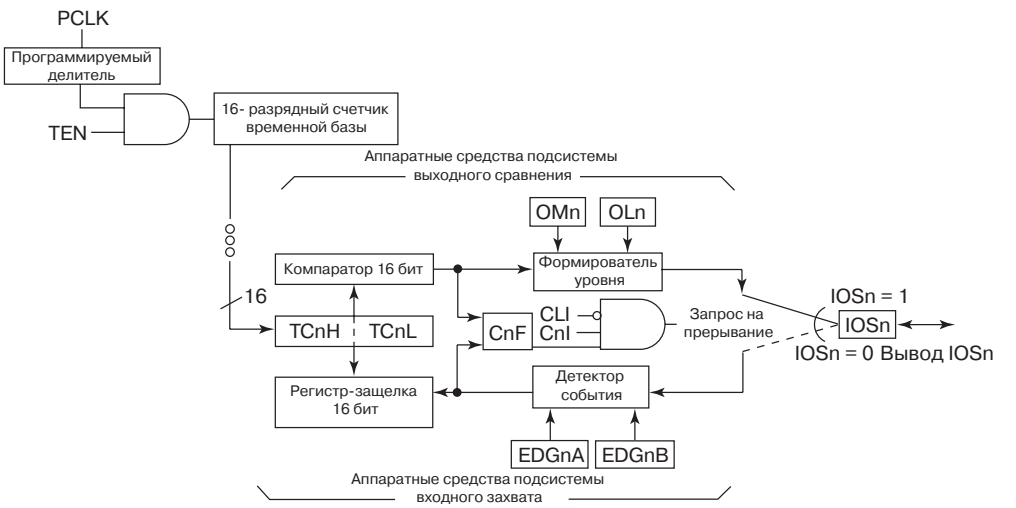
7	6	5	4	3	2	1	0
TOI	0	PUPT	RDPT	TCRE	PR2	PR1	PR0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.32. *Формат регистра TMSK2*

ния программируемого делителя на входе счетчика временной базы в соответствие с табл. рис. 4.28. Обратите внимание, что минимальный коэффициент деления равен 1, т.е. максимальная частота тактирования счетчика равна f_{BUS}. Максимальный коэффициент деления составляет 32. Биты PUPT и TDRB предназначены для управления схемотехникой входных и выходных буферов линий порта PORT T.

4.14.4. Каналы захвата/сравнения

Модуль таймера TIM содержит в себе восемь идентичных блоков захвата/сравнения, которые в микропроцессорной технике принято именовать каналами захвата/сравнения. Структурная схема аппаратных средств одного канала захвата/сравнения в составе модуля таймера МК семейства 68HC12/HCS12 представлена на рис. 4.33.



Имя регистра: TIOS Адрес: $0080

7	6	5	4	3	2	1	0
I/OS7	I/OS6	I/OS5	I/OS4	I/OS3	I/OS2	I/OS1	I/OS0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.33. *Структура одного канала сравнения/захвата таймера и регистр выбора режима работы каналов TIOS*

Каждый из восьми каналов захвата/сравнения подключен к выводу IOSn, где n – номер канала, n = 0, 1, 2...7. Если канал с номером n конфигурирован как канал захвата, то вывод IOSn автоматически подключается к одноименной линии PTn порта T. Работа в качестве входов подсистемы входного захвата или выходов подсистемы выходного сравнения является альтернативной функцией порта T. Регистр данных порта T расположен по адресу $00AE.

Реализуемая каналом n модуля таймера функция (входной захват или выходное сравнение) определяется битом IOCn регистра TIOC. Регистр расположен по адресу $0080, формат регистра представлен на рис. 4.33. Если бит IOCn установлен в 1, то канал n работает в режиме выходного сравнения. Если же бит IOCn равен 0, то канал n работает в режиме входного захвата.

Аппаратные средства каждого состоят из 16-разрядного регистра данных канала TCn, 16-разрядных регистра-защелки и цифрового компаратора, детектора события, формирователя выходного уровня и триггера события канала (рис. 4.33). Каждый канал использует в качестве эталона реального времени общий для всех каналов счетчик временной базы.

Режим входного захвата

Структура аппаратных средств подсистемы входного захвата IC, которая образуется в результате конфигурирования универсального канала модуля таймера на режим захвата, представлена на рис. 4.34. Подсистема входного захвата запоминает код счетчика временной базы в момент изменения логического сигнала на входе IOSn (n – номер канала). Изменение логического сигнала распознается детектором события, который может быть программно настроен на один из четырех режимов работы:

- Распознавание изменения сигнала с 0 на 1 – положительный фронт;
- Распознавание изменения сигнала с 1 на 0 – отрицательный фронт;
- Распознавание любого изменения уровня сигнала;
- Соответствующий вывод МК не подключен к каналу входного захвата и является выводом порта T.

Если детектор определил заданное изменение входного сигнала, то говорят, что наступило событие входного захвата. В момент наступления события код счетчика временной базы запоминается в регистре-защелке, одновременно устанавливается триггер события канал CnF (рис. 4.34). Триггер может быть считан программно, или генерируется запрос на прерывание, если прерывания от канала n модуля таймера разрешены.

Для настройки детектора события каждого канала на один из трех перечисленных режимов используются биты EDGnB:EDGnA в регистрах TCTL3 ($008A) и TCTL4 ($008B). Формат этих регистров представлен на рис. 4.34. Таблица рис. 4.35 устанавливает соответствие между режимом работы детектора события и кодом инициализации в разрядах EDGnB:EDGnA.

Подсистема входного захвата IC используемся в микропроцессорной технике для измерения различных временных характеристик импульсных сигналов, таких как период следования, коэффициент заполнения, длительность нулевого или единичного состояния. Так для того, чтобы измерить период импульсной последовательности, необходимо запомнить состояние счетчика временной базы в моменты двух соседних изменений сигнала с 0 на 1 (положительный фронт) или с 1 на 0 (отрицательный фронт). Разность этих значений и составит период повторения импульсного сигнала, выраженный в числе периодов частоты тактирования счетчика временной базы. Таким образом будет произведено измерение в относительных единицах конкретной микропроцессорной системы. Если измерение производится с целью управления, то предс-

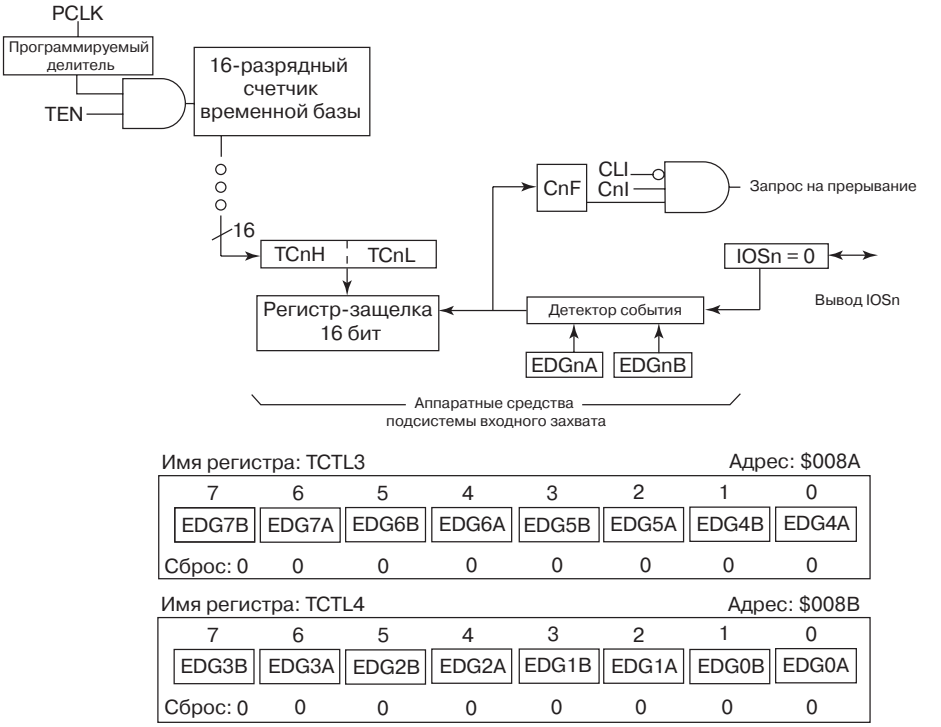


Рис. 4.34. *Структура одного канала таймера в режиме входного захвата и регистры управления каналом TCTL3 и TCTL4*

EDGnB: EDGnA	Режим детектора событий
00	Входной захват не реализуется
01	Мониторинг нарастающего фронта
10	Мониторинг спадающего фронта
11	Мониторинг изменения уровня

Рис. 4.35. *Выбор режима работы детектора события*

тавление временного параметра в относительных единицах обычно является достаточным. Однако, если измеренный параметр должен быть отображен на дисплее, то он должен быть представлен в универсальных единицах измерения, т.е. в микросекундах, миллисекундах и т.д. Для получения численного значения последнего необходимо полученное число относительных единиц умножить на длительность периода частоты тактирования счетчика временной базы. При программировании на Си операция умножения реализуется с использованием стандартной библиотеки. При программировании на ассемблере Вам потребуются дополнительные знания, поскольку операцию умножения необходимо будет исполнять над двухбайтовыми числами.

Если детектор события распознал изменение входного сигнала, которое указано в его текущей конфигурации, то аппаратные средства канала входного захвата автоматически совершают следующие действия:

1. Текущее состояние 16-разрядного счетчика временной базы запоминается в регистре-защелке канала и сразу копируется в 16-разрядный регистр данных канала TCn (n – номер канала). Поскольку данные в регистре TCn не изменяются, то они могут быть считаны побайтно (TCnH – старший байт регистра данных канала, TCnL – младший байт регистра данных), или в двухбайтовом формате. При программировании на Си рекомендуется использовать двухбайтовый формат, используя для этого переменную в формате unsigned integer.
2. Устанавливается флаг события канала CnF. Этот флаг «сообщает» основной программе о том, что событие произошло, и регистр данных канала TCn должен быть считан программой.
3. Если прерывания по флагу события CnF разрешены (бит CnI установлен), то генерируется запрос на прерывание.

Подсистема входного захвата может быть настроена для реализации разнообразных функций. Рассмотрим пример измерения длительности единичного состояния входного сигнала, т.е. импульса положительной полярности. Предположим, что длительность импульса не превышает периода переполнения счетчика временной базы таймера, исследуемая импульсная последовательность поступает на вход канала 2 модуля таймера, который предварительно настроен в режим входного захвата. Тогда для измерения длительности импульса должна быть реализована следующая последовательность действий:

1. Управляющая программа устанавливает режим работы детектора событий канала 2. Должен быть выбран режим мониторинга нарастающего фронта входного сигнала. Для этого следует установить биты EDG2B:EDG2A в регистрах TCTL3:TCTL4 в состояние 01.
2. Управляющая программа контролирует состояние триггера события C2F.
3. Если триггер C2F установился в 1, то контролируемый сигнал на линии PT2 изменился с 0 на 1. В момент изменения код счетчика временной базы был автоматически переписан в регистр-защелку канала 2. Теперь этот код доступен для чтения из регистра данных TC2.
4. Управляющая программа обнаруживает, что триггер C2F установился. Тогда программа считывает двухбайтовый код из регистра данных канала TC2 и записывает его в двухбайтовую беззнаковую переменную rising_edge.
5. Управляющая программа сбрасывает триггер события C2F посредством записи в бит C2F единицы.
6. Управляющая программа изменяет режим работы детектора событий канала 2. Должен быть выбран режим мониторинга отрицательного фронта входного сигнала. Для этого следует установить биты EDG2B:EDG2A в регистрах TCTL3:TCTL4 в состояние 10.
7. Управляющая программа контролирует состояние триггера события C2F.
8. Если триггер C2F установился в 1, то сигнал на линии PT2 изменился с 1 на 0. В момент изменения код счетчика временной базы был опять автоматически переписан в регистр-защелку канала 2.
9. Управляющая программа обнаруживает установленный триггер C2F, считывает двухбайтовый код из регистра данных канала TC2 и записывает его в двухбайтовую беззнаковую переменную falling_edge.

10. Управляющая программа сбрасывает триггер события C2F посредством записи в бит C2F единицы.
11. Управляющая программа вычисляет число периодов частоты тактирования счетчика временной базы между положительным и отрицательным фронтами исследуемого сигнала: TIME = falling_edge – rising_edge. Это число и есть искомая длительность импульса положительной полярности сигнала на входе PT2, выраженная числом периодов частоты тактирования счетчика временной базы.
12. При необходимости длительность импульса может быть представлена в общепринятых единицах измерения времени. Для этого управляющая программа должна выполнить операцию умножения числа TIME на длительность единицы измерения времени таймера, т.е. на длительность периода частоты тактирования счетчика временной базы: $t_{IZM} = TIME \times 1/f_{BASE}$.

    Регистры специальных функций, связанные с подсистемой входного захвата, мы обсудим позже, после рассмотрения подсистемы выходного сравнения.

Вопросы для самопроверки

1. Какие изменения необходимо внести в рассмотренную выше последовательность действий, чтобы произвести измерение длительности нулевого состояния сигнала на входе PT2?

   Ответ: В п.1. следует изменить инициализацию режима работы детектора событий. Должен быть выбран режим мониторинга отрицательного фронта входного сигнала. Также следует внести изменение в п. 6, в котором для детектора событий следует выбрать режим мониторинга нарастающего фронта входного сигнала.
2. Какие изменения необходимо внести для измерения периода повторяемости импульсного сигнала на входе PT2?

   Ответ: Рассматриваем изменения исходной последовательности действий, которая приведена в предыдущем параграфе. Пункт 6 последовательности необходимо исключить. Тогда триггер события C2F будет в обоих случаях устанавливаться по положительному фронту исследуемого сигнала. Следовательно, в п.п. 11 и 12 будет вычисляться интервал времени между соседними положительным фронтами сигнала, т.е. период повторяемости этого сигнала.
3. Какие дополнительные изменения по отношению к вопросу 2 необходимо внести для измерения периода повторяемости импульсного сигнала, длительность которого превышает период переполнения счетчика временной базы?

   Ответ: Необходимо организовать программный счетчик и вести наблюдение не только за состоянием триггера события канала входного захвата, но и триггера переполнения счетчика временной базы. Если триггер TOF установился. То следует программно инкрементировать программный счетчик. Также следует изменить формулу для подсчета длительности периода.

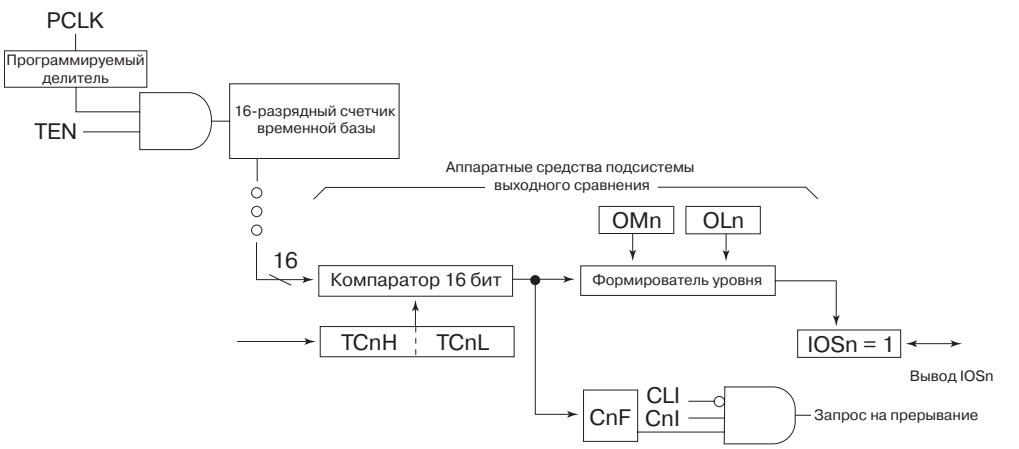
Режим выходного сравнения

Подсистема выходного сравнения ОС используемся в микропроцессорной технике для генерации на выводах МК импульсных сигналов с заданными временными характеристиками. Например, средствами подсистемы выходного сравнения может быть сформирован одиночный импульс предварительно вычисленной длительности, или импульсная последовательность определенной частоты с регулируемым по результатам расчетов в МК коэффициентом заполнения. Структура аппаратных средств подсистемы выходного сравнения ОС, которая образуется после конфигурирования уни-

версального канала модуля таймера на режим сравнения, представлена на рис. 4.36.

Цифровой компаратор подсистемы выходного сравнения непрерывно сравнивает код счетчика временной базы с 16-разрядным кодом в регистре данных канала TCn (n – номер канала). Момент равенства кодов в микропроцессорной технике называют событием выходного сравнения. Если цифровой компаратор определил равенство кодов, то аппаратные средства канала выходного сравнения автоматически совершают следующие действия:

1. Устанавливается флаг события канала CnF. Обратите внимание, события входного захвата и выходного сравнения отмечаются одним и тем же флагом. Это флаг события канала CnF. Смысловое значение флага (IC или OC) определяется ранее выбранным в процессе инициализации режимом работы канала. Флаг CnF «сообщает» основной программе о том, что событие выходного сравнения произошло, и в регистр данных канала TCn следует записать новое значение кода для сравнения.
2. Если прерывания по флагу события CnF разрешены (бит CnI установлен), то генерируется запрос на прерывание.
3. Формирователь уровня генерирует на выводе канала IOSn предварительно заданный логический уровень. Формирователь уровня в процессе инициализации может быть программно настроен на один из четырех режимов работы:



Имя регистра: TCTL1 Адрес: $0088

	7	6	5	4	3	2	1	0
	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: TCTL2 Адрес: $0089

	7	6	5	4	3	2	1	0
	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.36. *Структура одного канала таймера в режиме выходного сравнения и регистры управления каналом TCTL1 и TCTL2*

OMn: OLn	Режим формирователя уровня
00	Выход формирователя не соединен с выводом IOSn
01	Инвертирует сигнал на выходе
10	Устанавливает выход в 0
11	Устанавливает выход в 1

Рис. 4.37. *Выбор режима работы формирователя уровня*

- Установка вывода в 1;
- Установка вывода в 0;
- Инвертирование уровня сигнала на выводе;
- Соответствующий вывод МК не подключен к каналу выходного сравнения и является выводом порта T.

Для настройки формирователя уровня каждого канала на один из трех перечисленных режимов работы используются биты OMn:OLn в регистрах TCTL1 ($0088) и TCTL2 ($0089). Формат этих регистров представлен на рис. 4.36. Таблица рис. 4.37 устанавливает соответствие между режимом работы формирователя уровня и кодом инициализации в разрядах OMn:OLn.

Рассмотрим последовательность действий, которая должна быть реализована для генерации средствами подсистемы выходного сравнения «отрицательного» импульса заданной длительности. Под «отрицательным» импульсом мы будем понимать низкий логический уровень сигнала с последующим его изменением на высокий логический уровень.

1. Управляющая программа считывает текущее состояние счетчика временной базы из двухбайтового регистра TCNT.
2. Управляющая программа устанавливает выход используемого канала в 0.
3. Управляющая программа вычисляет длительность временного интервала, в течение которого на выходе должен удерживаться низкий логический уровень. Обратите внимание, полученная длительность временного интервала должна быть представлена в числе периодов тактирования счетчика временной базы.
4. Управляющая программа производит сложение ранее считанного кода счетчика и полученного кода длительности временного интервала. Полученное значение записывается в регистр данных канала выходного сравнения.
5. Управляющая программа задает режим работы формирователя уровня канала. Целесообразно выбрать режим установки выхода в 1, для чего необходимо записать в разряды OMn:OLn регистров TCTL1: TCTL2 комбинацию 11 (см. рис. 4.37).
6. Когда значение кода регистра данных совпадет с кодом счетчика временной базы, на выходе канала выходного сравнения автоматически, без участия программ установится 1.

Канал 7 в режиме выходного сравнения

До настоящего параграфа мы рассматривали все восемь каналов модуля таймера TIM, как полностью идентичные. Однако на самом деле полностью идентичными, работающими или в режиме входного захвата или в режиме выходного сравнения, являются лишь каналы 0...6. Канал 7 в режиме выходного сравнения обладает дополнительными возможностями. При этом рассмотренные режимы захвата/сравнения, совпадающие с режимами каналов 0...6, при соответствующей инициализации им полностью поддерживаются.

Если канал 7 находится в режиме выходного сравнения и его дополнительная функция разрешена, то в момент события в канале 7 устанавливается в назначенное состояние не только выход канала 7, но и любая комбинация выходов каналов 0...6. При этом выбранные для принудительной установки каналы также должны работать в режиме выходного сравнения. Два регистра специальных функций OC7M и OC7D используются для управления режимом принудительной установки по событию в канале 7. Каждый из битов регистра OC7M разрешает (при OC7Mn = 1) режим принудительной установки для одноименного (с номером n) канала. Соответствующий бит в регистре OC7D задает значение, которое должно быть установлено на выходе канала в момент принудительной установки. Например, если в регистре OC7M записан двоичный код 10110001, а в регистре OC7D – 01010101, то с каналом 7 в режиме принудительной установки будут связаны каналы 0, 4, 5. Тогда в момент события выходного сравнения на выходе канала 0 установится 1, на выходе канала 4 установится также 1, на выходе канала 5 установится 0 и на выходе канала 7 – тоже 0. На остальные каналы событие выходного сравнения в канале 7 влияния не окажет.

Свойство принудительной установки позволяет два раза за период работы счетчика временной базы изменять состояние на выходах каналов 0...6. Один раз по собственному событию выходного сравнения канала, второй раз – по событию в канале 7.

Регистры для управления каналами захвата/сравнения

В данном разделе мы рассмотрим регистры специальных функций, которые используются для управления режимами работы универсальных каналов захвата/сравнения. Формат регистров управления счетчиком временной базы в составе модуля таймера был рассмотрены ранее.

Регистр режимов каналов захвата/сравнения

Регистр режимов каналов захвата/сравнения TIOS (Timer Input capture/Output compare Register) располагается в памяти МК по адресу $0080. Формат регистра представлен на рис. 4.38. Каждый из битов регистра IOSn определяет режим работы канала с номером n. Если бит установлен в 1, то канал работает в режиме выходного сравнения. Если бит равен 0, то канал настроен на режим входного захвата.

Регистр принудительного события выходного сравнения

Регистр принудительного события выходного сравнения CFORC (Timer Compare Force Register) располагается в памяти МК по адресу $0081. Формат регистра представлен на рис. 4.38. Установка бита FOCn в 1 немедленно вызывает событие выходного сравнения в канале с номером n, независимо от текущего состояния счетчика временной базы и регистра данных канала. Функция принудительной установки очень удобна для задания начального уровня выхода канала выходного сравнения.

Регистр разрешения работы под управлением канала 7

Регистр разрешения работы под управлением канала 7 OC7M (Timer Output Compare 7 Mask Register) располагается в памяти МК по адресу $0082. Формат регистра представлен на рис. 4.38. Каждый из битов регистра OC7M разрешает (при

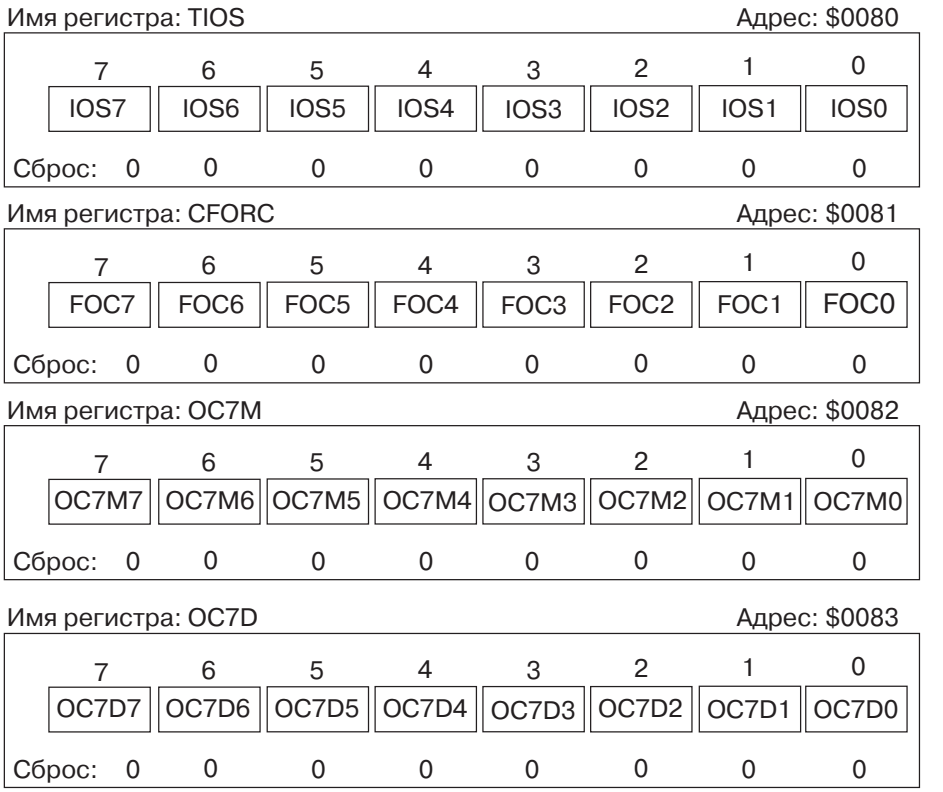


Рис. 4.38. *Формат дополнительных регистров управления таймером: TIOS, CFORC, OC7M, OC7D*

OC7Mn = 1) режим принудительной установки для одноименного (с номером n) канала по событию в канале выходного сравнения 7. Если бит OC7Mn = 0, то канал с номером n работает в автономном режиме.

Регистр установки данных под управлением канала 7

Регистр установки данных под управлением канала 7 OC7D (Timer Output Compare 7 Data Register) располагается в памяти МК по адресу $0083. Формат регистра представлен на рис. 4.38. Каждый из битов регистра OC7Dn задает значение, которое должно быть установлено на выходе канала с номером n в момент в момент события выходного сравнения в канале 7. При этом необходимо предварительно разрешить работу желаемых каналов под управлением канала 7 установкой соответствующих битов в регистре OC7M.

Регистры управления таймером 1 и 2

Регистры управления таймером TCTL1 и TCTL2 (Timer Control Register) располагаются в памяти МК по адресам $0088 и $0089. Форматы регистров приведены на рис. 4.39. Каждому каналу таймера поставлены в соответствие два бита OMn:OLn регистров TCTL1 и TCTL2. Биты OMn:OLn определяют один из четырех режимов работы формирователя уровня канала, если этот канал работает в режиме выходного сравнения. Комбинации кодов для битов OMn:OLn приведены на рис. 4.37.

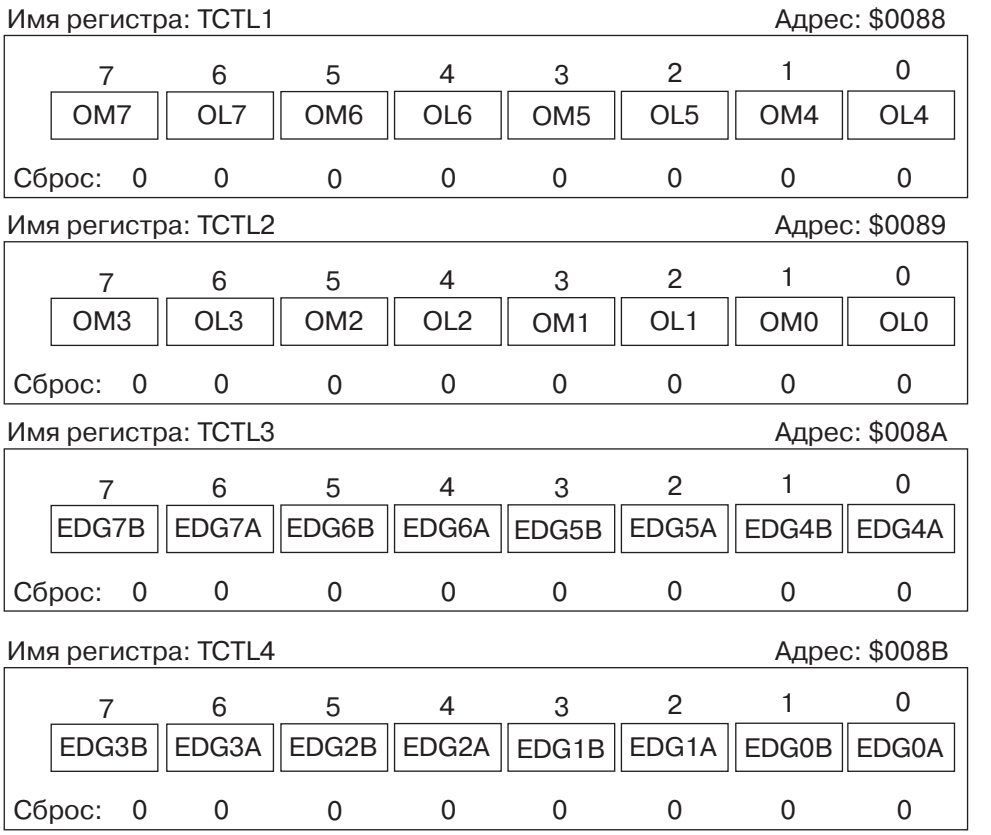


Рис. 4.39. *Формат основных регистров управления таймером:TCTL1, TCTL2, TCTL3, TCTL4*

Регистры управления таймером 3 и 4

Регистры управления таймером TCTL3 и TCTL4 (Timer Control Register) располагаются в памяти МК по адресам $008A и $008B. Форматы регистров приведены на рис. 4.39. Каждому каналу таймера поставлены в соответствие два бита EDGnB:EDGnA регистров TCTL3 и TCTL4. Биты EDGnB:EDGnA определяют один из четырех режимов работы детектора собвтия канала, если этот канал работает в режиме входного захвата. Комбинации кодов для битов EDGnB:EDGnA приведены на рис. 4.35.

Регистр масок таймера 1

Регистр масок таймера TMSK1 (Timer Mask Register 1) располагается в памяти МК по адресу $008C. Формат регистра приведен на рис. 4.40. Каждый бит этого регистра CnI разрешает или запрещает прерывания по событию в одноименном (с номером n) канале таймера. Если бит CnI равен 1, то прерывания разрешены. При CnI = 0 прерывания по событию в канале запрещены.

Регистр масок таймера 2

Регистр масок таймера TMSK2 (Timer Mask Register 2) располагается в памяти МК по адресу $008D. Формат регистра приведен на рис. 4.40. Бит TOI разрешает прерывания по флагу переполнения счетчика временной базы TOF. Бит TCRE раз-

Имя регистра: TMSK1 Адрес: $008C

	7	6	5	4	3	2	1	0
	C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: TMSK2 Адрес: $008D

	7	6	5	4	3	2	1	0
	TOI	0	PUPT	RDPT	TCRE	PR2	PR1	PR0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.40. *Формат регистров масок таймера:TMSK1, TMSK2*

решает сброс счетчика временной базы (см. раздел 4.14.2, сброс счетчика временной базы). Биты PR2:PR1:PR0 устанавливают коэффициент деления программируемого делителя на входе счетчика временной базы в соответствие с табл. рис. 4.28.

Регистр флагов таймера 1

Регистр флагов таймера TFLG1 (Timer Flag Register 1) располагается в памяти МК по адресу $008E. Формат регистра представлен на рис. 4.41. Каждому каналу таймера поставлен в соответствие флаг события CnF. Флаг CnF устанавливается в 1 автоматически, если в канале произошло событие входного захвата или выходного сравнения, в зависимости от текущего режима работы канала. Установленный бит события CnF вызовет прерывание, если в регистре TMSK1 установлен одноименный бит разрешения прерывания. Флаг события CnF должен быть сброшен под управлением программы, для чего в бит CnF должна быть записана 1. Существует альтернативный способ для сброса флагов события CnF. Если бит TFFCA в регистре TSCR установлен, то чтение или запись в регистр данных канала автоматически сбрасывает бит события этого канала.

Регистр флагов таймера 2

Регистр флагов таймера TFLG2 (Timer Flag Register 2) располагается в памяти МК по адресу $008F. Формат регистра представлен на рис. 4.41. В регистре присутствует всего один флаг – флаг переполнения счетчика временной базы TOF. Этот флаг сбрасывается посредством записи 1 в уже установленный бит TOF.

Имя регистра: TFLG1 Адрес: $008E

	7	6	5	4	3	2	1	0
	C7F	C6F	C5F	C4F	C3F	C2F	C1F	C0F
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: TFLG2 Адрес: $008F

	7	6	5	4	3	2	1	0
	TOF	0	0	0	0	0	0	0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.41. *Формат регистров флагов таймера:TFLG1, TFLG2*

Регистр данных канала 0: TC0H:TC0L	Адрес: $0090-0091
Регистр данных канала 1: TC1H:TC1L	Адрес: $0092-0093
Регистр данных канала 2: TC2H:TC2L	Адрес: $0094-0095
Регистр данных канала 3: TC3H:TC3L	Адрес: $0096-0097
Регистр данных канала 4: TC4H:TC4L	Адрес: $0098-0099
Регистр данных канала 5: TC5H:TC5L	Адрес: $009A-009B
Регистр данных канала 6: TC6H:TC6L	Адрес: $009C-009D
Регистр данных канала 7: TC7H:TC7L	Адрес: $009E-009F

Имя регистра: TCnH Адрес:

7	6	5	4	3	2	1	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Сброс: 0	0	0	0	0	0	0	0

Имя регистра: TCnL Адрес:

7	6	5	4	3	2	1	0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.42. *Формат регистров данных таймера:TCnH, TCnL*

Регистры данных каналов захвата/сравнения

Регистры данных каналов захвата/сравнения TCn – 16-разрядные. В памяти каждый регистр представлен двумя 8-разрядными регистрами: TCnH – старший байт регистра данных канала с номером n, TCnL – младший байт регистра данных канала с номером n. Если канал настроен на режим входного захвата, то в регистре данных TCn содержится код счетчика временной базы в момент последнего события входного захвата. Если же канал настроен на режим выходного сравнения, то в регистр данных TCn под управлением программы записывается код момента сравнения. Формат и адреса расположения в памяти восьми регистров данных каналов TC0...TC7 приведены на рис. 4.42.

Вопросы для самопроверки

1. Опишите два возможных способа для сброса флага события в регистре TFLG1.
   Ответ: По первому способу биты флагов событий в каналах захвата/сравнения CnF в регистре TFLG1 сбрасываются посредством записи 1 в тот разряд регистра TFLG1B, который подлежит сбросу. Попытка записи 0 в разряд, который установлен в 1, не даст желаемого результата. п.1. По второму способу чтение или запись в регистр данных канала автоматически сбрасывает бит события этого канала, если в регистре TSCR установлен бит TFFCA.
2. Какой код должен быть записан в регистр режимов каналов захвата/сравнения TIOS, чтобы каналы с четными номерами работали в режиме входного захвата, а каналы с нечетными номерами в режиме выходного сравнения?
   Ответ: $AA.

3. Какой код и в какие разряды регистра TCTL1 должен быть записан, чтобы настроить формирователь уровня канала 7 в режим установки 1?
   Ответ: Код 11 в разряды OM7:OL7.
4. Какой код и в какие разряды регистра TCTL4 должен быть записан, чтобы настроить детектор события канала 1 в режим мониторинга за любым изменением уровня сигнала на входе канала 1?
   Ответ: Код 11 в разряды EDG1B:EDG1A.

Примеры работы с таймером

Познакомившись с основными подсистемами модуля таймера, мы рассмотрим несколько примеров применения. В первом примере мы будем использовать подсистему входного захвата для измерения частоты и периода следования импульсов некоторого логического сигнала. Во втором примере мы будем формировать на одном из выходов МК импульсную последовательность, используя для этого подсистему выходного сравнения и подсистему прерывания МК.

Измерение частоты и периода логического сигнала

Для того чтобы воспользоваться аппаратными средствами универсального канала таймера для измерения частоты и периода следования импульсов, необходимо установить этот канал в режим входного захвата, а также выполнить ряд дополнительных установок конфигурации для счетчика временной базы, детектора события канала и подсистемы прерывания. Полное описание регистров специальных функций модуля таймера было приведено выше. В нашем примере мы будем использовать следующие биты и регистры управления:

- Бит разрешения работы модуля таймера TEN (регистр управления модулем таймера TSCR);
- Бит разрешения прерывания по переполнению счетчика временной базы TOI и биты выбора коэффициента деления программируемого делителя частоты на входе счетчика временной базы PR2:PR1:PR0 (регистр масок таймера TMSK2);
- Бит выбора режима работы канала IOSn (регистр режимов каналов захвата/сравнения TIOS). Если бит IOSn установлен в 1, то канал работает в режиме выходного сравнения. Если бит IOSn равен 0, то канал настроен на режим входного захвата;
- Биты выбора режима работы детектора события канала входного захвата EDGnB:EDGnA (регистры управления таймером TCTL3 и TCTL4);
- Бит события в канале CnF (регистр флагов таймера TFLG1);
- Бит разрешения прерывания по событию в канале CnI (регистр масок таймера TMSK1);
- Регистр данных канала TCn, в который автоматически записывается код счетчика временной базы в момент события входного захвата.

Предположим, что период исследуемого сигнала не превышает длительности периода переполнения счетчика временной базы. Тогда, для измерения частоты и периода следования импульсов логического сигнала, необходимо реализовать в микроконтроллере следующую последовательность действий:

1. Разрешить работу модуля таймера;
2. Выбрать частоту тактирования счетчика временной базы, для чего установить желаемый коэффициент деления программируемого делителя частоты на входе счетчика временной базы;
3. Установить один из каналов в режим входного захвата по нарастающему фронту импульса;

4. Сохранить в памяти МК код счетчика временной базы в момент появления первого фронта импульса;
5. Сохранить в памяти МК код счетчика временной базы в момент появления второго фронта импульса;
6. Взять разность полученных кодов, которая будет равна периоду исследуемого сигнала. Воспользовавшись операцией деления, вычислить частоту исследуемого сигнала.

На рис. 4.43 приведена блок-схема рассмотренного алгоритма. Ниже приведен программный фрагмент (timerl.c), который реализует этот алгоритм. В программе применен метод программного опроса триггера события входного захвата C2F. Когда программа «обнаруживает» установленный в 1 триггер события первый раз, она копирует регистр данных канала в переменную rising_l, сбрасывает триггер и ожидает следующей установки триггера события. Когда триггер события установится второй

Рис. 4.43. *Блок-схема алгоритма измерения периода и частоты исследуемого сигнала*

раз, программа копирует регистр данных канала в переменную rising_2. Разность двух зафиксированных в регистре данных канала входного захвата значений позволит вычислить период, а затем и частоту исследуемого импульсного сигнала.

Отметим три момента в стиле написания исходного текста представленного программного фрагмента для измерения периода и частоты импульсного сигнала:

- Основная программа «main» содержит вызовы трех функций, каждая из которых выполняет отдельную смысловую задачу;
- Имена функций и переменных выбраны в соответствии с их смысловым назначением;
- Представленный исходный текст программы достаточно полно документирован посредством комментариев перед записью каждой функции.

```
/*---------------------------------------------------------------------------*/
/* filename: timer1.c                                                        */
/* MAIN PROGRAM: Эта программа измеряет период и частоту импульсного */
/* сигнала. Сигнал подключается на вход 2 подсистемы таймера (IC2).   */
/*---------------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>
#include<stdio.h>
#include<math.h>

/*используемые функции*/
void timer_init(void);
void measure_wave(void);
void period_freq(void);

/*глобальные переменные*/
unsigned long int rising_1;
unsigned long int rising_2;

void main(void)
{
timer_init();              /*инициализация таймера*/
measure_wave();            /*определение времени двух фронтов сигнала*/
period_freq();             /*вычисление периода и частоты*/
}
/*---------------------------------------------------------------------------*/
/* Функция timer_init производит инициализацию модуля таймера          */
/* Канал 2 таймера настраивается на режим входного захвата по переднему */
/* фронту сигнала                                                      */
/* Частота тактирования счетчика временной базы устанавливается 2 МГц. */
/*---------------------------------------------------------------------------*/
void timer_init(void)
{
TMSK1 = 0x00;       /*запретить прерывания от каналов таймера*/
TMSK2 = 0x02;       /*назначить коэффициент деления 4*/
TI0S = 0x00;        /*установить канал 2 в режим входного захвата*/
TSCR = 0x80;        /*разрешить работу таймера*/
TCTL4 = 0xl0;       /*назначить режим детектора событий по*/
                    /*положительному фронту*/
TFLGl = 0xFF;       /*очистить флаги событий*/
}
```

```
/*-------------------------------------------------------------------*/
/* Функция measure_wave запоминает два последовательных момента     */
/* нарастающего фронта исследуемого сигнала. Значения запоминаются с */
/* использованием глобальных переменных                              */
/*-------------------------------------------------------------------*/
void measure_wave(void)
{
while((TFLGl & 0x04) = = 0)
       {                    /*ожидать нарастающего фронта*/
       ;
       }
rising_l = TCNT;            /*запомнить код счетчика временной базы*/
                            /*в переменной rising_l*/
TFLGl = 0x04;               /*сброс триггера события канала 2*/

while((TFLGl & OxO4)= = 0)
       {                    /*ожидать нарастающего фронта*/
       ;
       }
rising_2 = TCNT;            /*запомнить код счетчика временной базы*/
                            /*в переменной rising_2*/
TFLGl = 0x04;               /*сброс триггера события канала 2*/
}

/*-------------------------------------------------------------------*/
/* Функция period_freq вычисляет период и частоту исследуемого импульсного*/
/* сигнала и отображает полученные значения на экране                */
/*-------------------------------------------------------------------*/
void period_freq (void)
{
unsigned long int new_rising;
unsigned long int new_falling;
float frequency;
float period;
unsigned int int_period;
unsigned int int_freq, freq_tenths;

if(rising_2 < rising_l)                 /*вычисление периода*/
       {
new_rising = rising_2 + 0xFFFF;
period = ((float)new_rising - (float)rising_l)*0.0000005;
       }
else
       {
period = ((float)rising_2 - (float)rising_l)*0.0000005;
       }

frequency = 1.0/period;                 /*вычисление частоты*/
int_freq = (int)(frequency/l000.0);
freq_tenths = (int)((frequency - (float) int_freq*1000)/100.0);

                                        /*вывод результатов*/
printf("Frequency = %d.%d kHz \n\n" int_freq, freq_tenths);

int_period = (int) (1000000*period);
```

```
printf("Period = %d µs \ n\n", int_period);
printf{"Period = %f ms \ n\n", (period*1000))};
}
/*-------------------------------------------------------------------*/
```

Приведенная программа выдаст ошибочный результат для сигналов, период которых превышает период переполнения счетчика внешних событий. В одном из домашних заданий в конце данной главы мы попросим Вас изменить исходный текст программы так, чтобы измерение более «медленных» сигналов также стало возможным. Какие изменения в программу следует внести? Вы должны будете контролировать, сколько раз переполнился счетчик временной базы между двумя соседними событиями в канале входного захвата. Для этого следует организовать программный счетчик, который будет инкрементироваться каждый раз, когда счетчик переполнится. Переполнение счетчика Вы будете фиксировать по установленному флагу TOF. Этот флаг может программно считываться с последующим сбросом, или по флагу могут быть разрешены прерывания. В подпрограмме прерывания будет инкрементироваться программный счетчик. После того, как второй нарастающий фронт зафиксирован, программа должна выполнить вычисления, используя формулу:

$$Period = 2^{16} \times n + (ri\sin g\_2 - ri\sin g\_1),$$

Рассматриваемая программа имеет также ограничение по измерению сигналов с достаточно высокой частотой. Как узнать максимальную частоту, которая может быть измерена? Для этого следует вспомнить, что в нашем учебном примере частота внутренней шины МК составляет 8 МГц. Вы должны просмотреть ассемблерный код функции измерения частоты measure_wave и определить, сколько машинных тактов необходимо для распознавания установленного в 1 флага события и считывания кода первого события из регистра данных канала. Именно этот интервал является минимальным периодом сигнала, который может быть измерен.

Генерация импульсной последовательности

Для того, чтобы сформировать последовательность импульсов на одном из выходов МК, следует воспользоваться подсистемой выходного сравнения. В данном примере мы рассмотрим технику использования одного из каналов таймера в режиме выходного сравнения для генерации импульсной последовательности с заданной частотой и коэффициентом заполнения. В примере мы будем использовать следующие биты и регистры управления:

- Бит разрешения работы модуля таймера TEN (регистр управления модулем таймера TSCR);
- Бит разрешения прерывания по переполнению счетчика временной базы TOI и биты выбора коэффициента деления программируемого делителя частоты на входе счетчика временной базы PR2:PR1:PR0 (регистр масок таймера TMSK2);
- Бит выбора режима работы канала IOSn (регистр режимов каналов захвата/сравнения TIOS). Если бит IOSn установлен в 1, то канал работает в режиме выходного сравнения. Если бит IOSn равен 0, то канал настроен на режим входного захвата;

- Биты выбора режима работы формирователя уровня канала выходного сравнения OMn:OLn (регистры управления таймером TCTL1 и TCTL2);
- Бит события в канала CnF (регистр флагов таймера TFLG1);
- Бит разрешения прерывания по событию в канале выходного сравнения CnI (регистр масок таймера TMSK1);
- Регистр данных канала TCn, код которого автоматически сравнивается с кодом счетчика временной базы. Момент равенства кодов и является событием выходного сравнения.

В нашем примере мы будем формировать импульсную последовательность, параметры которой определяются числом и годом рождения разработчика. Допустим, Вы родились 19 мая 1977 года. Тогда частота генерируемого сигнала будет равна 519 Гц, коэффициент заполнения будет равен 77%. Для задания численных констант, определяющих частоту и коэффициент заполнения генерируемого импульсного сигнала, проведем следующие несложные расчеты:

1. Назначим в качестве источника тактирования для счетчика временной базы программируемый делитель, на вход которого подается f_{BUS} = 8 МГц;
2. Установим коэффициент деления программируемого делителя частоты на входе счетчика временной базы, равный 4. Тогда частота тактирования счетчика будет равна 2 МГц, что соответствует разрешающей способности счетчика 0,5 мкс. Выбранная таким образом разрешающая способность должна быть достаточна для формирования периода следования и длительности импульса;
3. По условию задачи частота формируемого сигнала составляет 519 Гц, что соответствует периоду следования T = 1/519 = 0.0019268 с;
4. Вычислим длительность высокого и низкого уровня сигнала на интервале периода. По условию задачи коэффициент заполнения равен 77%. Длительность искомых временных интервалов составляет:

   Длительность _1 = 0.77*(0.0019268) = 0.001484 с
   Длительность _0 = 0.23*(0.0019268) = 0.0004432 с

5. Преобразуем полученные временные интервалы в целое число периодов частоты тактирования счетчика временной базы:

   Код_1 = 0.001484 с/0.5 мкс = 2968 тактов
   Код_0 = 0.0004432 с /0.5 мкс = 886 тактов

На рис. 4.44 приведена упрощенная блок-схема алгоритма генерации импульсной последовательности. Ниже представлен исходный текст программы на Си (timer2.c), в котором используется метод программного опроса триггера события в канале выходного сравнения.

```
/*-------------------------------------------------------------------*/
/* filename: timer2.c                                                */
/* MAIN PROGRAM: Эта программа генерирует импульсную последовательность */
/* с частотой 519 Гц и коэффициентом заполнения 77%. Сигнал формируется на*/
/* выходе 2 подсистемы таймера (IC2)                                 */
/*-------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>
```

```
/*используемые функции*/
void timer_init(void);
void half_cycle(unsigned int time);

void main(void)
{
unsigned int high_time = 2968;   /*число тактов высокого уровня*/
```

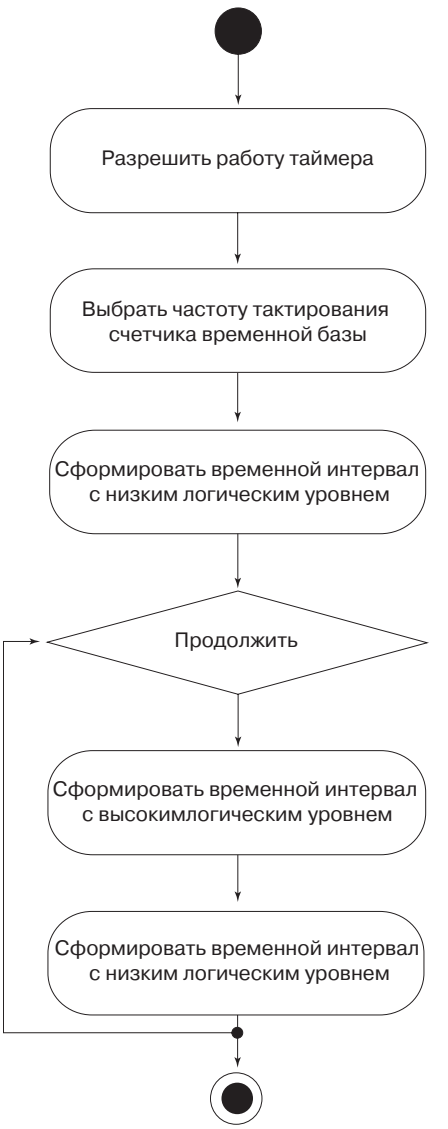


Рис. 4.44. *Блок-схема алгоритма генерации импульсной последовательности с заданными временными параметрами*

```
unsigned int low_time = 886;      /*число тактов низкого уровня*/

timer_init();
half_cycle(low_time);             /*генерация низкого уровня*/

while (1)
{
half_cycle(high_time);            /*генерация высокого уровня*/
half_cycle(low_time);             /*генерация низкого уровня*/
}
}

/*-------------------------------------------------------------------*/
/* Функция timer_init производит инициализацию модуля таймера.      */
/* Канал 2 таймера настраивается на режим выходного сравнения.      */
/* Частота тактирования счетчика временной базы устанавливается 2 МГц.*/
/*-------------------------------------------------------------------*/
void timer_init(void)
{
TMSK1 = 0x00;        /*запретить прерывания от каналов таймера*/
TMSK2 = 0x02;        /*назначить коэффициент деления 4*/
TI0S = 0x04;         /*установить канал 2 в режим выходного сравнения*/
TSCR = 0x80;         /*разрешить работу таймера*/
TCTL2 = 0x10;        /*назначить режим формирователя уровня */
                     /*"инвертирование"*/
TFLG1 = 0x04;        /*очистить флаг события канала 2*/
TC2 = TCNT           /*записать в регистр данных канала 2 текущее*/
                     /*состояние счетчика временной базы*/
}

/*-------------------------------------------------------------------*/
/* Функция half_cycle генерирует временной интервал заданной длительности */
/* Число тактов для отсчета должно быть определено вне функции      */
/*-------------------------------------------------------------------*/
void half_cycle(unsigned int time)
{
TC2 += time;         /*задать код сравнения в регистр данных канала*/
while ((TFLG1 & 0x04) = = 0)      /*ожидать события выходного сравнения*/
       {
       ;
       }
TFLG1 = 0x04;                     /*очистить флаг события канала 2*/
}
/*-------------------------------------------------------------------*/
```

Предложенный к рассмотрению программный фрагмент (timer2.c) отличается своей простотой и легкостью отладки. Это объясняется использованным методом программного опроса триггера события канала. Однако такой способ генерации импульсного сигнала становится непригодным, если по условию задачи управления необходимо формировать сразу несколько импульсных сигналов, и в каждом из них должны быть точно реализованы их временные параметры. Поэтому рассмотрим способ генерации импульсного сигнала с использованием подсистемы выходного сравнения таймера и подсистемы прерывания.

Генерация импульсной последовательности с использованием прерывания

Цель рассматриваемого примера (timer3.c) – познакомить читателя с техникой генерации импульсных сигналов с использованием подсистемы выходного сравнения, перезагрузка регистра данных которой реализуется в подпрограмме прерывания по очередному событию в канале выходного сравнения. Ниже перечислены биты и регистры управления, которые используются в данном примере:

- Бит разрешения работы модуля таймера TEN (регистр управления модулем таймера TSCR);
- Бит разрешения прерывания по переполнению счетчика временной базы TOI и биты выбора коэффициента деления программируемого делителя частоты на входе счетчика временной базы PR2:PR1:PR0 (регистр масок таймера TMSK2);
- Бит выбора режима работы канала IOSn (регистр режимов каналов захвата/сравнения TIOS). Если бит IOSn установлен в 1, то канал работает в режиме выходного сравнения;
- Биты выбора режима работы формирователя уровня канала выходного сравнения OMn:OLn (регистры управления таймером TCTL1 и TCTL2);
- Бит события в канала CnF (регистр флагов таймера TFLG1);
- Бит разрешения прерывания по событию в канале выходного сравнения CnI (регистр масок таймера TMSK1);
- Регистр данных канала TCn, код которого автоматически сравнивается с кодом счетчика временной базы. Момент равенства кодов и является событием выходного сравнения.

В тексте примера timer3.c мы будем генерировать импульсный сигнал с произвольной частотой и коэффициентом заполнения. Проанализировав текст программа Вы объясните, какую форму и какие временные параметры будет иметь формируемый сигнал.

Прежде, чем знакомиться с текстом программы timer3.c, вспомним, как оформляются подпрограммы прерывания при программировании на Си. Сначала следует убедиться, что Ваш заголовочный файл содержит определения макросов, которые позволят Вам использовать одноименные с командами ассемблера функции для запрета и разрешения прерываний в системе (см. раздел 4.12):

```
#define CLI ( )    asm("cli \n");      //разрешить маскируемые прерывания
#define SEI ( )  asm(''sei \n");       //запретить маскируемые прерывания
```

Также следует отметить, что специфика записи подпрограммы прерывания частично определяется типом используемого компилятора. В соответствии с предварительной договоренностью, в текстах примеров этой книги используется компилятор ICC12 компании Imagekraft.

```
/*-------------------------------------------------------------------*/
/* filename: timer3.c                                                */
/* MAIN PROGRAM: Эта программа генерирует импульсную последовательность */
/* в форме меандра с использованием таймера и подсистемы прерывания  */
/* Сигнал формируется на выходе 2 таймера (IC2)                      */
/*-------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>
```

```
/*используемые функции*/
void initialize(void);           /*функция инициализации*/
void toggle_isr(void);           /*подпрограмма прерывания toggle_isr*/

                            //обьявление функции обслуживания прерывания
#pragma interrupt_handler toggle_isr

//инициализация вектора прерывания
#pragma abs_address: OxF7EA
void (*Timer_Channel_2_interrupt_vector[]) () = {toggle_isr};
#pragma end_abs address

/*глобальные переменные*/
int c;

void main(void)
{
c = 100;
initialize() ;      /*инициализация подсистемы таймера*/
TMSK1 = 0x04;       /*разрешение прерывания по событию в канале 2*/
TFLG1 = 0xFF;       /*сброс всех флагов событий от каналов*/
CLI() ;             /*разрешить прерывания*/

while(l)
       {            /*ожидание прерывания*/
       ;
       }
}

/*---------------------------------------------------------------------*/
/* Функция initialize задает начальные установки модуля таймера       */
/*---------------------------------------------------------------------*/
void initialize(void)
{
TMSK2 = 0x02;       /*назначить коэффициент деления 4*/
TI0S = 0x04;        /*установить канал 2 в режим выходного сравнения*/
TSCR = 0x80;        /*разрешить работу таймера*/
TCTL2 = 0xl0;       /*назначить режим формирователя уровня */
                    /*"инвертирование"*/
}

/*---------------------------------------------------------------------*/
/* Функция toggle_isr - подпрограмма прерывания                        */
/*---------------------------------------------------------------------*/
void toggle_isr (void)
{
TFLG1 = 0xFF;       /*сброс всех флагов событий от каналов*/
TC2 = TC2 + c;      /*задать код сравнения в регистр данных канала*/
c = c + 100;        /*вычислить новое значение с*/
}

/*---------------------------------------------------------------------*/
```

Проанализируйте функцию toggle_isr. Генерацию какого импульсного сигнала она обеспечивает?

В следующем примере timer4.c показаны потенциальные возможности подсистемы выходного сравнения, обслуживаемой по прерываниям. В примере генерируются сразу две импульсные последовательности с разными частотами и коэффициентами заполнения. Нетрудно видеть, что число одновременного генерируемых импульсных сигналов может быть увеличено.

```
/*-------------------------------------------------------------------*/
/*filename: timer4.c                                                 */
/*MAIN PROGRAM: Эта программа генерирует две импульсных последовательности*/
/* с использованием таймера и двух каналов подсистемы прерывания    */
/*. Сигналы формируются на выходах 2  и 3 таймера.                  */
/*-------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>

/*используемые функции*/
void initialize(void);           /*функция инициализации*/
void toggle1_isr(void);          /*подпрограмма прерывания toggle1_isr*/
void toggle2_isr(void);          /*подпрограмма прерывания toggle2_isr*/

//обьявление функции обслуживания прерывания
#pragma interrupt_handler toggle1_isr
#pragma interrupt_handler toggle2_isr

//инициализация векторов прерывания
#pragma abs_address: OxF7E8
void (*Timer_Channel_3_interrupt_vector[]) () = {toggle2_isr};
void (*Timer_Channel_2_interrupt_vector[]) () = {toggle1_isr};
#pragma end_abs address

void main(void)
{
initialize() ;      /*инициализация подсистемы таймера*/
TMSK1 = 0x0C;       /*разрешение прерывания по событию в каналах 2 и 3*/
TFLG1 = 0xFF;       /*сброс всех флагов событий от каналов*/
CLI() ;             /*разрешить прерывания*/

while(1)
       {            /*ожидание прерывания*/
       ;
       }
}
/*-------------------------------------------------------------------*/
/* Функция initialize задает начальные установки модуля таймера     */
/*-------------------------------------------------------------------*/
void initialize(void)
{
TMSK2 = 0x02;       /*назначить коэффициент деления 4*/
TI0S = 0x0C;        /*установить каналы 2 и 3 в режим выходного*/
                    /*сравнения*/
TSCR = 0x80;        /*разрешить работу таймера*/
TCTL2 = 0x50;       /*назначить режим формирователя уровня */
                    /*"инвертирование" для обоих каналов*/
}
```

```
/*--------------------------------------------------------------------*/
/* Функция toggle1_isr - подпрограмма прерывания по событию в канале 2*/
/*--------------------------------------------------------------------*/
void toggle1_isr (void)
{
TFLG1 = 0x04;       /*сброс флага события канала 2*/
TC2 += 9091;        /*задать код сравнения в регистр данных канала*/
}

/*--------------------------------------------------------------------*/

/*--------------------------------------------------------------------*/
/* Функция toggle2_isr - - подпрограмма прерывания по событию в канале 3 */
/*--------------------------------------------------------------------*/
void toggle2_isr (void)
{
TFLG1 = 0x08;       /*сброс флага события канала 3*/
TC2 += 4854;        /*задать код сравнения в регистр данных канала*/
}

/*--------------------------------------------------------------------*/
```

Проанализируйте функции toggle1_isr и toggle2_isr. Генерацию каких импульсных последовательностей они обеспечивают?

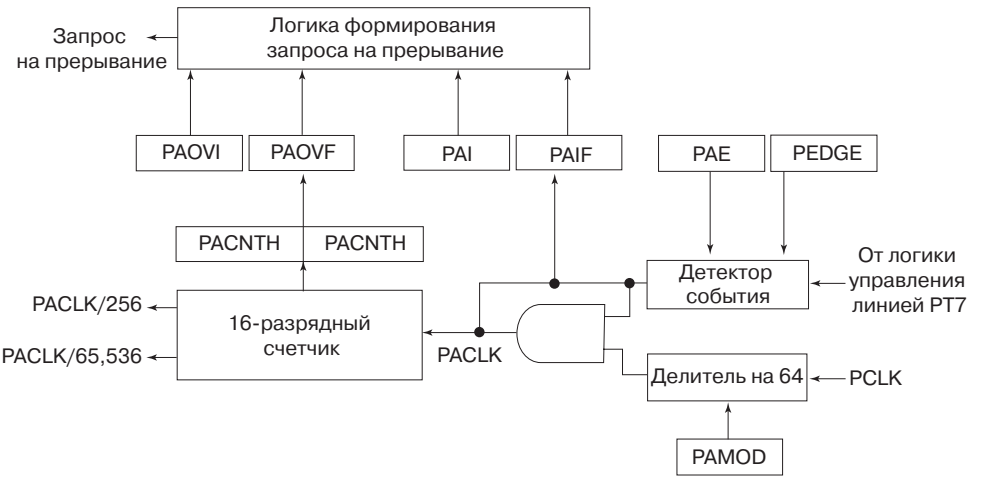
4.14.5. Счетчик событий

Счетчик событий или аккумулятор импульсов (Pulse Accumulator) – это подсистема модуля таймера, которая разработана специально для подсчета импульсов. Основной элемент счетчика событий – 16-разрядный счетчик, который может работать в двух режимах:

- *Счетчик внешних событий*. В этом режиме счетчик PACNT подсчитывает число событий, которое детектируется его аппаратными средствам на входе PAI. Тип события (только нарастающий или только спадающий фронт сигнала) определяется посредством программных установок.
- *Стробируемый таймер*. В этом режиме счетчик PACNT тактируется внутренними импульсами с частотой f_{BUS} /64, а сигналы на входе PAI разрешают или запрещают счет.

Режимы работы счетчика

Структурная схема подсистемы счетчика событий представлена на рис. 4.45. Изучив ее, вы увидите, что счетчик событий в качестве входа PAI, на который должны поступать внешние импульсы, использует линию PT7 порта PORT T. Таким образом, обслуживание подсистемы счетчика событий - альтернативная функция линии 7 порта PORT T. Однако линия PT7 обладает еще одной альтернативной функцией. Она используется как вход или как выход канала 7 подсистемы захвата/сравнения модуля таймера. Какие программные установки в регистрах управления таймера необходимо выполнить, чтобы конфигурировать линию PT7 как вход PAI счетчика событий?



Имя регистра: PACTL — Адрес: $00A0

	7	6	5	4	3	2	1	0
	0	PAEN	PAMOD	PEDGE	CLK1	CLK0	PAOVI	PAI
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: PAFLG — Адрес: $00 A1

	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	PAOVF	PAIF
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: PACNTH — Адрес: $00A2

	7	6	5	4	3	2	1	0
	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: PACNTL — Адрес: $00A3

	7	6	5	4	3	2	1	0
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.45. *Структура и регистры управления счетчика вешних событий*

- Сбросить бит IOS7 в регистре TIOS ($0080), тогда в канале 7 будет запрещен режим выходного сравнения;
- Сбросить биты OM7:OL7 регистра TCNTL1($0088),тогда линия PT7 окажется отсоединенной от аппаратных средств канала 7;
- Сбросить бит OC7M7 в регистре OC7M ($0082).

Если все эти установки выполнены, то сигнал с лиги PT7 будет поступать на вход детектора событий подсистемы счетчика событий. В каждом из режимов работы счетчика детектор выполняет разные функции, однако в обоих режимах работы детектор становится активным при установке бита PAEN регистра управления счетчиком PACTL (рис. 4.45) в 1.

Если подсистема находится в режиме счетчика внешних событий, бит PEDGE определяет: по какому фронту сигнала, нарастающему (при PEDGE = 1) или спадающему (при PEDGE = 0) будет переключаться счетчик. Если детектор фиксирует на входе PAI заданное битом PEDGE изменение входного сигнала, то устанавливается флаг события PAIF.

Если подсистема работает в режиме стробируемого таймера, то бит PEDGE определяет уровень сигнала на входе PAI, при котором будет разрешен счет для внутреннего счетчика PACNT. Если бит PEDGE = 1, то счет разрешен при низком логическом уровне на входе PAI. При этом по нарастающему фронту сигнала на входе PAI будет устанавливаться флаг событий PAIF. Если же бит PEDGE = 0, счет разрешается при высоком уровне PAI, флаг PAIF устанавливается по спадающему фронту PAI.

Если работы подсистемы счетчика событий разрешена, то 16-разрядный счетчик начинает считать с нулевого значения. Текущий код счетчика может быть считан из 16-разрядного регистра текущего состояния счетчика PACNT, который в памяти размещается в двух ячейках памяти: PACNTH – старший байт счетчика событий, PACNTL – младший байт счетчика событий (рис. 4.45). Каждое переполнение 16-разрядного счетчика событий фиксируется установкой триггера переполнения PAOVF. Этот триггер сбрасывается посредством записи в установленный разряд PAOVF единичного значения.

Оба рассмотренных флага: флаг переполнения счетчика PAOVF и флаг события на входе счетчика PAIF, – способны генерировать запросы прерывания. Прерывания от указанных флагов разрешаются битами PAOVI и PAI соответственно. Биты PAOVI и PAI располагаются в регистре флагов счетчика событий PAFLG (рис. 4.45).

Регистры управления счетчиком событий

В данном разделе мы рассмотрим регистры специальных функций, которые используются для управления режимами работы счетчика событий.

Регистр управления счетчиком событий

Регистр управления счетчиком событий PACTL (Pulse Accumulator Control Register) располагается в памяти МК по адресу $00A0. Формат регистра представлен на рис. 4.45. Регистр PACTL используется для задания режима работы счетчика событий, для назначения режима работы его детектора событий, для разрешения прерываний от подсистемы счетчика событий, а также для выбора источника тактирования счетчика временной базы модуля таймера.

Бит PTAE разрешает работу подсистемы счетчика событий. При PTAE = 1 работа подсистемы разрешена, при PTAE = 0 подсистема счетчика находится в неактивном состоянии. Бит PAMOD служит для выбора режима работы. При PAMOD = 0 подсистема работает в режиме счетчика внешних событий, при PAMOD = 1 для подсистемы назначается режим стробируемого таймера.

Бит PEDGE управляет режимом работы детектора событий подсистемы счетчика. Если счетчик работает в режиме счетчика внешних событий (PAMOD = 0), то при PEDGE = 1 детектор события настраивается на распознавание положительного фронта сигнала, при PEDGE = 0 – отрицательного фронта. При работе в режиме стробируемого таймера (PAMOD = 1) бит PEDGE определяет вид сигнала, разрешающего переключение счетчика таймера. Если PEDGE = 1, то счет разрешается при подаче 1 на вход PAI, при этом флаг PAI устанавливается по отрицательному фронту входного сигнала. Если же PEDGE = 0, то счет разрешается при подаче 0 на вход PAI, при этом флаг PAI устанавливается по положительному фронту входного сигнала.

Биты CLK1:CLK0 назначают источник тактирования для счетчика временной базы модуля таймера (см. рис. 4.27).

Биты PAOVI и PAI (не путать с одноименным входом) разрешают прерывания по событию переполнения счетчика (флаг PAOVF) и по событию на входе PAI (флаг PAIF), тип которого определяется режимом работы и значением бита PEDGE.

Регистр флагов счетчика событий

Регистр флагов счетчика событий PAFLG (Pulse Accumulator Flag Register) располагается в памяти МК по адресу $00A1 и содержит всего два значимых бита (рис. 4.45). Флаг переполнения счетчика устанавливается, когда 16-разрядный счетчик подсистемы изменяет свое значение с $FFFF на $0000. Флаг события подсистемы PAIF устанавливается в 1 при каждом перепаде потенциала на входе PAI, который указан битом PEDGE и режимом работы счетчика. Оба этих бита сбрасываются посредством записи в установленный бит 1.

Регистр текущего состояния счетчика событий

Регистр счетчика PACNT (Pulse Accumulator Counter Register) содержит текущий код 16-разрядного счетчика подсистемы. Поэтому он располагается в двух ячейках памяти: старший байт PACNTH – по адресу $00A2, младший байт PACNTL – по адресу $00A3. Поскольку изменение кода счетчика может произойти в произвольный момент времени, рекомендуется производить его чтение в двухбайтовом формата с использованием типа LDD или LDX/LDY.

Пример использования счетчика событий

Мы рассмотрим пример определения скорости движения велосипеда по сигналам датчика Холла, который установлен на колесе велосипеда. Датчики Холла доступны в различных модификациях. Обобщая сведения о датчиках Холла, можно выделить следующие три типа датчиков:

- Линейные датчики, напряжение на выходе которых пропорционально напряженности магнитного поля, в которое помещен датчик;
- Биполярные датчики, формируют выходной сигнал, находясь вблизи южного магнитного полюса, сбрасываются в 0, находясь вблизи северного магнитного полюса;

Рис. 4.46. *Колесо велосипеда с датчиком Холла*

- Однополярные датчики, формируют выходной сигнал, находясь вблизи южного магнитного полюса, сбрасываются в 0 при отсутствии магнитного поля.

В нашем примере мы помести магнит на спицу вращающегося колеса. Однополярный датчик Холла установим на вилке колеса, выход датчика соединим со входом (PT7) счетчика внешних событий МК семейства 68HC12 (рис. 4.46). Тогда на выходе датчика будет генерироваться один импульс при каждом полном обороте колеса. Если мы подсчитаем число импульсов с выхода датчика на известном временном интервале, мы сможем определить скорость движения и расстояние, которое было преодолено за время измерения.

Читателю предлагается самостоятельно написать программу расчета скорости и преодоленного расстояния, используя следующие подсказки:

- В первую очередь определите связь между одним оборотом колеса и пройденной дистанцией пути. Предположите, что диаметр колеса равен 66,04 см (26 дюймов).
- Произведите инициализацию подсистемы счетчика внешних событий:
  a. Подсистема счетчика событий использует вход PT7 для подключения внешнего импульсного сигнала. Для конфигурирования линии PT7 на ввод установите бит 7 регистра TIOS в 0, также в 0 должны быть установлены биты 6 и 7 регистра PCTL1.
  b. Запрограммируйте на выбранный режим регистр управления счетчиком событий PACTL. Определите необходимое состояние каждого бита самостоятельно.
- Считайте под управлением программы текущее состояние счетчика внешних событий, используя регистр PACNT.
- Сформируйте временной интервал, длительность которого должна быть достаточна накопления в счетчике событий нескольких десятков отсчетов. Тогда точность измерения скорости будет приемлемой (несколько единиц %).
- Считайте под управлением программы новое текущее состояние счетчика событий.
- Получите разность кодов и вычислите скорость и пройденное расстояние.

Приведенный ниже программный фрагмент поможет Вам выполнить инициализацию подсистемы счетчика событий.

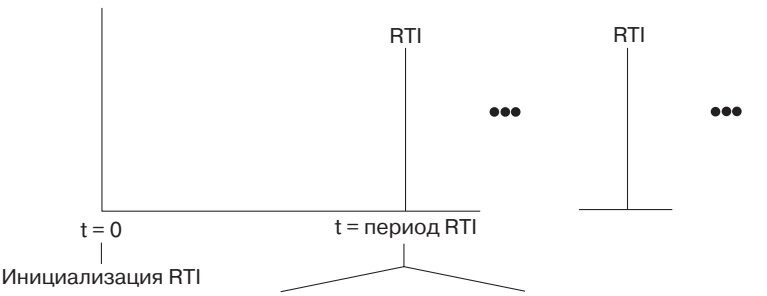
```
/*-------------------------------------------------------------------*/
/* Функция initialize_PA задает начальные установки подсистемы счетчика событий */
/*-------------------------------------------------------------------*/
void initialize_PA(void)
{
TIOS = 0x00;              /*конфигурировать вход PT7 для работы */
TCTL1 = 0x00;             /*в качестве источника сигналов для счетчика*/
OC7M = 0x00               /*внешних событий - 3 команды*/

TSCR = 0x80;              /*разрешить работу всего модуля таймера*/
PACTL = 0x70;             /*разрешить работу в режиме счетчика внешних*/
                          /*событий по положительному фронту сигнала*/
}
/*-------------------------------------------------------------------*/
```

4.15. Модуль меток реального времени

Многие применения требуют организации выполнения одного и того же фрагмента программы через равные интервалы времени. Для удобства разработчиков МК семейства 68HC12/HCS12 оснащены специальным модулем меток реального времени RTI (Real Time Interrupt), который генерирует равноотстоящие во времени запросы на прерывание (рис. 4.47). Тогда фрагмент программы, которые должен исполняться через равные интервалы времени, может быть оформлен как подпрограмма прерывания по запросу RTI. И желаемый алгоритм функционирования устройства будет реализован.

Модуль меток реального времени RTI использует два регистра специальных функций: регистр управления RTICTL (Real Time Interrupt Control Register) и регистр флагов RTIFLG (Real Time Interrupt Flag Register).

Бит RTIE в регистре управления RTICTL (рис. 4.47) разрешает работу подсистемы меток реального времени. При RTIE = 1 подсистема находится в активном состоянии, при RTIE = 0 работа системы запрещена. Биты RTR2:RTR1:RTR0 определяют период следования меток реального времени. Таблица рис. 4.47 устанавливает соответствие между кодовой комбинацией битов RTR2:RTR1:RTR0 и временным интервалом между двумя соседними метками модуля RTI. Последний именуют периодом RTI.



Имя регистра: RTICTL — Адрес: $0014

	7	6	5	4	3	2	1	0
	RTIE	RSWAI	RSBCK	0	RTBYP	RTR2	RTR1	RTR0
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: RTIFLG — Адрес: $0015

		6	5	4	3	2	1	0
	RTIF	0	0	0	0	0	0	0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.47. *Временная диаграмма, поясняющая принцип действия модуля меток реального времени, и регистры управления модулем*

Выбор периода генерации меток реального времени

RTR[2:0]	Коэффициент деления 2^x	При частоте внутренней системной шины	
		4 МГц	8 МГц
000	*зарезервирован*	нет	нет
001	13	2,048 мс	1,024 мс
010	14	4,096 мс	2,048 мс
011	15	8,192 мс	4,096 мс
100	16	16,384 мс	8,192 мс
101	17	32,768 мс	16,384 мс
110	18	65,536 мс	32,768 мс
111	19	131,072 мс	65,536 мс

Рис. 4.47. *Временная диаграмма, поясняющая принцип действия модуля меток реального времени, и регистры управления модулем*

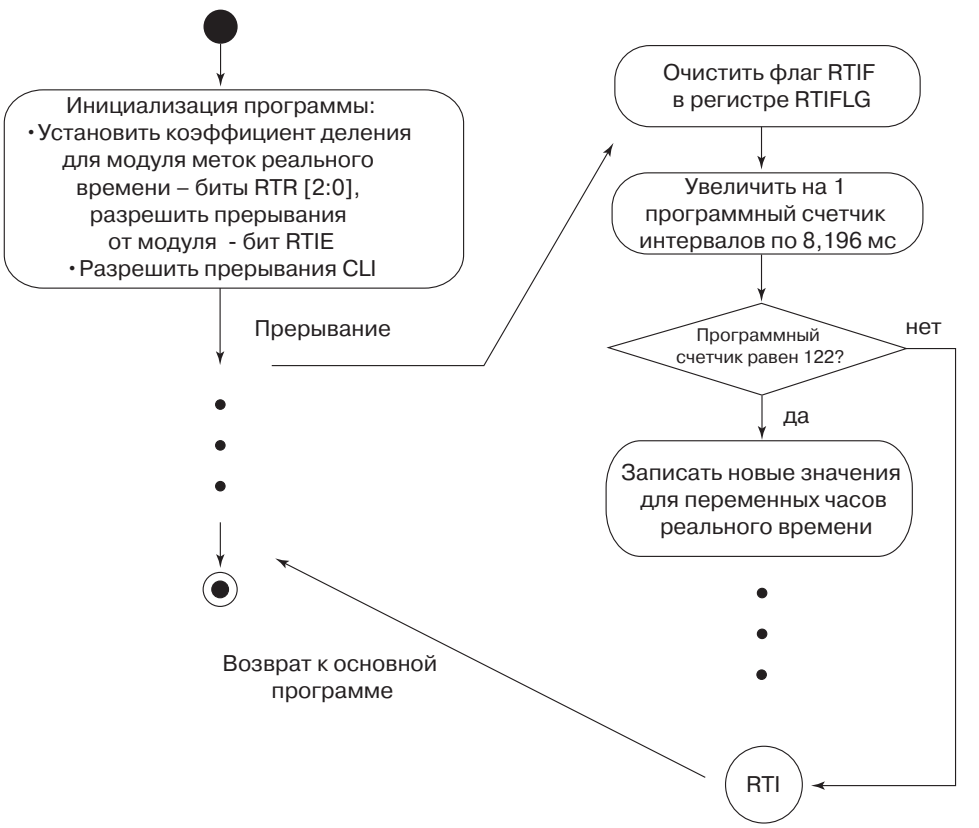


Рис. 4.48. *Блок-схема алгоритма часов реального времени*

Регистр флагов RTIFLG (рис. 4.47) содержит всего один флаг RTIF. Этот флаг устанавливается, когда модуль закончил отсчет очередного периода RTI.

Счетчик подсистемы меток реального времени тактируется импульсной последовательностью с частотой f_{BUS}. В модуле меток реального времени эта частота делится. Коэффициент деления, устанавливаемый разрядами RTR2:RTR1:RTR0, можно выбрать по таблице рис. 4.47. Проанализировав данные таблицы, можно установить, что для МК с частотой внутренней шины 8 МГц максимальный период меток реального времени составляет примерно 65 мс.

Пример использования модуля меток реального времени

Ниже приведен исходный текст программы realtime.c для реализации часов реального времени на основе отсчетов модуля RTI с интервалами 8,196 мс. В подпрограмме прерывания программный счетчик накапливает 122 отсчета RTI (рис.4.48), которые составляют временной интервал длительностью 1 с. По прошествии каждой секунды инкрементируется программный счетчик sec_ctr, по прошествии каждой минуты – счетчик mins_ctr и т.д. вплоть до счетчика дней. При желании функции программы можно расширить, введя подсчет дня недели, месяца и года.

```
/*-------------------------------------------------------------------*/
/* filename: realtime.c                                              */
/* MAIN PROGRAM: Эта программа генерирует две импульсных             */
/* последовательности с использованием таймера и двух каналов подсистемы */
/* прерывания.Сигналы формируются на выходах 2  и 3 таймера.         */
/*-------------------------------------------------------------------*/

/*подключаемые файлы*/
#include<912b32.h>

/*используемые функции*/
void RTI_isr{void);              /*подпрограмма прерывания по RTI*/

/* interrupt pragrma */
#pragrma interrupt_handler RTI_isr

/*инициализация таблицы векторов прерывания*/
#pragrna abs_address: 0xF7F0
void (*RTI_interrupt_vector[]) ()={RTI_isr};
#pragrna end_abs_address

/*глобальные переменные*/
unsigned int ms_ctr, sec_ctr, mins_ctr, hrs_ctr, days_ctr;

void main{void)
{
ms_ctr = 0;               /*инициализация переменных*/
sec_ctr = 0;
mins_ctr = 0;
hrs_ctr = 0;
days_ctr = 0;

RTICTL = 0x84;            /*разрешить прерывания от модуля RTI, выбрать*/
CLI ();                   /*период RTI 8,196 мс*/
```

```
while(1)                    /*ожидать прерывание*/
       {
       ;
       }
}
/*-------------------------------------------------------------------*/
/* Функция RTI_isr подпрограмма прерывания каждые 8,196 мс           */
/*-------------------------------------------------------------------*/
void RTI_isr(void)
{
RTIFLG = 0x80;              /*сброс флага события RTI*/

ms_ctr = ms_ctr+1;          /*обновить счетчик миллисекунд*/

if{ms_ctr = = 122)          /*обновить счетчик секунд*/
       {
       ms_ctr = 0;
       sec_ctr = sec_ctr +1;
       }

if(sec_ctr = = 60)          /*обновить счетчик минут*/
       {
       sec_ctr = 0;
       mins_ctr = mins_ctr + 1;
       }

if(mins_ctr = = 60)                /*обновить счетчик часов*/
       {
       mins_ctr = 0;
       hrs_ctr = hrs_ctr + 1;
}
if(hrs_ctr = = 24)          /*обновить счетчик дней*/
       {
       hrs_ctr = 0;
       days_ctr = days_ctr +1;
       }
}
/*-------------------------------------------------------------------*/
```

4.16. Модуль таймера ECT в составе МК MC68HC12BE32 и HCS12

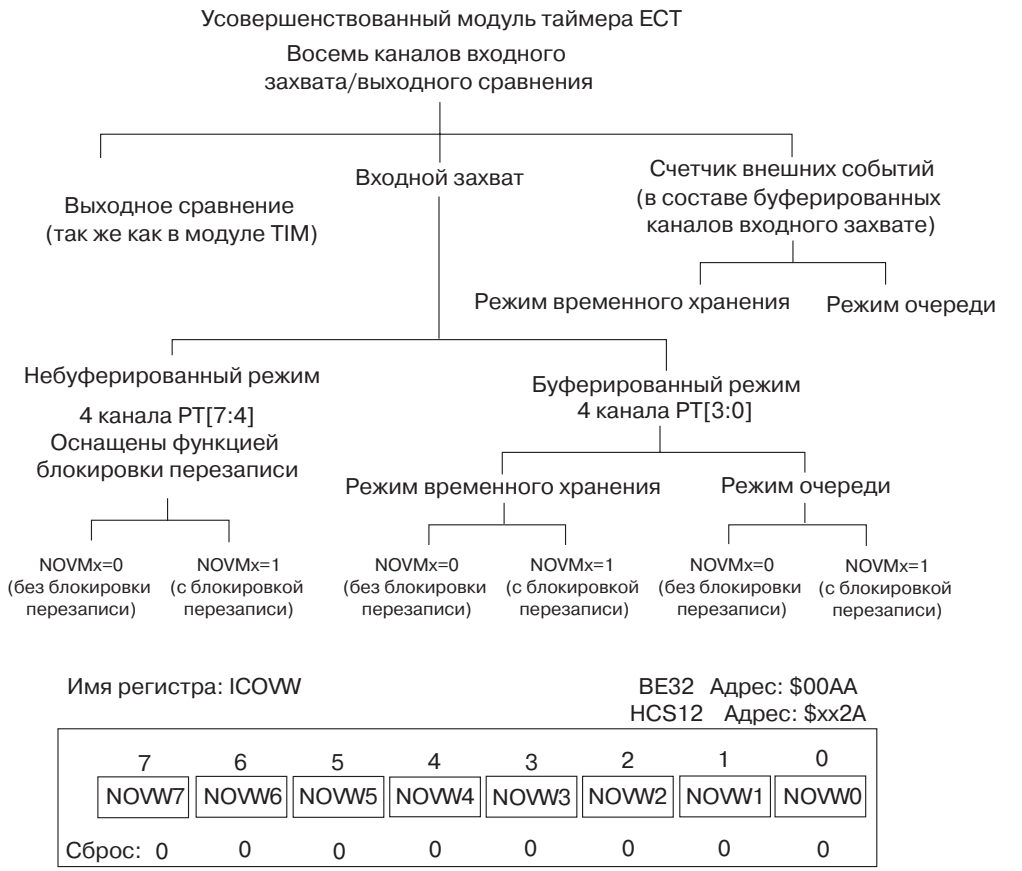
Все микроконтроллеры семейства HCS12 и всего одна модель MC68HC12BE32 семейства 68HC12 оснащены более совершенным модулем таймера ECT (Enhanced Capture Timer). Модуль таймера EST унаследовал основные технические решения от своего предшественника – модуля таймера TIM. Поэтому, так же как в модуле TIM, основу таймера EST составляют 16-разрядный счетчик временной базы и восемь универсальных каналов захвата/сравнения.

Однако в модуле EST четыре канала из восьми в режиме входного захвата обладают дополнительным регистром. Такое решение позволяет в режиме входного захвата зафиксировать два момента изменения сигнала на входе канала прежде, чем бу-

дет установлен триггер события в канале CnF. Рассматриваемые каналы называют буферированными, они могут работать как в режиме временного хранения, так и в режиме очереди. Четыре оставшихся канала называют небуферированными. Правила функционирования этих каналов в режиме входного захвата полностью соответствуют аналогичным для модуля TIM. Полный набор функций каналов захвата/сравнения модуля таймера EST приведен на рис. 4.49.

Второе отличие модуля EST от модуля TIM состоит в том, что модуль EST имеет в своем составе четыре 8-разрядных счетчика события (PACN3...PACN0). Эти счетчики могут быть объединены парами для получения двух 16-разрядных счетчиков событий: (PACN3: PACN2) и (PACN1: PACN0).

Обратимся далее к более подробному рассмотрению дополнительных режимов работы модуля EST.



Имя регистра: ICOVW

BE32 Адрес: $00AA
HCS12 Адрес: $xx2A

7	6	5	4	3	2	1	0
NOVW7	NOVW6	NOVW5	NOVW4	NOVW3	NOVW2	NOVW1	NOVW0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.49. *Усовершенствованный модуль таймера ECT*

4.16.1. Небуферированные каналы входного захвата

Алгоритм функционирования небуферированных каналов входного захвата полностью аналогичен функционированию в аналогичном режиме каналов модуля TIM. Однако небуферированные каналы модуля EST снабжены дополнительной функцией управления режимом. Регистр управления порядком перезаписи ICOVW (Input Control Overwrite Register) содержит восемь битов NOVWn (рис. 4.49). Если бит NOVWn установлен в 1, то регистр данных небуферированного канала с номером n не может быть перезаписан под действием аппаратных средств, пока этот регистр не будет считан программой. Если бит NOVWn = 0, то функция блокировки записи для небуферированного канала снимается.

4.16.2. Буферированные каналы входного захвата

Четыре буферированных канала входного захвата оснащены вторым регистром данных. В результате, канал может запомнить два момента времени, которые соответствуют двум последовательным событиям на входе канала. Флаг события канала устанавливается после того, как оба регистра данных канала будут заполнены кодами событий.

В приложении к буферированным каналам входного захвата принято использовать следующую терминологию. Первый регистр данных канала считается очищенным, если его значение переписано во второй регистр данных или считано программой. Второй регистр данных канала считается очищенным, если его значение прочитано программой.

Буферированные каналы входного захвата могут работать или в режиме временного хранения, или в режиме очереди. Режим работы назначается битом LATQ в регистре управления режимом входного захвата ICSYS. Если бит LATQ установлен в 1, то буферированные каналы работают в режиме временного хранения. Если же бит LATQ = 0, то для буферированных каналов назначается режим очереди.

В режиме временного хранения (LATQ = 1) текущий код счетчика временной базы запоминается в первом регистре данных при изменении входного сигнала, на которое настроен детектор события. Далее этот код автоматически переписывается во второй регистр данных при наступлении одного из трех событий:

- Код 16-разрядного вычитающего счетчика MCCNT достиг $0000;
- В счетчик MCCNT под управлением программы вписан код $0000;
- Бит ICLAT в регистре управления вычитающим счетчиком MCCTL (рис. 4.50) программно установлен в 1.

В режиме временного хранения биты управления NOVWn действуют также, как и небуферированных каналах (см. 14.6.1)

В режиме очереди (LATQ = 0) и при NOVWn = 0 событие входного захвата вызывает автоматическую перезапись содержимого первого регистра данных канала во второй регистр данных, а в первый регистр данных загружается текущий код счетчика временной базы (рис. 4.51). Если буферированный канал работает в режиме очереди и бит NOVWn = 1, то и первый и второй регистры данных не запоминают момента наступления события, если они не очищены, т.е. не были считаны программой до наступления события.

4.16.3. Особенности счетчиков событий

Модуль таймера EST оснащен четырьмя 8-разрядными счетчиками событий (рис. 4.52). Счетчики событий могут быть объединены попарно, тогда они образуют два 16-разрядных счетчика события. Так же, как и каналы захвата, счетчики событий могут работать или в режиме временного хранения, или в режиме очереди (рис. 4.52).

4.16.4. Регистры управления модуля EST

На рис. 4.53 приведен формат регистров модуля таймера EST, которые отсутствуют в рассмотренном ранее модуле TIM. Адреса регистров указаны как для МК BE32 семейства 68HC12, так и для микроконтроллеров семейства HCS12. Приведенные адреса можно рассматривать как абсолютный адрес, если в проекте используется настоящий физический адрес начала блока регистров $0000. Если же блок регистров перемещен в другое, виртуальное адресное пространство, то указанные адреса следует рассматривать как смещение адреса регистра относительно адреса начала регистрового блока.

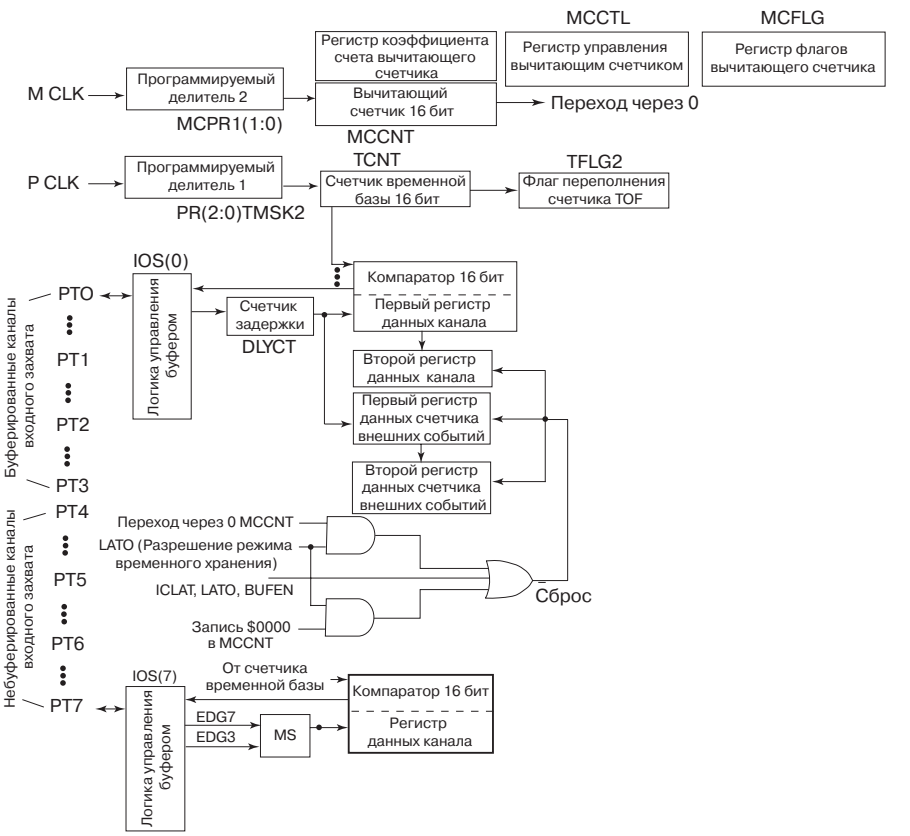


Рис. 4.50. *Аппаратные средства буферированного канала входного захвата – режим временного хранения*

Регистр управления порядком перезаписи

Регистр управления порядком перезаписи ICOVW (Input Control Overwrite Register) располагается в памяти МК по адресу $00AA/$002A. Формат регистра представлен на рис. 4.53. Регистр содержит восемь битов NOVWn (n – номер канала). Если бит NOVWn = 0, то первый и второй регистры данных канала входного захвата с номером n могут быть автоматически перезаписаны при наступлении очередного события входного захвата. Если же бит NOVWn = 1, то регистры данных канала не могут быть перезаписаны, если они предварительно не были очищены, т.е. считаны под управлением программы.

Регистр управления режимом входного захвата

Регистр управления режимом входного захвата ICSYS (Input Control System Control Register) содержит бит разрешения буферированного режима в каналах BUFEN и бит выбора режима работы буферированных каналов LATQ (временного хранения или очереди). Если бит BUFEN установлен, то работа второго регистра данных и буферного регистра счетчика событий буферированных каналов модуля

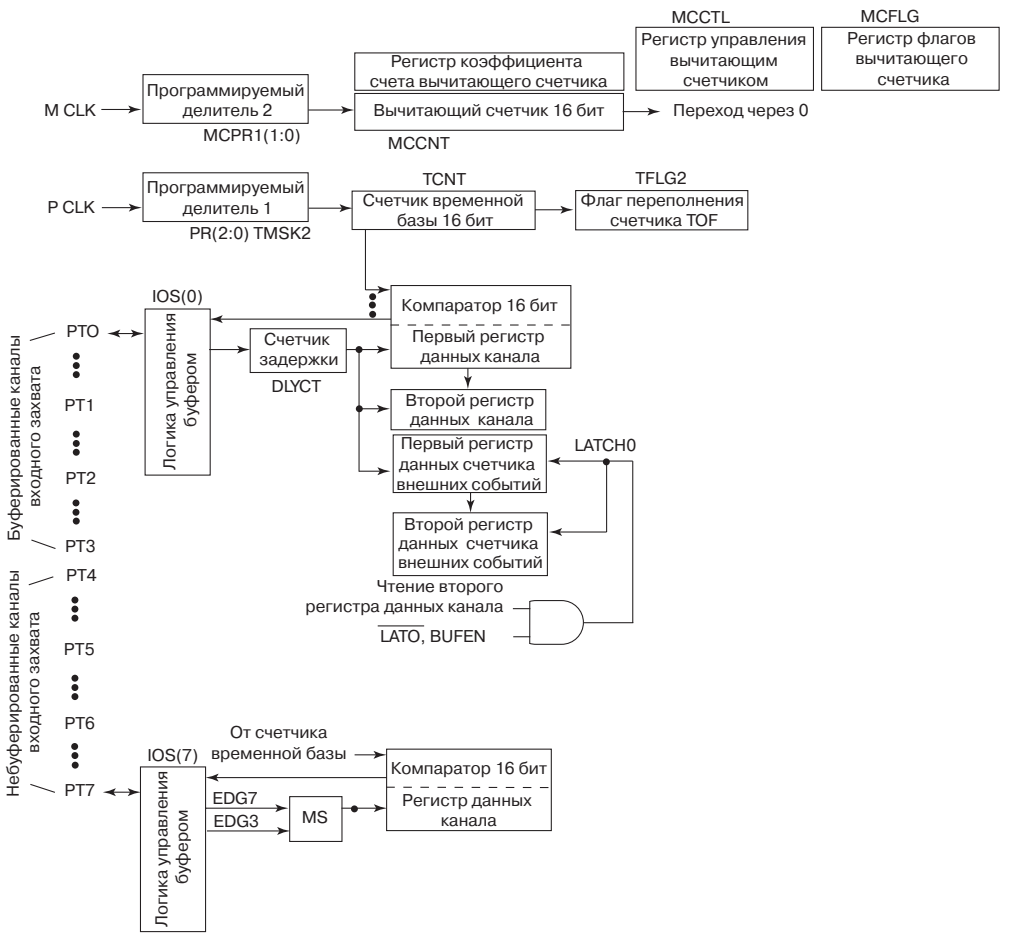


Рис. 4.51. *Аппаратные средства буферированного канала входного захвата – режим очереди*

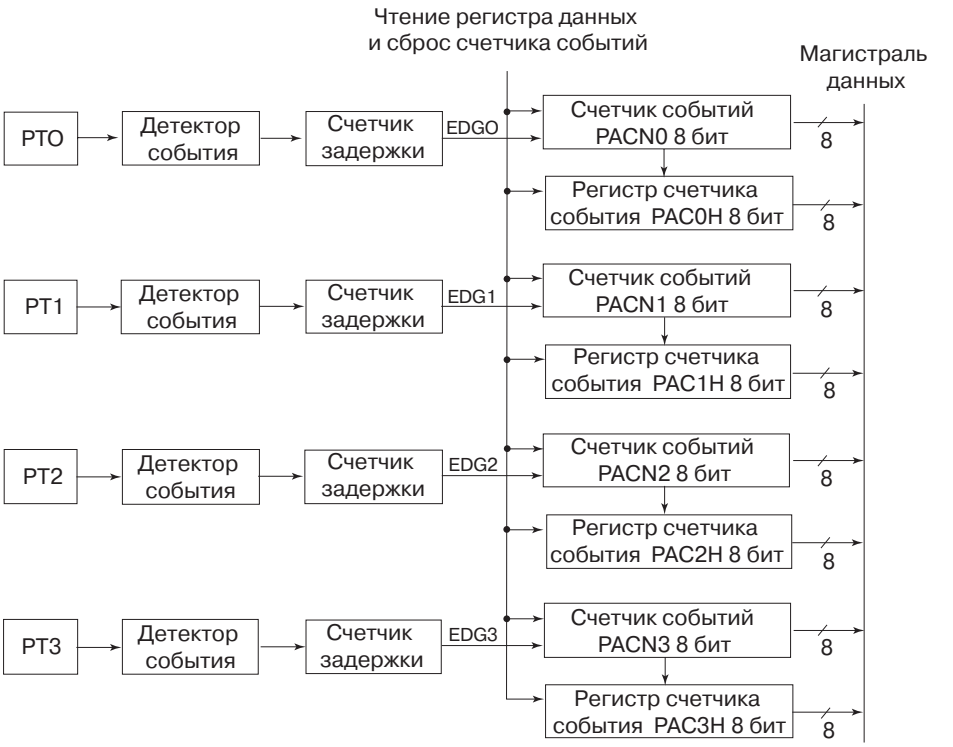


Рис. 4.52. *Аппаратные средства счетчика внешних событий в составе таймера ECT*

таймера разрешена. Если бит LATQ равен 1, то буферированные каналы входного захвата работают в режиме временного хранения, при LATQ = 0 эти каналы работают в режиме очереди.

Регистр управления счетчиком задержки

Регистр управления счетчиком задержки DLYCT (Delay Counter Control Register) разрешает вставку регулируемого временного интервала между моментом распознавания детектором события изменения входного сигнала и интерпретацией этого изменения как события входного захвата со всеми последующими действиями аппаратных средств канала таймера. Событие входного захвата будет распознано, если логический уровень сигнала по истечении временной задержки будет противоположен уровню сигнала перед запуском счетчика задержки. Такое решение позволяет повысить устойчивость входов к шумам на фронтах переключения. Длительность задержки определяется битами DLY1:DLY0. При DLY1:DLY0 = 00 задержка не вставляется. При DLY1:DLY0 = 01 задержка равна 256 периодам сигнала P clock системы тактирования, при DLY1:DLY0 = 10 задержка составляет 512 периодов, при DLY1:DLY0 = 11 – 1024 периода того же сигнала.

Регистр управления 16-разрядным вычитающим счетчиком

Регистр управления вычитающим счетчиком MCCTL (Modulus Down-Counter Control Register) предназначен для задания режима работы 16-разрядным вычитающим счетчиком с изменяемым коэффициентом счета. Если бит MODMC установлен в 0, то счетчик уменьшает код, начиная с предварительно записанного значения до 0,

затем останавливается. Если бит MODMC равен 1, счетчик считает непрерывно. При достижении кода $0000, он автоматически перезагружается кодом, записанным в регистр коэффициента счета MCCNT.

Регистр коэффициента счета вычитающего счетчика

Регистр коэффициента счета вычитающего счетчика MCCNT (Modulus Down-Counter Register) содержит изменяемый программно коэффициент счета 16-разрядного вычитающего счетчика.

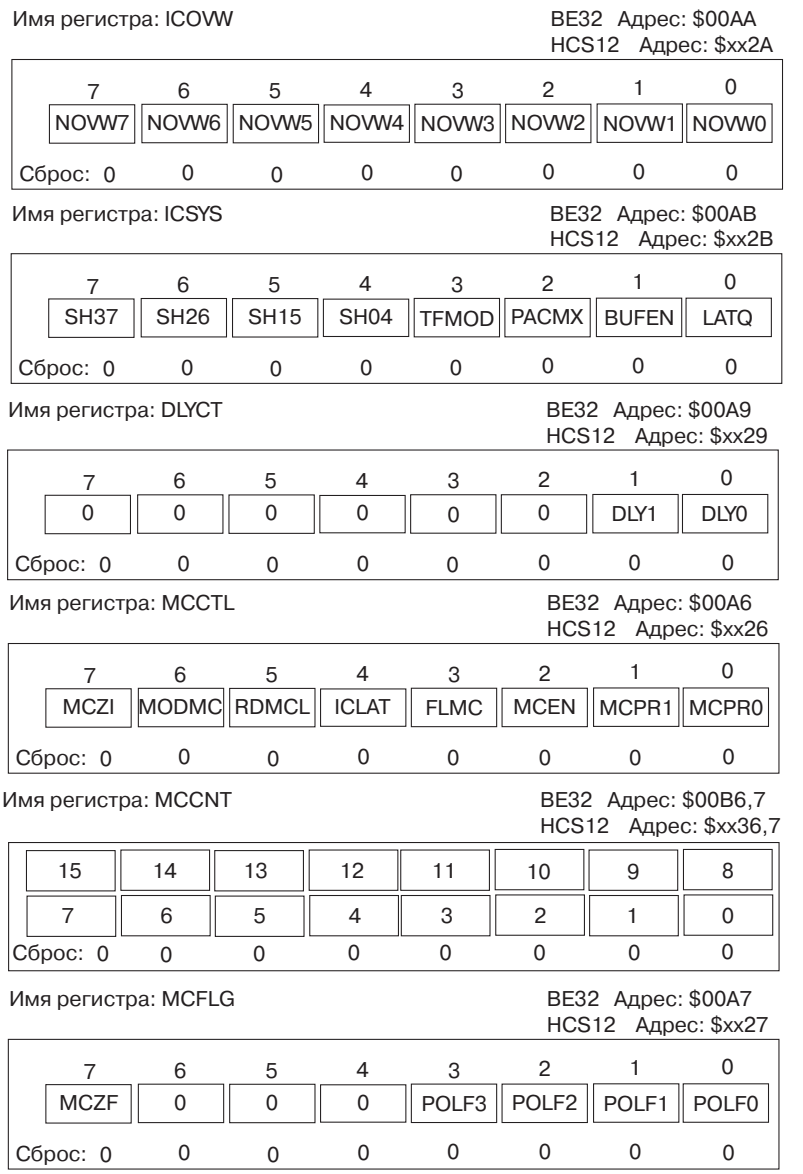


Рис. 4.53. *Формат регистров модуля ECT*

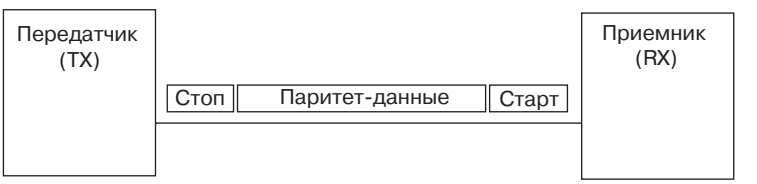
Регистр флагов вычитающего счетчика

Регистр флагов вычитающего счетчика MCFLG (Modulus Down-Counter Flag Register) содержит флаг обнуления счетчика MCZF. Этот флаг устанавливается в 1, когда код счетчика становится равным $0000. Флаг MCZF сбрасывается посредством записи в его разряд 1.

4.17. Обмен информацией в последовательном коде: многофункциональный последовательный интерфейс

Обмен данными в последовательном коде – эффективный с точки зрения аппаратной реализации способ связи между двумя интеллектуальными устройствами. Всего одна линия потребуется для передачи цифровых кодов от одного микропроцессорного устройства к другому. Данные по этой линии передаются последовательно во времени так, что в каждый момент времени происходит передача только одного бита данных. Обмен в параллельном коде значительно превышает по скорости обмен в последовательном коде. Однако он требует множества линий связи между устройствами, что увеличивает энергию потребления и размеры изделия.

В МК семейства 68HC12/HCS12 обмен в последовательном коде обеспечивает модуль многофункционального последовательного интерфейса (MSI – Multiple Serial Interface). Этот модуль содержит в себе две независимых подсистемы последовательного обмена: контроллер асинхронного обмена SCI (Serial Communication Interface) и контроллер синхронного обмена SPI (Serial Peripheral Interface). Каждый из контроллеров обслуживают линии порта S, для которых эта функция является альтернативной.



а) Последовательный коммуникационный интерфейс SCI

а) Последовательный периферийный интерфейс SPI

Рис. 4.54. *Два наиболее распространенных типа последовательных интерфейсов*

Контроллер SCI поддерживает полнодуплексный обмен в асинхронном режиме с форматом кадра, который совместим с последовательным интерфейсом персонального компьютера RS-232. Термин полнодуплексный означает, что в один и тот же момент времени один и тот же контроллер SCI может как принимать, так и передавать информацию. Термин асинхронный характеризует способ передачи данных, при котором дополнительный сигнал синхронизации, подтверждающий наличие очередного бита данных на линии, не используется. Следовательно, синхронизация между обменивающимися устройствами отсутствует. Для однонаправленной асинхронной передачи информации между двумя устройствами потребуется всего линия связи, что отражено на рис. 4.54.

Синхронный интерфейс SPI поддерживает синхронный последовательный обмен между МК и другими ИС, установленными на плате изделия. Эти интегральные схемы дополняют функции периферии микропроцессорной системы, которые не могут быть реализованы средствами встроенных модулей МК. Именно поэтому в названии этого интерфейса присутствует термин «периферийный». Интерфейс SPI также используют для связи двух МК, однако такое решение встречается в разработках не столь часто, как обмен с периферийными ИС.

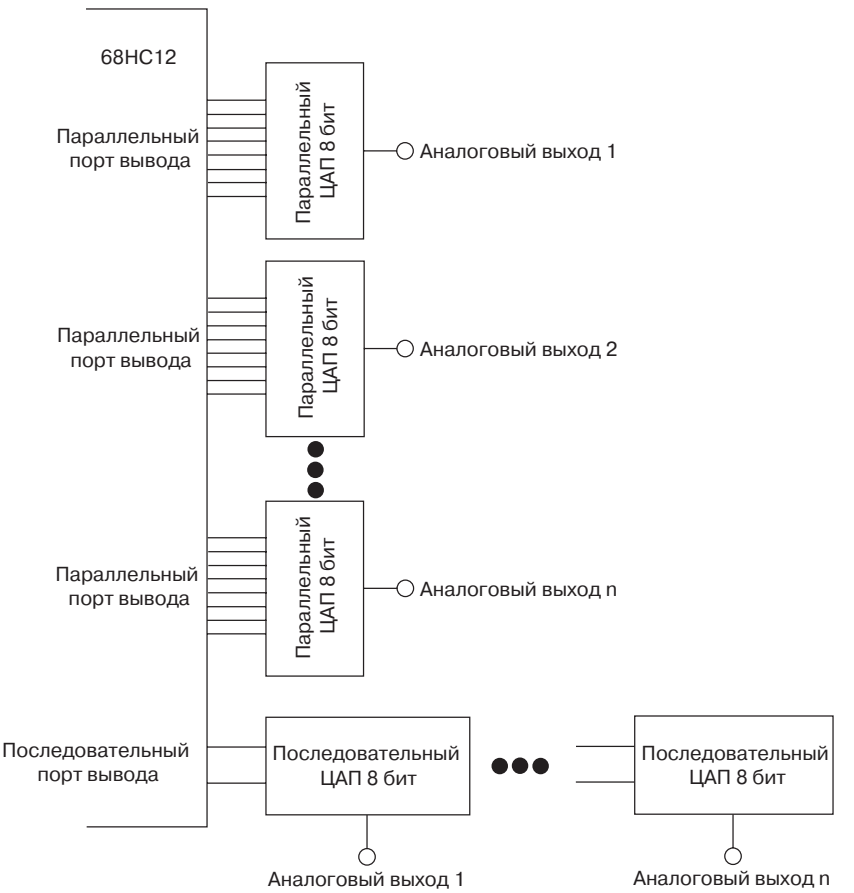


Рис. 4.55. *Сопряжение МК с ЦАП по параллельному и последовательному интерфейсу*

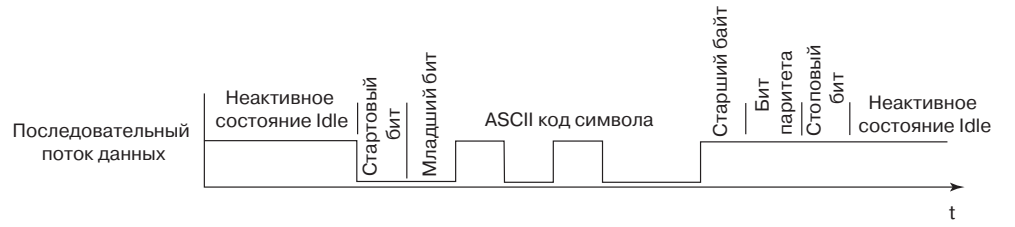


Рис. 4.56. *Форма кадра асинхронного обмена*

При обмене в синхронном режиме в одном направлении (рис. 4.54) используются две линии связи между устройствами. По одной линии передаются данные, на другой в это время формируются сигналы синхронизации. То устройство, которое формирует импульсы синхронизации, называется ведущим или «master», а то устройство, которое использует эти сигналы синхронизации – ведомым или «slave».

Теоретически скорость обмена в синхронном режиме значительно превышает скорость обмена в асинхронном режиме.

При обмене между двумя устройствами информация передается порциями, которые в терминологии последовательного обмена принято называть кадрами. Каждый кадр содержит определенное число бит данных и дополнительные биты, которые используются для установления надежной связи между передающим и принимающим устройствами. Одной из задач по обеспечению надежной связи является задача синхронизации обмена. Принимающее устройство должно по определенной комбинации сигналов на линиях связи распознать начало каждого нового кадра. В противном случае одна ошибка при передаче какого-либо кадра вызовет нарушение приема всех последующих кадров сеанса связи.

Интерфейсы последовательного обмена используют различные способы синхронизации обмена. Формат кадра асинхронного обмена представлен на рис. 4.56. В асинхронном режиме с использованием интерфейса SCI данные передаются байтами в стандартном американском коде ASCII (American Standard Code for Information Interchange). Кроме восьми битов данных кадр обмена содержит также стартовый и стоповый биты, отмечающие начало и конец кадра, и необязательный бит паритета. Стартовый и стоповый биты участвуют в процессе синхронизации между передатчиком и приемником, бит паритета (если он присутствует в кадре) используется для контроля на стороне приемника за наличием ошибок в принятом байте данных.

Контроллер интерфейса SCI в составе МК 68HC12 обслуживают две линии ввода/вывода: TxD – выход передатчика контроллера SCI, RxD – вход приемника контроллера SCI. При использовании асинхронного интерфейса SCI взаимодействующие устройства перед сеансом обмена должны обязательно «договориться» о скорости передачи данных. Если передатчик находится в неактивном состоянии (Idle), то на его выходе устанавливается сигнал высокого логического уровня (рис. 4.56). В это время приемник аналогичного контроллера SCI на другом конце линии постоянно сканирует уровень сигнала на входе RxD. Если приемник обнаруживает, что сигнал на линии изменил состояние с 1 на 0, то он производит несколько контрольных выборок сигнала, чтобы убедиться в наличии на линии низкого логического уровня. Если низкий уровень присутствует на линии в течение времени, равного интервалу передачи одного бита для установленной скорости обмена, то приемник распознает такое состояние как старт-бит (рис. 4.56) и начинает прием последую-

щих бит данных. В процессе приема аппаратные средства приемника формируют метки времени, которые должны соответствовать середине интервала присутствия на линии каждого бита. По каждой метке производятся три выборки уровня сигнала на линии. По результатам выборки методом мажоритарной логики определяется значение очередного принятого бита информации. Если все три значения равны, то прием бита полностью успешный. Если значения разные, то аппаратные средства приемника устанавливают бит звона на линии, который затем может быть использован программистом при оценке надежности приема. После приема восьми (при использовании бита паритета девяти) бит данных приемник контролирует наличие на линии логической 1. Это стоп-бит, который завершает прием одного кадра. Если аппаратные средства приемника не обнаружили на линии стоп-бита, будет установлен бит ошибки формата кадра. Тогда весь кадр должен быть воспринят программой как ошибочный.

В синхронных последовательных интерфейсах передача по линии каждого бита сопровождается сигналом подтверждения по другой вспомогательной линии. Эту линию в интерфейсе SPI обозначают как SCK (Shift Clock). Такой способ передачи данных позволяет достичь очень высоких скоростей обмена. Именно он и используется в контроллерах SPI в составе МК 68HC12.

4.17.1. Термины последовательного обмена

На рис. 4.57 представлена обобщенная временная диаграмма обмена в последовательном коде. Эта диаграмма позволит нам познакомиться с терминами, которые принято использовать при описании последовательного обмена.

Синхронизация (clock):

Сигнал, который определяет скорость обмена данными в последовательных синхронных интерфейсах. Как следует из рис. 4.57, каждый бит передаваемых данных сопровождается одним импульсом синхронизации.

Скорость обмена (bit rate):

Число бит, которые передаются по линии в одну секунду (бит/с). Скорость обмена в бит/с равна частоте сигнала синхронизации в Гц.

Скорость обмена (baut rate):

Число бит, которые передаются по линии в одну секунду, выраженная в бодах. 1 бод = 1 бит/с.

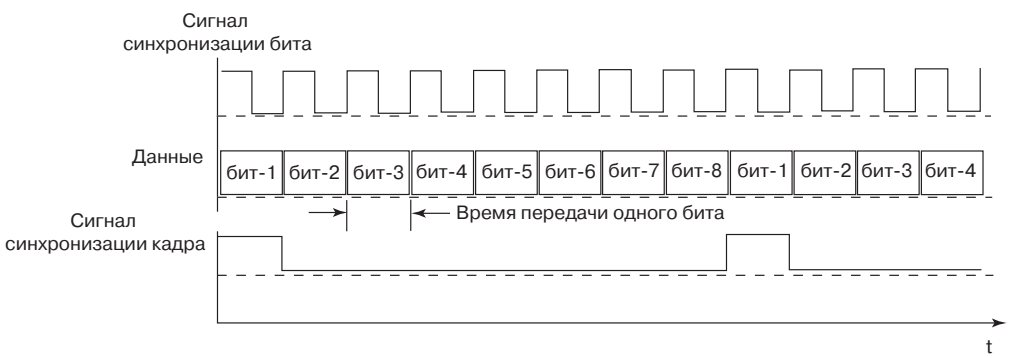


Рис. 4.57. *Временные диаграммы синхронного последовательного обмена*

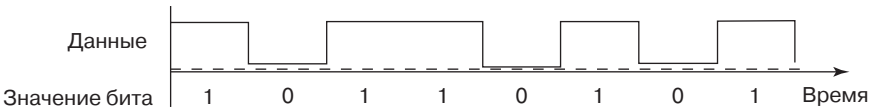


Рис. 4.58. *Временные диаграммы передачи данных в коде с невозвращением к нулю*

Время передачи одного бита (bit time):

Определяет временной интервал, в течение которого по линии передается один бит информации. Вычисляется как 1/(скорость обмена).

Кодирование информации для передачи одного бита (line code):

Способ представления одного бита информации при передаче линиям связи. Микроконтроллеры 68HC12 испоьзуют NRZ кодирование (NonReturn-to Zero – код с невозвращением к нулю). Этот способ кодирования предполагает, что при передаче единицы на линию выставляется высокий логический уровень, при передаче 0 – низкий логический уровень.

Стандартный американский код ASCII (American Standard Code for Information Interchange):

Принятый всеми производителями вычислительной техники способ кодирования букв и цифр, а также знакой пунктуации. Каждый из перечисленных символов представляется одним байтом. При кодировании букв латинского алфавита исполуются только 7 младших битов, старший бит байта остается нулевым. В этот бит может быть помещен бит паритета. Таблица кодов представления символов латинского алфавита приведена на рис. 4.59.

		Старший полубайт							
		$0_	$1_	$2_	$3_	$4_	$5_	$6_	$7_
Младший полубайт	$_0	NUL	DLE	SP	0	@	P	`	p
	$_1	SON	DC1	!	1	A	Q	a	q
	$_2	STX	DC2	“	2	B	R	b	r
	$_3	ETX	DC3	#	3	C	S	c	s
	$_4	EOT	DC4	$	4	D	T	d	t
	$_5	ENQ	NAK	%	5	E	U	e	u
	$_6	ACK	SYN	&	6	F	V	f	v
	$_7	BEL	ETB	‘	7	G	W	g	w
	$_8	BS	CAN	(	8	H	X	h	x
	$_9	HT	EM	)	9	I	Y	i	y
	$_A	LF	SUB	*	:	J	Z	j	z
	$_B	VT	ESC	+	;	K	[	k	{
	$_C	FF	FS	,	<	L	\	l	|
	$_D	CR	GS	–	=	M	]	m	}
	$_E	SO	RS	.	>	N	^	n	~
	$_F	SI	US	/	?	O	_	o	DEL

Рис. 4.59. *Коды символов в ASCII*

Бит паритета (parity bit):

Бит паритета используется для выявления одиночных ошибок при передаче одного байта информации. При использовании четного паритета значение бита равно 0, если число единичных бит в передаваемом байте является числом четным. Иначе бит паритета устанавливается равным 1, так, чтобы число единиц в байте стало четным. При использовании нечетного паритета значение бита равно 0, если число единичных бит в передаваемом байте является числом нечетным. Иначе бит паритета устанавливается равным 1, так, чтобы число единиц в байте стало нечетным.

Симплексный обмен:

При симплексном режиме обмена возможна лишь однонаправленная передача информации от одного устройства к другому.

Полудуплексный обмен:

При полудуплексномм режиме обмена в каждый момент времени возможна лишь однонаправленная передача информации от одного устройства к другому. Но в другой момент времени направление передави может быть изменено на противоположное.

Дупрлексный обмен:

При дуплексномм режиме обмена в каждый момент времени осуществляет двунаправленная передача информации между двумя устройствами.

Вопросы для самопроверки

1. Приведите коды ASCII для выражения «B32-EVB». Для ответа воспользуйтесь таблицей рис. 4.59.
   Ответ: SCI$42 $33 $32 $2D $45 $56 $42
2. Если в задании предыдущего вопроса в старший бит каждого кода поставить значение бита паритета, то какой станет кодовая последовательность для выражения «B32-EVB». Для ответа используйте четный бит паритета.
   Ответ: $42 $33 $B2 $2D $C5 $56 $42
3. В приведенной кодовой строке использован нечетный паритет. Бит паритета находится в разряде D7 кода. Какое слово зашифровано в строке $C1, $F7, $E5, $73, $EF, $6D, $E5, $A1 ?
   Ответ: Awesome!
4. Сравните SCI и SPI интерфейсы.
   Ответ: Интерфейсы SCI и SPI – это интерфейсы для последовательной передачи данных. Интерфейс SCI использует асинхронный способ передачи данных, при котором дополнительный сигнал синхронизации не используется. Вместо него применяются специальные биты синхронизации (старт- и стоп-бит). Напротив, интерфейс SPI использует дополнительную линию синхронизации.

4.18. Контроллер асинхронного обмена SCI

Различные модели МК семейства 68HC12 и HCS12 могут интегрировать на кристалле сразу несколько интерфейсов для последовательного асинхронного обмена. Однако увеличение портов асинхронного обмена не сопровождается изменением аппаратных средств контроллра SCI. На кристалле МК просто размещают несколько полностью идентичных одноканальных контроллеров SCI, различая их порядковыми номерами: SCI0, SCI1 и т.д.

Основные технические характеристики контроллера асинхронного обмена (модуля SCI) в составе МК семейства 68HC12:

- Обеспечивает полнодуплексный асинхронный режим обмена, при котором прием и передача данных могут происходить одновременно.
- Использует NRZ – кодирование, при котором для передачи единицы на линию выставляется высокий логический уровень, для передачи 0 – низкий логический уровень.
- Реализует широкий диапазон скоростей приема и передачи данных. Для задания скорости используются два регистра скорости обмена SCxBDH и SCxBDL (x – номер контроллера SCI в составе МК).
- Обеспечивает два стандартных кадра обмена в асинхронном режиме: 10-битовый (8 бит данных) и 11-битовый (9 бит данных) формат. Выбор формата кадра обмена определяет бит M в регистре управления.
- Обладает независимыми аппаратными средствами приемника (Transmitter) и передатчика (Receiver). Каждое из устройств имеет собственный бит разрешения работы: TE и RE соответственно.
- Приемник модуля SCI имеет специальный режим ожидания, который позволяет организовать локальную сеть в мультипроцессорных системах. В таких системах на основе асинхронного интерфейса одно устройство является ведущим, а все остальные – ведомыми. В каждый момент времени может происходить обмен между ведущим и одним из ведомых. Остальные ведомые при этом не должны воспринимать сигналы на общей линии связи. Это достигается путем перевода приемника контроллера SCI в состояние ожидания «Seep Mode». Перевод приемника в это состояние осуществляется установкой бита RWU в регистре управления. Выход из состояния ожидания может происходить по двум сценариям. По первому сценарию аппаратные средства приемника должны распознать отсутствие обмена на линии связи (состояние Idle). Это состояние характеризуется наличием высокого логического уровня на линии в течение 10 или 11 интервалов передачи бита при назначенной скорости обмена. Если приемник обнаружил состояние Idle на линии, то это означает, что сеанс обмена с другим ведомым в сети окончен, ведущий может начать новый сеанс, поэтому ведомый должен стать активным, чтобы не пропустить обращение ведущего к нему. По второму сценарию ведущий посылает первый кадр обмена со специальным маркером, который информирует приемник о том, что в кадре указан адрес устройства, с которым ведущий будет производить сеанс связи. Аппаратные средства приемника реагируют на этот маркер в режиме ожидания, и при его поступлении переходят в активный режим работы. Бит WAKE в регистре управления модулем определяет выбор сценария для перевода приемника в активный режим работы.
- Аппаратные средства контроллера устанавливают четыре флага, которые могут генерировать запросы на прерывание от контроллера SCI.

1. TDRE – бит готовности буфера передатчика к приему новых данных. Устанавливается в момент, когда предварительно загруженные в регистр буфера передатчика данные автоматически переписываются в сдвиговый регистр передатчика.
2. TC – бит завершения передачи данных. Устанавливается, если данные для передачи в сдвиговом и буферном регистре данных передатчика отсутствуют. Бит TC информирует МК об отсутствии процесса передачи данных в текущий момент времени. В это время на линии TxD установлен высокий логический уровень сигнала (состояние IDLE).

3. RDRF – бит завершения приема байта данных. Устанавливается в момент, когда принятые по линии RxD данные автоматически переписываются в буферный регистр данных приемника.
4. IDLE – бит неактивного состояния линии связхи. Устанавливается в 1, если на линии RxD диагностируются 10 или 11 (в зависимости от формата кадра) последовательных единиц.

- Аппаратные средства приемника диагностируют три типа ошибок:

1. Наличие шума на линии RxD. Диагностируется в случае, если при выборке очередного бита информационного кадра, включая стартовый и стоповый биты, не все три детектированные значения бита оказались равными. При обнаружении этого типа ошибки в регистре состояния устанавливается бит NF.
2. Нарушение формата принимаемого кадра. Диагностируется, если поступающая на вход RxD последовательность битов не соответствует меткам синхронизации, формируемым внутренним счетчиком приемника. Аппаратные средства приемника распознают состояние нарушения синхронизации по признаку наличия на линии нулевого логического уровня в то время, когда должен присутствовать стоповый бит с высоким логическим уровнем сигнала. При обнаружении этого типа ошибки в регистре состояния устанавливается бит FE.
3. Нарушение логики паритета кадра. Диагностируется, если функция паритета при обмене разрешена, и в принятом кадре значение бита паритета не удовлетворяет принятой логике формирования паритета: при назначенном нечетном паритете число единиц в слове четное, и наоборот. При обнаружении этого типа ошибки в регистре состояния устанавливается бит PF.

Теперь, когда Вы познакомились с общими характеристиками контроллера асинхронного обмена SCI, следует перейти к изучению его аппаратных средств и программно-логической модели. Это позволит Вам разрабатывать программны управления для контроллера SCI. А пока несколько вопросов.

Вопросы для самопроверки

1. Каково назначение бита паритета?
   Ответ: Бит паритета используется для обнаружения ошибок передачи данных. При использовании всего одного бита паритета может быть обнаружена только однократная ошибка. Причем исправить эту ошибку на стороне принимающего устройства при помощи бита паритета невозможно. Однако существуют такие способы кодирования, при которых с помощью некоторого дополнительного числа бит возможно как обнаружить ошибку, так и исправить ее.
2. По какому правилу определяется значения бита паритета? Поясните разницу между четным и нечетным паритетом.
   Ответ: Существенной разницы нет. Правило формирования бита паритета:
   - В передаваемом слове данных подсчитывается число единиц;
   - Если это число четное (при четном паритете) или нечетное (при нечетном паритете), то бит паритета устанавливается равным 0;
   - Наоборот, если число нечетное (при четном паритете) или четное (при нечетном паритете), то бит паритета устанавливается равным 1.

4.18.1. Передатчик контроллера SCI

Функциональная схема передатчика в составе контроллера SCI представлена на рис. 4.60. Основным ее элементом является 11-разрядный сдвиговый регистр. Ранее, рассматривая временные диаграммы обмена, Вы могли заметить, что код символа ASCII содержит всего 7 бит. Зачем тогда передатчик использует 11-разрядный регистр? Ело в том, что в разряды этого регистра аппаратными средствами вписывается стартовый и стоповый биты. Кроме того, предусмотрена возможность обмена и 8-разрядными словами данных. В этом случае бит паритета уже невозможно будет поместить на место разряда D7, и для него предусмотрели разряд D8 сдвигового регистра. Аппаратные средства передатчика предусматривают возможность формирования как 10-разрядного кадра обмена (8 бит данных, старт и стоп биты), так и 11-разрядного (9 бит данных, старт и стоп биты). Выбор формата кадра обмена определяет бит M в регистре управления.

Аппаратные средства передатчика модуля SCI в составе МК 68HC12 предусматривают возможность автоматического (без специальных команд программы) формирования бита паритета по содержимому передаваемого слова данных.

Блок управления формирует все необходимые сигналы для корректной работы аппаратных средств передатчика. Работа передатчика разрешается независимо от состояния приемника в составе этого же контроллера SCI. Если бит TE в регистре управления установлен, то работа передатчика активизируется, и соответствующий вывод порта PORT S конфигурируется как выход TxD. Блок управления также генерирует все запросы на прерывания, связанные с работой передатчика.

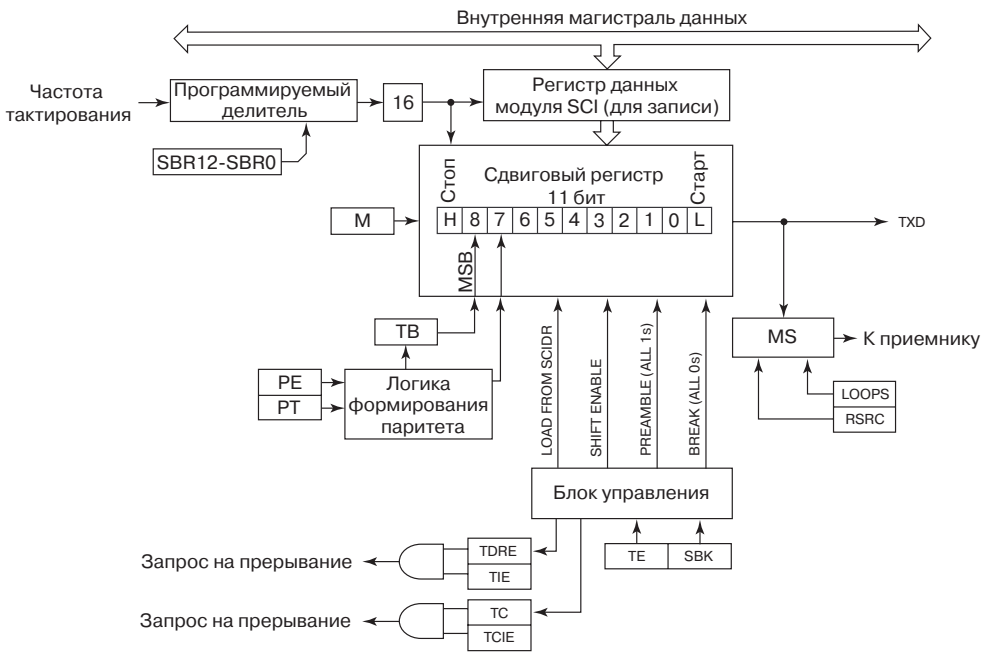


Рис. 4.60. *Аппаратные средства блока передатчика в составе контроллера асинхронного обмена*

Скорость передачи данных определяется программируемым делителем, на вход которого поступает импульсная последовательность P_CLOCK системы тактирования МК. Коэффициент деления назначается битами SBR12...SBR0 регистра скорости обмена.

Байт данных, подлежащий передаче, должен быть записан с использованием любой команды пересылки в регистр данных контроллера SCI. Если сдвиговый регистр в момент записи не занят, то байт данных будет автоматически перемещен из регистра данных в сдвиговый регистр. При этом бит паритета, стартовый и стоповый биты будут подставлены автоматически. Далее без дополнительных команд программы управления начнется передачасформированного в сдвиговом регистре кадра обмена на выход TxD.

4.18.2. Приемник контроллера SCI

Функциональная схема приемника в составе контроллера SCI представлена на рис. 4.61. Так же, как и в приемнике, основным ее элементом является 11-разрядный сдвиговый регистр. Однако это уже собственный регистр приемника, поскольку контроллер SCI обеспечивает возможность обмена в двух направлениях одновременно. Напртив, программируемый делитель является общим для приемника и передатчика. Поэтому скорость приема и передачи данных для одного и того же контроллера SCI всегда одинакова.

Работа приемника разрешается установкой бита PE в регистре управления. Если приемник активирован, то соответствующая линия порта PORT S конфигурируется как вход RxD. Блок мажоритарной логики орбрабатывает входной сигнал и формирует очередной младший бит сдвигового регистра приемника. В процессе приема каждый поступающий бит опрашивается три раза. Если все три выборки совпали (три 1 или три 0), то соответствующее значение передается в сдвиговый регистр. При несовпадении (два бита равны 1, третий – 0, или наоборот) значение бита формируется по логике «два из трех». Но одновременно вустанавливается бит наличия шума на линии NF.

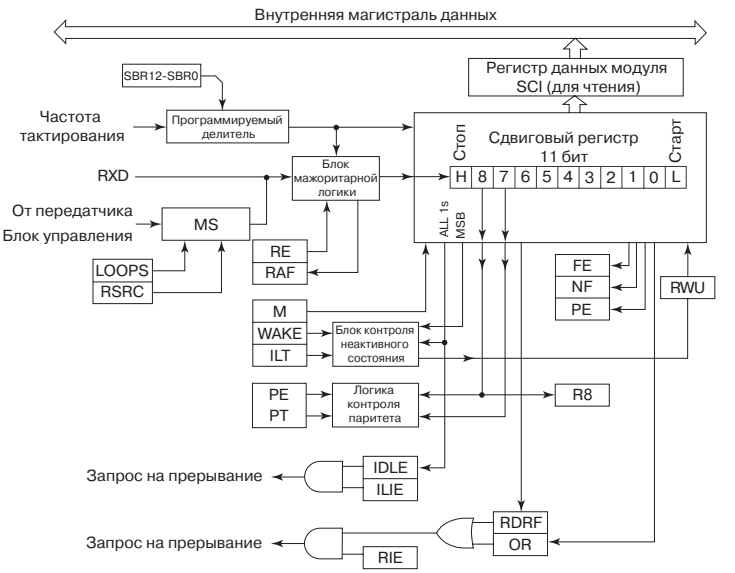


Рис. 4.61. *Аппаратные средства блока приемника в составе контроллера асинхронного обмена*

По завершении приема всех 10 или 11 бит кадра приемник автоматически переписывает принятую информацию в регистр данных контроллера SCI и устанавливает в 1 флаг RDRF, который информирует МК о необходимости считавания принятого байта в память МК.

4.18.3. Регистры контроллера SCI

Контроллер асинхронного обмена обслуживается несколькими группами регистров специальных функций:

- Регистры скорости обмена;
- Регистры управления;
- Регистры состояния;
- Регистры данных.

Далее мы рассмотрим формат и назначение битов каждого регистра модуля SCI.

Регистры скорости обмена SCxBDH и SCxBDL

Два 8-разрядных регистра SCxBDH и SCxBDL предназначены для управления скоростью обмена по последовательному интерфейсу SCI. Для назначения желаемой скорости используются разряды SBR12...SBR0 этих двух регистров (рис. 4.63). Число, которое следует записать в эти разряды может быть определено с использованием таблицы рис. 4.62 или рассчитано по формуле:

$$SBR = PCLOCK/(16 \times BAUD\_RATE),$$

где SBR – десятичный эквивалент двоичного кода, который должен быть записан в разряды SBR12...SBR0 регистров SCxBDH и SCxBDL, PCLOCK – частота импульсной последовательности PCLK в Герцах, BAUD_RATE – частота обмена в бодах.

Желаемая скорость обмена (бод)	Значение коэффициента при частоте внутренней системной шины	
	4 МГц	8 МГц
110	2273	4545
300	833	1667
600	417	833
1200	208	417
2400	104	208
4800	52	104
9600	26	52
14400	17	35
19200	13	26
38400	–	13

Рис. 4.62. *Выбор коэффициента деления модуля SCI*

Имя регистра: SCxBDH Адрес: $00C0

	7	6	5	4	3	2	1	0
	BTST	BSPL	BRLD	SBR12	SBR11	SBR10	SBR9	SBR8
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: SCxBDL Адрес: $00C1

	7	6	5	4	3	2	1	0
	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
Сброс:	0	0	0	0	0	1	0	0

Рис. 4.63. *Формат регистра скорости обмена SCxBDH/SCxBDL*

Пример.

Необходимо настроить первый контроллер асинхронного обмена (SCI0) на скорость 9600 бод. Если частота импульсной последовательности PCLK равна 8 МГц, то коэффициент деления, который должен быть записан в регистры SC0BDH и SC0BDL, равен:

$$SBR0 = 8000000/(16 \times 9600) = 52 = \$34$$

Следующий программный фрагмент реализует инициализацию скорости обмена:

```
/*-------------------------------------------------------------------*/
/* filename: ini_SCI0.c                                              */
/*-------------------------------------------------------------------*/
#include<912b32.h>

void main(void)
{
SC0BDH = 0x00;
SC0BDL = 0x34;
}
/*-------------------------------------------------------------------*/
```

Регистры управления SCxCR1 и SCxCR2

Формат первого регистра управления контроллера SCI представлен на рис. 4.64. Биты этого регистра имеет следующее назначение:

LOOPS:

Бит разрешения «замкнутого» режима работы контроллера SCI. Установка в 1 бита LOOPS вызывает перекоммутацию входа приемника линии RxD, который внутренними средствами отсоединяется от вывода RxD и подсоединяется к выходу передатчика. В этом режиме возможен контроль передаваемой информации. Также режим может быть использован для тестирования работы программного обеспечения без использования устройства управления верхнего уровня. Для реализации «замкнутого» режима должна быть разрешена работа как передатчика, так и приемника.

1 – «замкнутый» режим работы разрешен;

0 – «замкнутый» режим работы запрещен.

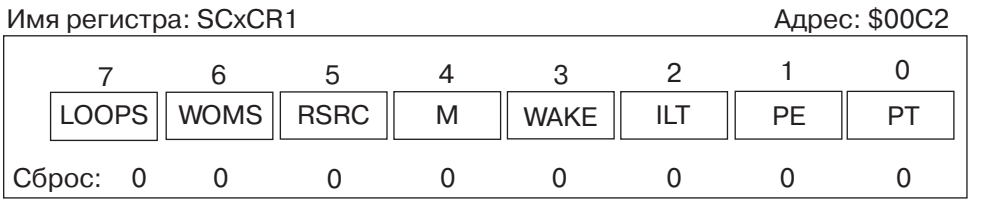


Рис. 4.64. Формат регистра управления SCxCR1

WOMS:

Бит выбора режима открытого коллектора. Этот бит определяет состояние выходных буферов линий TxD и RxD.

1 – буферы переведены в режим открытого коллекторного выхода;

0 – буферы работают в режиме обычного двухстабильного логического выхода (TxD) и входа (RxD).

Перевод линий TxD и RxD в режим открытого коллектора позволяет соединить их по схеме «монтажное И», что делает возможным двусторонний обмен по одной линии. Кроме того, при такой конфигурации возможно создание системы с несколькими передающими устройствами.

RSRC:

Бит выбора внутренней схемотехники в «замкнутом» режиме работы. Если бит LOOPS = 1 и бит RSRC = 1 то вход приемника коммутируется непосредственно к выводу TxD микроконтроллера. При RSRC = 0 вход приемника подсоединяется к выходу передатчика внутри МК.

M:

Бит выбора формата кадра асинхронного обмена.

1 – 11-битовый формат кадра: 1 стартовый бит, 9 бит слова данных, 1 стоповый бит;

0 – 10-битовый формат кадра: 1 стартовый бит, 8 бит слова данных, 1 стоповый бит.

WAKE:

Бит выбора способа выхода приемника из неактивного состояния:

1 – установка маркера адреса (бит D7 при M=0 или бит D8 при M=1)) переводит приемник в активный режим работы;

0 – состояние IDLE на линии переводит приемник в активное состояние.

После сброса МК бит устанавливается в 0.

ILT:

Бит выбора режима распознавания неактивного состояния линии RxD. Этот бит определяет момент начала отсчета для определения неактивного состояния линии **RxD:**

1 – отсчет начинается после идентификации стоп-бита;

0 – отсчет начинается после идентификации старт-бита.

После сброса МК бит устанавливается в 0.

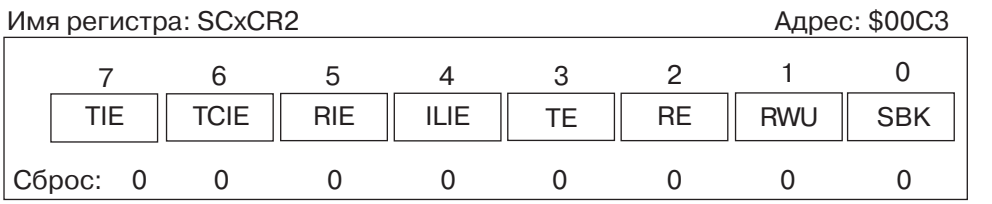


Рис. 4.65. *Формат регистра управления SCxCR2*

PE:
Бит разрешения работы логики паритета.
1 – формирование бита паритета передатчиком и его анализ приемником реализуются;
0 – функция паритета отключена.
PT:
Бит выбора четного или нечетного паритета (Parity Bit)
1 – бит паритета формируется из условия нечетного числа 1 в слове;
0 – бит паритета формируется из условия четного числа 1 в слове.

Формат второго регистра управления контроллера SCI приведен на рис. 4.65. Четыре старших бита этого регистра (TIE, TCIE, RIE, ILIE) разрешают генерацию запросов на прерывания по разным событиям контроллера SCI. Назначение отдельных битов регистра SCxCR2 следующее:

TIE:
Бит разрешения прерывания от передатчика контроллера SCI. Этот бит разрешает генерацию запроса на прерывание при установке в 1 флага готовности буфера данных передатчика к приему от программы нового байта (бит TDRE).
1 – прерывания от передатчика по флагу SCDE разрешены
0 – прерывания от передатчика по флагу SCDEE запрещены.
TCIE:
Бит разрешения прерывания от передатчика контроллера SCI. Этот бит разрешает генерацию запроса на прерывание при установке в 1 флага завершения работы передатчика TC.
1 – прерывания от передатчика по флагу TC разрешены;
0 – прерывания от передатчика по флагу TC запрещены.
RIE:
Бит разрешения прерывания от приемника контроллера SCI. Этот бит разрешает генерацию запроса на прерывание при установке в 1 флага завершения приема очередного байта RDRF.
1 – прерывания от приемника по флагу SCRF разрешены;
0 – прерывания от приемника по флагу SCRF запрещены.
ILIE:
Бит разрешения прерывания от приемника по флагу IDLE. Этот бит разрешает генерацию запроса на прерывание при установке в 1 флага неактивного состояния линии RxD.
1 – прерывания при установленном флаге IDLE разрешены;
0 – прерывания по флагу IDLE запрещены.
TE:
Бит разрешения работы передатчика контроллера SCI. Если бит TE будет сброшен в процессе передачи, то передача текущего байта будет завершена.
1 – передача разрешена;
0 – передача запрещена.
RE:
Бит разрешения работы приемника контроллера SCI.
1 – прием разрешен;
0 – прием запрещен.

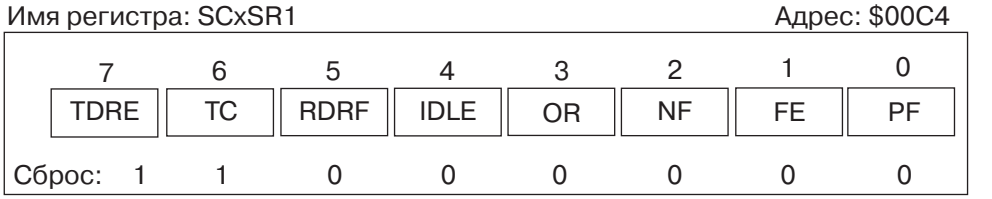


Рис. 4.66. *Формат регистра состояния SCxSR1*

RWU:

Бит управления режимом ожидания приемника контроллера SCI. Установка бита RWU под управлением программы в 1 переводит приемник контроллера SCI в режим ожидания. Пока приемник находится в этом режиме, ни один из флагов, которые связаны с работой приемника (RDRF, IDLE, OR, NF, FE, PE), не может быть установлен. Однако те флаги, которые уже были установлены к моменту перевода приемника в режим ожидания, не сбрасываются в момент записи 1 в бит RWU. Способ перевода приемника в активный режим работы определяет бит WAKE в регистре SCxCR1.

SBK:

Бит управления сообщением «конец сеанса обмена». Если бит SBK установить под управлением программы в 1, то передатчик контроллера SCI генерирует в линию TxD последовательность из 10 (бит M=0) или 11 (бит M=1) нулевых битов и одного единичного бита.

Регистры состояния SCxSR1 и SCxSR2

Формат первого регистра состояния контроллера SCI представлен на рис. 4.66. Старшие четыре бита регистра SCxSR1 содержат флаги событий, которые используются в штатных режимах работы приемника и передатчика. Младшие четыре бита отражают тип зафиксированной ошибки приема. Все флаги, связанные с работой приемника сбрасываются после выполнения операции чтения младшего байта регистра данных приемника. Назначение отдельных битов регистра SCxSR1 следующее:

TDRE:

Бит готовности буфера передатчика к приему новых данных. Устанавливается в момент, когда предварительно загруженные в регистр буфера передатчика данные автоматически переписываются в сдвиговый регистр передатчика.

TC:

Бит завершения передачи данных. Устанавливается, если данные для передачи в сдвиговом и буферном регистре данных передатчика отсутствуют, а также, если не реализуется режим передачи сообщения «конец сеанса обмена». Бит TC информирует МК об отсутствии процесса передачи данных в текущий момент времени. В это время на линии TxD установлен высокий логический уровень сигнала (состояние IDLE). Бит TC вызывает генерацию запроса на прерывание, если бит TCIE в регистре SCxCR2 установлен.

RDRF:

Бит завершения приема байта данных. Устанавливается в момент, когда принятые по линии RxD данные автоматически переписываются в буферный регистр данных приемника. Бит RDRF вызывает генерацию запроса на прерывание, если бит RIE в регистре SCxCR2 установлен.

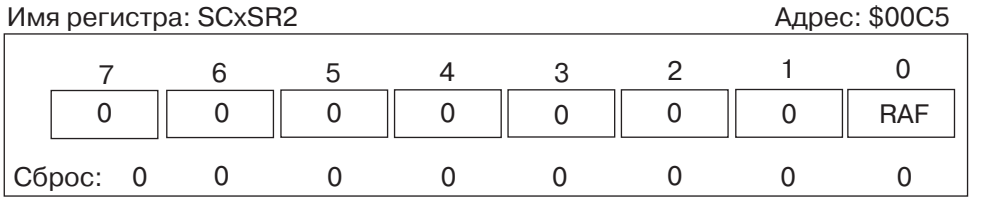


Рис. 4.67. *Формат регистра состояния SCxSR2*

IDLE:

Бит неактивного состояния линии RxD. Устанавливается в 1, если на линии RxD диагностируются 10 или 11 (в зависимости от формата кадра) последовательных единиц. Бит IDLE вызывает генерацию запроса на прерывание, если бит ILIE в регистре SCxCR2 установлен.

OR:

Бит ошибки приема. Устанавливается при попытке записи аппаратными средствами приемника очередного принятого байта из сдвигового регистра в буферный регистр данных в то время, как предыдущие данные из буферного регистра еще не считаны (бит RDRF установлен).

NF:

Бит наличия шума на линии приемника RxD. Устанавливается в случае, если при выборке очередного бита информационного кадра, включая стартовый и стоповый биты, не все три детектированные значения бита оказались равными. Бит NF устанавливается одновременно с битом завершения приема того байта, при передаче которого обнаружен шум.

FE:

Бит нарушения формата кадра. Устанавливается, если поступающая на вход RxD последовательность битов не соответствует меткам синхронизации, формируемым внутренним счетчиком приемника. Аппаратные средства приемника распознают состояние нарушения синхронизации по признаку наличия на линии нулевого логического уровня в то время, когда должен присутствовать стоповый бит с высоким логическим уровнем сигнала.

PF:

Бит нарушения паритета кадра. Устанавливается, если функция паритета при обмене разрешена, и в принятом кадре значение бита паритета не удовлетворяет принятой логике формирования паритета: при назначенном нечетном паритете число единиц в слове четное, и наоборот.

Формат второго регистра состояния контроллера SCI представлен на рис. 4.67. Регистр SCxSR2 содержит всего один бит RAF. Этот бит автоматически устанавливается в 1 в то время, когда в сдвиговом регистре приемника продолжается формирование очередного принимаемого байта данных.

Регистры данных SCxDRH и SCxDRL

Формат двух 8-разрядных регистров данных контроллера SCI представлен на рис. 4.68. Регистр SCxDRL используется, если контроллер SCI настроен на обмен данными в 8-разрядном формате (10-битовый формат кадра). Тогда по адресу регистра SCxDRL записываются данные для передачи, и из этого же регистра считываются принятые данные. Если контроллер SCI настроен на обмен данными в 9-разрядном формате (11-битовый формат кадра), то старший бит для передачи записы-

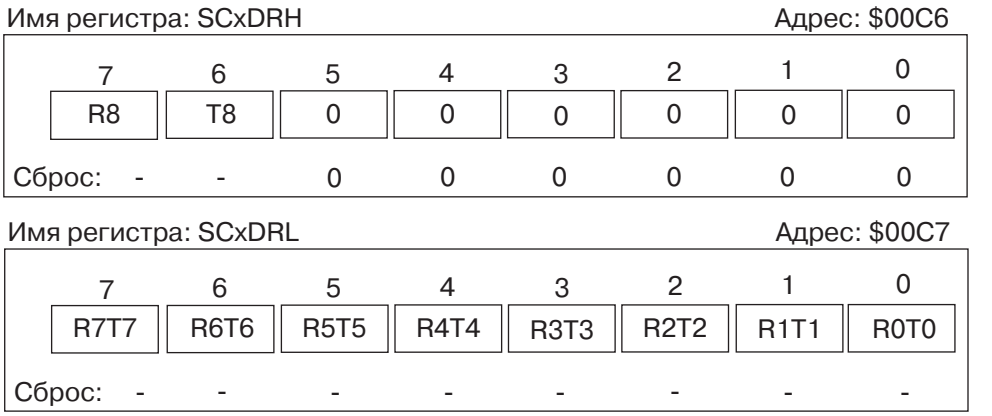


Рис. 4.68. *Формат регистра данных SCxDRH/SCxDRL*

вается под управлением программы в разряд T8 регистра SCxDRH, а при приеме из разряда R8 считывается старший 9-ый бит принятого слова. Младшие восемь разрядов при передаче и при приеме, как и в случае 8-разрядного слова данных, находятся в регистре SCxDRL.

Вопросы для самопроверки

1. Каково различие между битами TDRE и RDRF?
   Ответ: Оба бита являются флагами, которые отражают состояние контроллера SCI. Флаг TDRE устанавливается в 1, когда регистр данных передатчика автоматически переписывается в сдвиговый регистр. В этом случае говорят, что регистр данных передатчика пуст. Флаг RDRF устанавливается в 1, когда процесс приема данных в сдвиговый регистр приемника закончен, и очередное слово данных готовы к считыванию.
2. Каково назначение флага PF? Почему он чрезвычайно важен при обмене информацией в последовательном коде?
   Ответ: Флаг нарушения паритета кадра PF устанавливается, когда произошла ошибка в приеме одного или нескольких бит одного кадра. Прикладная программа контролирует бит PF, как индикатор неверного приема, и должна повторить обмен.
3. Опишите события, которые могут генерировать прерывания от модуля SCI.
   Ответ: Четыре источника прерываний ассоциируются с модулем SCI:
   - Прерывание по флагу TDRE, когда регистр данных приемника пуст. Это прерывание разрешается битом TIE в регистре управления SCxCR2;
   - Прерывание по флагу TC, когда передача слова закончена. Это прерывание разрешается битом TCIE в регистре управления SCxCR2;
   - Прерывание по флагу RDRF, когда прием очередного слова завершен. Это прерывание разрешается битом RIE в регистре управления SCxCR2;
   - Прерывание по флагу IDLE, когда приемник находится в неактивном состоянии. Это прерывание разрешается битом ILIE в регистре управления SCxCR2.

4.18.4. Алгоритмы программного обслуживания контроллера SCI

Три относительно независимых последовательности действий должна выполнить прикладная программы в процессе использования контроллера асинхронного обмена SCI:

- Инициализацию приемника и передатчика контроллера;
- Управление процессом передачи информации;
- Управление процессом приема информации.

Обобщенные блок-схемы алгоритмов управления программно-доступными ресурсами контроллера SCI для каждого из перечисленных действий приведены на рис. 4.69.

Инициализация контроллера SCI. Инициализацию контроллера SCI рекомендуется проводить в следующем порядке:

- Установить скорость обмена;
- Выбрать формат кадра обмена: 8 или 9 бит данных;
- Назначить параметры приема и передачи в регистрах управления SCxCR1 и SCxCR2;
- Очистить флаг TDRE. Для этого сначала считать регистр состояния SCxSR1, а затем выполнить операцию записи в регистр данных SCxRD.

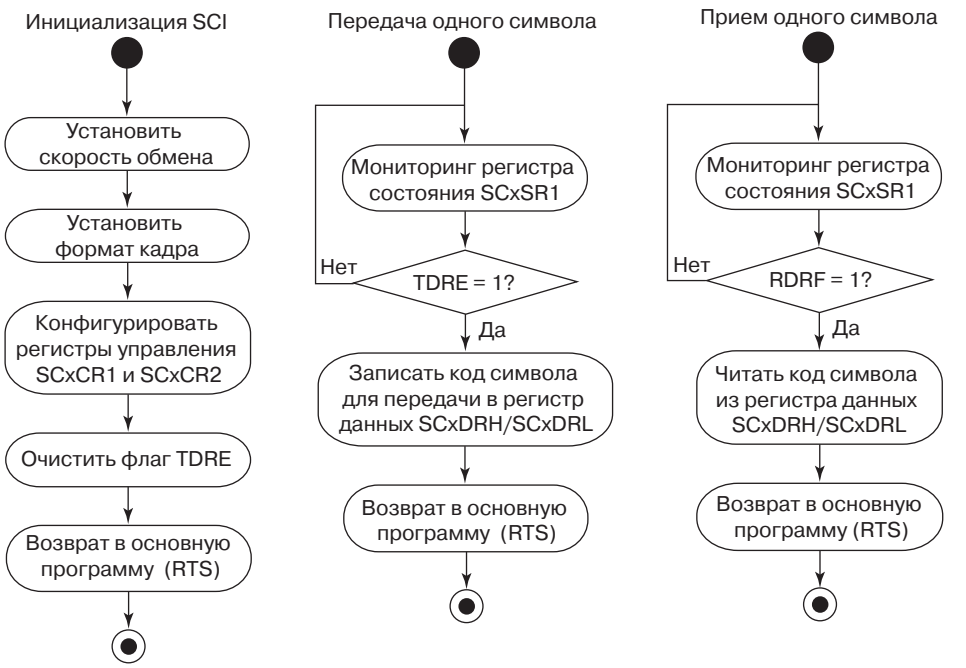


Рис. 4.69. *Блок-схемы алгоритмов программного обслуживания контроллера асинхронного обмена SCI*

Следует заметить, что аппаратные средства модуля SCI предполагают, что и прием, и передача информации могут происходить только с одинаковой скоростью, с одинаковым форматом кадра и одинаковой логикой паритета.

Управление процессом передачи информации. Для того чтобы передать один байт информации с использованием передатчика контроллера SCI, следует сначала проверить состояние флага TDRE. Если этот флаг установлен в 1, то регистр данных передатчика пуст, и в него может быть записан новый байт для передачи. Далее данные загружаются в один регистр данных SCxDRL, если обмен происходи в 8-разрядном формате, или в два регистра данных SCxDRH: SCxDRL, если обмен производится в 9-разрядном формате. Коды из регистра данных загружаются в сдвиговый регистр передатчика автоматически, после чего начинается собственно процесс передачи. О завершении передачи информирует бит TC, который установится в 1, когда все разряды регистра сдвига будут последовательно выданы на выход TxD. Флаги TDRE и TC могут генерировать запросы на прерывания, если соответствующие биты разрешения прерывания установлены.

Управление процессом приема информации. При приеме информации с использованием приемника контроллера SCI, следует постоянно контролировать состояние флага RDRF. Этот флаг устанавливается в 1, когда прием очередного байта закончен, и принятые данные доступны в регистре данных приемника. Бит может генерировать запрос на прерывание, при условии, что эти прерывания разрешены. Обнаружив в процессе мониторинга флага RDRF или прервавшись по его запросу, МК должен считать данные из регистров SCxDRH: SCxDRL.

4.18.5. Пример программирования контроллера SCI

Приведенный ниже программный фрагмент SCI.c иллюстрирует технику программного обслуживания асинхронного последовательного интерфейса МК семейства 68HC12.

Отладочная плата MC68HC912B32EVB предоставляет возможность использования только одного контроллера SCI с номером 0. Поэтому в именах регистров специальных функций символ «x» будет заменен нами на символ «0».

Ниже перечислены биты и регистры управления, которые используются в данном примере:

- SC0BDH: SC0BDL – регистры скорости обмена контроллера SCI. Записанное в него двоичное число определяет скорость передачи данных и скорость приема данных, которые для одного контроллера в соответствии с его принципом действия могут быть только равными;
- SC0CR1 – первый регистр управления контроллера SCI. Используется для выбора формата 8-ми или 9-ти разрядного представления слова в кадре обмена данными (бит M), для выбора режима работы с паритетом или без него (бит PE), для назначения четной или нечетной логики формирования паритета (бит PT);
- SC0CR2 – второй регистр управления контроллера SCI. Его биты разрешают работу передатчика (бит TE) и приемника (бит RE);
- SC0DRL – регистр данных контроллера SPI, младший байт. Используется для обмена данными в последовательном коде в 8-разрядном формате;
- SC0SR1 – первый регистр состояния контроллера SCI. Этот регистр содержит флаг готовности буфера передатчика к приему новых данных TDRE и флаг завершения приема очередного слова в буфер приемника RDRF.

В нашем примере задействован только передатчик контроллера SCI. Он будет непрерывно посылать по линии последовательной связи TxD (вывод PORTS0) код символа «S». Прерывания от контролера SCI в данном примере не используются. Контроль за состоянием флага TDRE ведется методом полинга.

```
/*-------------------------------------------------------------------*/
/* filename: SCI.c                                                   */
/* MAIN PROGRAM: Эта программа реализует непрерывную посылку кода    */
/* символа "S"с скоростью 9600 бод в 8-разрядном формате (кадр 10 бит)*/
/* с битом паритета                                                  */
/*-------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>

/*используемые функции*/
void sci_init(void);
void sci_trans(void);

void main(void)
{
sci_init ();              /*инициализация модуля SCI*/
while (1)
      {
      sci_trans ();       /*передавать данные непрерывно*/
      }
}
/*-------------------------------------------------------------------*/
/* Функция sci_init производит инициализацию модуля SCI.             */
/*-------------------------------------------------------------------*/
void sci_init(void)
{
unsigned char clear;

SC0BDL = 0x34;      /*установить скорость 9600 бод*/
SC0BDH = 0x00;
SC0CR1 = 0x04;      /*10-разрядный формат кадра - 8 бит данных, с*/
                    /*контролем паритета, логика паритета - нечетная */
clear = SC0SR1;     /*операция для сброса флага TDRE*/
                    /*флаг сбрасывается в два действии*/
                    /*сначала читать регистр SC0SR1*/
                    /*затем записать в регистр SC0DRL*/
}
/*-------------------------------------------------------------------*/
/* Функция sci_trans осуществляет непрерывную пересылку одного байта  */
/*-------------------------------------------------------------------*/
void sci_trans (void)
{
SC0CR2 = 0x08;            /*разрешить работу передатчика*/
SC0DRL = 's';       /*загрузить в буфер передатчика код символа "S"*/
while (SC0SR1 != 0x80)    /*ожидать установления бита TDRE)
      {
      ;
      }
}
/*-------------------------------------------------------------------*/
```

Если соединить два микропроцессорных контроллера по последовательному интерфейсу так, что выход TxD первого МК будет соединен со входом RxD второго МК и наоборот, то можно будет организовать двусторонний обмен информацией. Именно эту задачу Вам предстоит решить в домашнем задании №11, полный текст которого Вы найдете в конце главы.

4.19. Синхронный последовательный интерфейс SPI

Интерфейс SPI относится к группе синхронных последовательных интерфейсов. Принцип действия интерфейсов этого типа предполагает, что передача каждого бита данных по однопроводной линии сопровождается сигналом синхронизации, который передается по другой линии. Спецификация интерфейса SPI определяет число линий связи и временные диаграммы сигналов на этих линиях в процессе обмена данными.

4.19.1 Концепция интерфейса SPI

Функциональная схема организации обмена данными с использованием интерфейса SPI представлена на рис. 4.70. При описании режимов работы и линий связи интерфейса SPI приняты к использованию следующие термины и обозначения:

- ***Master Mode*** – режим ведущего. Устройство, работающее в режиме ведущего, начинает сеанс обмена, генерирует передаваемые данные в ведомое устройство и формирует сигналы синхронизации, сопровождающие эти данные.
- ***Slave Mode*** – режим ведомого. Устройство, работающее в режиме ведомого, получает сигналы синхронизации от ведущего. В момент поступления импульса синхронизации ведомое устройство запоминает очередной бит 8-разрядного слова, переданный ведущим по одной линии данных, и выставляет на другую линию данных очередной бит другого 8-разрядного слова, передаваемого от ведомого к ведущему.

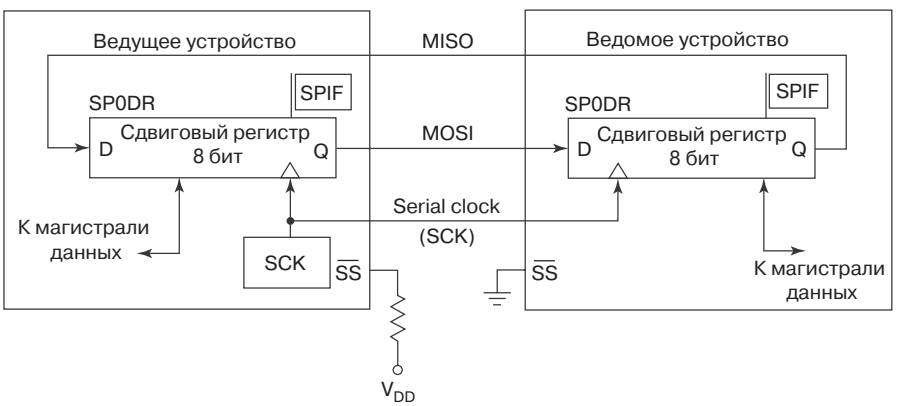


Рис. 4.70. *Функциональная схема обмена между двумя контроллерами SPI*

- ***MOSI (Master Output Slave Input)*** – линия передачи данных от ведущего к ведомому.
- ***MISO (Master Input Slave Output)*** – линия передачи данных от ведомого к ведущему.
- ***SCK (Serial Shift Clock)*** – линия сигнала синхронизации данных.
- ***SS (Slave select)*** – линия сигнала выбора ведомого устройства. По этой линии ведущее устройство «сообщает» ведомому о начале сеанса обмена с ним.

4.19.2. Алгоритмы работы контроллера SPI

В данном разделе мы рассмотрим организацию обмена данными с использованием аппаратных средств контроллера SPI в составе МК 68HC12. Мы также кратко остановимся на регистрах управления контроллера SPI. Более подробно эти регистры будут рассмотрены в следующем параграфе.

На рис. 4.71 представлена функциональная схема контроллера SPI. Остановимся сначала на системе синхронизации, которая располагается в левом верхнем углу рисунка. Модуль SPI использует в качестве источника тактирования импульсную последовательность PCLOCK, частота которой равна частоте ECLOCK (т.е. частоте системной шины МК). Частота PCLOCK делится делителем с программируемым коэффициентом. На выходе делителя формируется сигнал, частота которого определяет скорость обмена данными по SPI, если контроллер работает в ведущем режиме. Коэффициент деления назначается битами SPR2...SPR0 регистра скорости передачи SP0BR.

Данные, которые подлежат передаче, а также принятые от другого устройства данные, записываются в регистр данных SP0DR. Управление режимами работы контроллера осуществляется двумя регистрами управления SP0CR1 и SP0CR2. Текущее состояние процесса передачи данных отражает регистр состояния SP0SR. И, наконец, контроллер взаимодействует с внешним миром по четырем линиям порта S (сигналы MOSI, MISO, SCK и $\overline{SS}$).

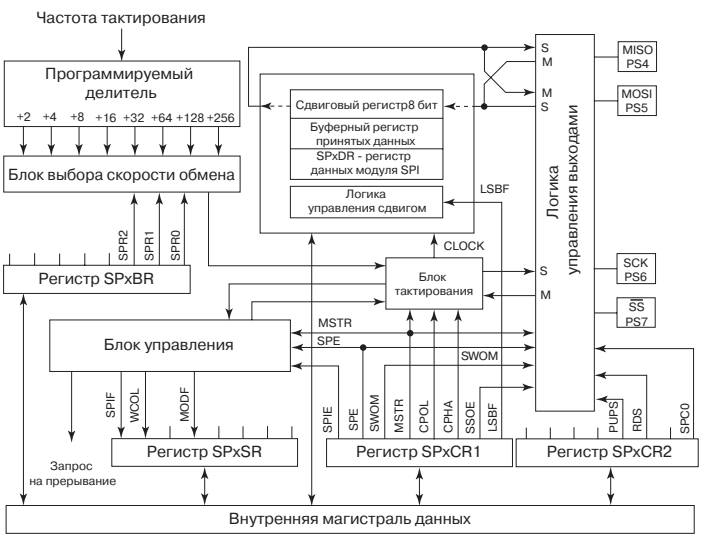


Рис. 4.71. *Аппаратные средства контроллера синхронного обмена SPI*

Если два устройства связаны по SPI, то 8-разрядный регистр данных ведущего устройства и 8-разрядный регистр данных ведомого устройства образуют вместе 16-разрядный кольцевой сдвиговый регистр (рис. 4.70). В процессе передачи код 16-разрядного регистра сдвигается под действием импульсов синхронизации SCK. В ответ на каждый импульс SCK один двоичный разряд выдвигается из регистра данных ведущего на линию MOSI и запоминается в первом слева разряде регистра ведомого (см. рис. 4.70). В то же время крайний правый бит регистра данных ведомого выдвигается на линию MISO и запоминается в регистре ведущего. В результате, по прошествии восьми импульсов синхронизации SCK код 8-разрядного регистра данных ведущего переместится в регистр данных ведомого, а 8-разрядный код ведомого – в регистр данных ведущего. Таким образом, в процессе обмена ведущее и ведомое устройства поменяются содержимым их регистров данных.

Для того, чтобы начать обмен данными по SPI, необходимо выполнить некоторую последовательность действий по инициализации контроллера SPI. Во-первых, порт S должен быть конфигурирован соответствующим образом с использованием регистра направления передачи порта DDRS. Далее следует установить коэффициент деления частоты PCLOCK. Возможные значения коэффициента деления: 2. 4. 8. 16. 32. 64. 128 и 256. Желаемый коэффициент деления устанавливается разрядами SPR2...SPR0 регистра скорости передачи SP0BR. Для сигнала синхронизации SCK следует определить не только частоту, но и форму сигнала. По форме сигнала различают четыре типа сигналов синхронизации SCK, которые отличаются полярностью и сдвигом фазы по отношению к сигналам на линиях MOSI и MISO. Выбор формы сигнала SCK определяется комбинацией битов CPOL:CPHA в регистре управления SP0CR1.

Выбрав скорость обмена, следует определить значения битов управляющих регистров SP0CR1 и SP0CR2. Так бит MTSR регистра управления SP0CR1 ведущего должен быть установлен в 1, а такой же бит в регистре управления ведомого – в 0. Тогда первому контроллеру будет назначен режим ведущего, а второму – режим ведомого. И они составят пару для обмена. Кроме того, для организации обмена необходимо правильно установить уровни сигналов на входах $\overline{SS}$ обоих контроллеров. Если в системе всего два устройства связаны по шине SPI, то вход $\overline{SS}$ ведущего должен быть установлен в 1, а аналогичный вход ведомого в 0 (рис. 4.70). Если в системе к шине SPI подключено несколько ведомых устройств, то вход выбора ведомого $\overline{SS}$ каждого устройства должен устанавливаться в 0 ведущим только тогда, когда ведущий обменивается данными именно с этим устройством. Во время обмена данным с другими ведомыми устройствами вход $\overline{SS}$ неактивного ведомого должен находиться в 1.

После того, как все биты регистров управления SP0CR1 и SP0CR2 установлены в соответствии с выбранным режимом работы, следует разрешить работу контроллера SPI. Для этого предназначен бит SPE регистра SP0CR1. Разрешение работы ведущего контроллера SPI должно быть выполнено ранее ведомого.

Если все операции по инициализации модулей SPI ведущего и ведомого МК завершены, то можно приступить к непосредственно к обмену данными. Начало обмена инициирует ведущий. Для этого необходимо под управлением программы записать пересылаемый байт информации в регистр данных SP0DR модуля SPI. Если сдвиговый регистр модуля в момент записи оказался пустым, то данные немедленно автоматически перемещаются из регистра данных в сдвиговый регистр. Далее аппаратные средства контроллера SPI формируют восемь импульсов синхронизации SCK. Каждый импульс SCK сдвигает один двоичный разряд регистра данных ведущего по линии MOSI в регистр данных ведомого. Одновременно другой разряд регистра данных ведомого по линии MISO вдвигается в регистр данных ведущего. По истечении восьми импульсов

SCK пересылка одного байта заканчивается, и устанавливаются флаги SPIF в регистре состояния обоих контроллеров. В ведущем МК флаг SPIF сигнализирует о завершении передачи одного байта, в то время, как в ведомом МК этот флаг информирует о завершении приема байта данных. Если пересылка следующих байтов не предполагается, то вход $\overline{SS}$ ведомого должен быть установлен в 1, что переводит контроллер SPI ведомого устройства в неактивное состояние. Рассмотренный способ обмена характеризуется одновременным перемещением данных от ведущего к ведомому и в обратном направлении. Такой способ обмена называют полнодуплексным.

Аппаратные средства контроллера SPI могут генерировать запросы на прерывание. Два источника запросов располагаются в регистре состояния SP0SR. Первый источник – триггер завершения обмена SPIF, второй источник – флаг нарушения режима MODF. Оба прерывания разрешаются установкой в 1 бита SPIE в регистре управления SP0CR1.

Вопросы для самопроверки

1. Каковы различия между двумя режимами работы контроллера SPI: режимом ведущего и режимом ведомого?
   Ответ: Контроллер SPI, работающий в режиме ведущего, начинает обмен и генерирует импульсы синхронизации SCK для обмена. Таким образом, ведущий контроллер управляет обменом. Ведомы контроле SPI ожидает сигналов от ведущего, и под их управлением запоминает информацию с линии MOSI, а также генерирует информацию на линию MISO. Завершается обмен под управлением ведущего.
2. Каково назначение сигнала SCK?
   Ответ: Сигнал SCK предназначается для синхронизации передачи информации между двумя устройствами. Частота этого сигнала определяет скорость передачи. В течение одного периода SCK два устройства обмениваются одним битом данных.

4.19.3. Регистры контроллера SPI

Подобно контроллеру асинхронного обмена контроллер SPI обслуживается несколькими регистрами специальных функций:

- Регистр скорости обмена;
- Регистры управления;
- Регистр состояния;
- Регистр данных.

Далее мы рассмотрим формат и назначение битов каждого регистра модуля SPI.

Регистр скорости обмена SPxBR

Регистр скорости обмена SPxBR позволяет выбрать частоту следования импульсов синхронизации SCK, а, следовательно, и скорость обмена по синхронному последовательному интерфейсу. Формат регистра SPxBR представлен на рис. 4.72. Три бита этого регистра SPR2...SPR0 определяют коэффициент деления импульсной последовательности ECLOCK, из которой образуется сигнал синхронизации SCK. Соответствие численных значений коэффициентов деления возможным комбинациям битов SPR2...SPR0 устанавливает таблица рис. 4.72. Внимательно проанализируйте данные этой таблицы. Вспомните, что при тех же частотах системной шины МК, максимальная скорость обмена в асинхронном режиме с использованием контроллера SCI составляла 38400 бод, что для SPI эквивалентно частоте SCK в 38,4 кГц. А для контроллера SPI максимальная частота синхронизации, а, следовательно, и максимальная скорость обмена составляет 4,0 МГц.

Регистры управления SPxCR1 и SPxCR2

Формат первого регистра управления SPxCR1 контроллера SPI представлен на рис. 4.72. Биты этого регистра имеет следующее назначение:

SPIE:

Бит разрешения прерывания по запросу модуля SPI. Бит разрешает генерацию запросов на прерывание от модуля SPI. Запросы в модуле SPI могут генерироваться при установленном флаге SPIF, который свидетельствует о завершении приема или передачи одного байта информации, или при установленном флаге ошибки MODF.

1 – прерывания разрешены;

0 – прерывания по запросу приемника запрещены.

SPE:

Бит разрешения работы модуля SPI.

1 – контроллер SPI включен;

0 – контроллер SPI выключен.

SWOM:

Бит выбора режима открытого коллектора. Этот бит определяет состояние выходных буферов линий MOSI, MISO, SCK, если эти линии работают на вывод.

1 – буферы переведены в режим открытого коллекторного выхода;

0 – буферы работают в режиме двунаправленной передачи с возможностью установки в высокоимпедансное состояние.

Перевод линий MOSI и MISO в режим открытого коллектора позволяет соединить их по схеме «монтажное И».

MSTR:

Бит режима работы контроллера SPI.

1 – контроллер SPI работает в режиме ведущего (Master);

0 – контроллер SPI работает в режиме ведомого (Slave).

CPOL:

Бит выбора полярности сигнала синхронизации SCK.

Этот бит определяет состояние линии SCK между сеансами передачи данных. Бит CPOL вместе с битом CPHA задает один из четырех возможных режимов SPI интерфейса.

1 – SCK=1 между сеансами передачи данных

0 – SCK=0 между сеансами передачи данных

CPHA:

Бит выбора фазы сигнала синхронизации SCK. Этот бит определяет протокол обмена по SPI шине. Если CPHA=0, то начало обмена инициируется установкой сигнала выбора ведомого $\overline{SS}$ в активное состояние (режимы 0 и 1). Первый перепад сигнала синхронизации SCK используется принимающим устройством для запоминания очередного бита в сдвиговом регистре. Передающее устройство выставляет очередной бит посылки на линии MOSI по каждому четному фронту сигнала SCK. Сигнал на линии выбора ведущего $\overline{SS}$ должен быть возвращен в неактивное состояние после передачи каждого байта в любом направлении. Режимы 0 и 1 предпочтительно использовать в системах, которые имеют более одного ведомого устройства.

Если CPHA=1, то начало обмена определяет первое изменение уровня сигнала на линии SCK после установки сигнала выбора ведомого $\overline{SS}$ в активное состояние (режимы 2 и 3). Все нечетные перепады SCK вызывают выдвижение очередного бита посылки из сдвигового регистра передатчика на линию. Каждый четный перепад используется для записи этого бита в сдвиговый регистр приемника. Сигнал выбора ведомого может оставаться в активном состоянии $\overline{SS}$ = 0 в течение передачи нескольких байт информации. Режимы 2 и 3 рекомендуется использовать в системах с одним ведомым устройством.

Выбор частоты обмена по SPI

SPR[2:0]	Коэффициент деления	При частоте внутренней системной шины	
		4 МГц	8 МГц
000	2	2,0 МГц	4,0 МГц
001	4	1,0 МГц	2,0 МГц
010	8	500 кГц	1,0 МГц
011	16	250 кГц	500 кГц
100	32	125 кГц	250 кГц
101	64	62,5 кГц	125 кГц
110	128	31,3 кГц	62,5 кГц
111	256	15,6кГц	31,3 кГц

Имя регистра: SPxBR Адрес: $00D2

7	6	5	4	3	2	1	0
0	0	0	0	0	SPR2	SPR1	SPR0
Сброс: 0	0	0	0	0	0	0	0

Имя регистра: SPxCR1 Адрес: $00D0

7	6	5	4	3	2	1	0
SPIE	SPE	SWOM	MSTR	CPOL	CPHA	SSOE	LSBF
Сброс: 0	0	0	0	0	1	0	0

Имя регистра: SPxCR2 Адрес: $00D1

7	6	5	4	3	2	1	0
0	0	0	0	PUPS	RDS	0	SPC0
Сброс: 0	0	0	0	1	0	0	0

Рис. 4.72. *Формат регистров управления контроллера SPI*

SSOE:

Бит разрешения работы вывода в режиме ведущего. Если контроллер SPI работает в режиме ведущего, то при установке этого бита в 1 вывод может использоваться как вывод для формирования сигнала «выбор ведомого»

1 – функция выхода $\overline{SS}$ разрешена;

0 – функция выхода $\overline{SS}$ не реализуется.

LSBF:

Бит выбора очередности выдачи битов данных на линию MOSI. Если бит LSBF сброшен, то данные в процессе передачи выставляются на линию MOSI, начиная со старшего бита. Этот режим считается нормальным режимом обмена для интерфейса SPI. Если же бит LSBF установлен в 1, то при передаче первым на линию MOSI выдается младший бит. Некоторые периферийные ИС требуют такого режима обмена.

Формат второго регистра управления SPxCR2 контроллера SPI также приведен на рис. 4.72. Этот регистр содержит всего три бита управления:

PUPS:

Бит выбора схемотехники входных буферов порта S. Если этот бит равен 1, то все линии порта S, которые находятся в режиме ввода, переходят в состояние входа с внутренним подтягивающим к напряжению питания резистором.

RDS:

Бит выбора схемотехники выходных буферов порта S. Если этот бит равен 1, то все линии порта S, которые находятся в режиме вывода, переходят в состояние выхода с пониженной нагрузочной способностью.

SPCO:

Совместно с битом MSTR этот бит определяет конфигурацию входов/выходов контроллера SPI на выводы корпуса МК в соответствие с таблицей рис. 4.73.

Познакомившись с назначением отдельных битов регистров управления SPxCR1 и SPxCR2, можно сказать, что эти регистры определяют конфигурацию аппаратных средств контроллера SPI. Разъясним дополнительно функции битов MSTR регистра SPxCR1 и SPCO регистра SPxCR2. Возможные комбинации кодов этих двух битов и соответствующие им внутренние соединения линии ввода и линии вывода контроллера SPI показаны на рис. 4.73. Например, если значения рассматриваемых битов равны SPCO:MSTR = 01, то вывод MISO будет выполнять функцию ввода данных, а вывод MOSI – функцию вывода данных контроллера SPI, работающего в режиме ведущего. Линия SCK при этом значении кодовой комбинации битов SPCO:MSTR будет использоваться для выдачи импульсов синхронизации, а линия $\overline{SS}$ может быть дополнительно сконфигурирована на ввод или на вывод. Примечания 1-5 на рис 4.73 содержат дополнительную информацию по конфигурированию линий порта S для обслуживания контроллера SPI записи 0 или 1 в соответствующие разряды регистра направления передачи DDRS порта S. Мы рассмотрим эти рекомендации в примере программного обслуживания контроллера SPI (раздел 4.19.4).

На рис. 4.73 показано, что выходы контроллера SPI могут работать как в нормальном двухпроводном режиме (обмен одновременно идет по линиям SI и SO), так и в однопроводном режиме (обмен идет по одной линии).

Функции линий обмена контроллера SPI в различных режимах работы

Режим обмена		SPCO [1]	MTSR	MISO [2]	MOSI [3]	SCK [4]	$\overline{SS}$ [5]
#1	Нормальный двухпроводный	0	0	Выход ведомого	Вход ведомого	Вход SCK	Вход $\overline{SS}$
#2	Нормальный двухпроводный	0	1	Вход ведущего	Выход ведущего	Выход SCK	Вход/выход $\overline{SS}$
#3	Дополнительный однопроводный	1	0	Вход/выход ведомого	Вход/выход общего назначения	Вход SCK	Вход $\overline{SS}$
#4	Дополнительный однопроводный	1	1	Вход/выход общего назначения	Вход/выход ведущего	Выход SCK	Вход/выход $\overline{SS}$

1) Бит SPCO = 1 в регистре SPxCR2 разрешает однопроводный режим работы
2) Выход ведомого разрешен при DDRS4=1, SS=0, MTSR=0 (#1, #3)
3) Выход ведущего разрешен при DDRS5=1, MTSR=1 (#2, #4)
4) Выход SCK разрешен при DDRS6=1, MTSR=1 (#2, #4)
5) Выход SS разрешен при DDRS7=1, SSOE=1, MTSR=1 (#2, #4)

Выбор режима работы линии SS

DDRS	SSOE	Режим ведущего	Режим ведомого
0	0	Выход SS с контролем ошибки типа MODF	Вход $\overline{SS}$
0	1	*Зарезервирован*	Вход $\overline{SS}$
1	0	Выход общего назначения	Вход $\overline{SS}$
1	1	Выход $\overline{SS}$	Вход $\overline{SS}$

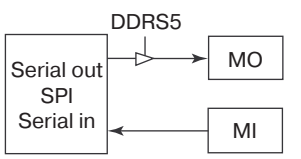
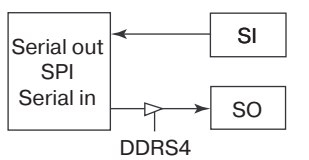
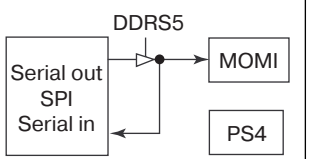
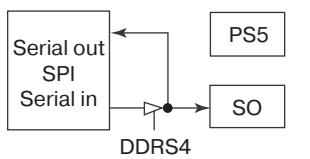
SPE = 1	Режим ведущего (MTSR = 1)	Режим ведомого (MTSR = 0)
Нормальный двухпроводный режим обмена SPCO = 0		
Двунаправленный режим обмена SPCO = 0		

Рис. 4.73. *Выбор режима работы контроллера SPI*

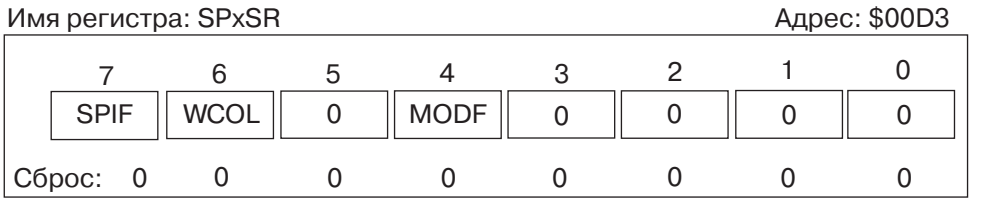


Рис. 4.74. *Формат регистра состояния контроллера SPI*

Регистр состояния SPxSR

Формат регистра состояния SPxSR контроллера SCI представлен на рис. 4.74. Регистр SPxSR содержит всего три бита флагов состояния:

SPIF:

Бит завершения обмена одним байтом данных. Устанавливается в момент, когда все восемь бит данных выставлены на линию последовательного обмена. Этот бит сигнализирует микроконтроллеру о том, что записанный ранее байт данных передан в линию и можно либо загружать новый байт для передачи, либо читать принятый в процессе передачи байт. Этот бит сбрасывается в результате выполнения двух последовательных операций. Сначала читают регистр состояния SPxSR, затем выполняют операцию чтения или записи регистра данных SP0DR.

WCOL:

Бит нарушения режима передачи данных. Этот бит устанавливается в 1, если МК пытается записать в регистр данных SPxDR новый байт для передачи в то время, когда предыдущий байт еще находится в процессе передачи.

MODF:

Бит нарушения режима контроллера SPI. Устанавливается, если на линию $\overline{SS}$ ведущего (MSTR = 1) подали сигнал низкого логического уровня. Бит MSTR = 1 назначает режим ведущего для контроллера SPI, а $\overline{SS}$ = 0 пытается перевести контроллер в режим ведомого. Такая ситуация является контфликтной, о чем и сигнализирует флаг MODF.

Регистр данных SPCxDR

Формат регистра данных контроллера SPI приведен на рис. 4.75. Особенностью этого регистра является то, что он одновременно является регистром для размещения как передаваемых, так и принимаемых данных. Эта особенность обусловлена рассмотренным ранее принципом обмена по интерфейсу SPI. В процессе обмена регистр данных передающего контроллера (этот контроллер ведущий) и регистр данных принимающего контроллера (этот контроллер ведомый) объединяются в 16-разрядный кольцевой сдвиговый регистр. По прошествии восьми импульсов синхронизации SCK 8-разрядное слово из ведущего контроллера сдвигается в регистр данных ведомого контроллера. При этом 8-разрядное слово регистра данных ведо-

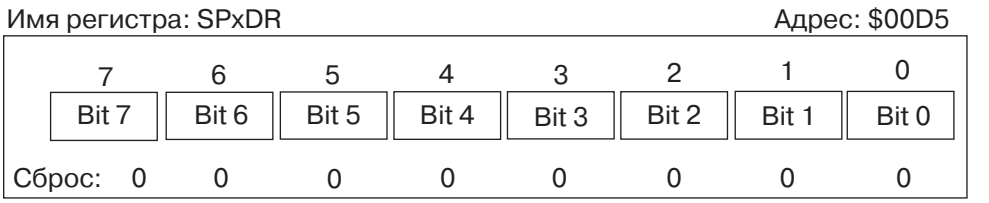


Рис. 4.75. *Формат регистра данных контроллера SPI*

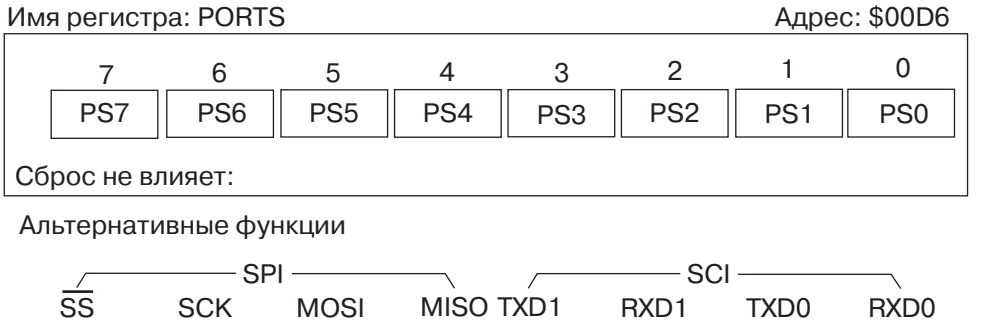


Рис. 4.76. *Формат регистра данных и альтернативные функции линий порта PORTS*

мого перемещается в регистр данных ведущего. Таким образом, независимо от того, желает программист передать данные из ведущего в ведомый или принять данные из ведомого в ведущий, в любом из этих случаев оба устройства одновременно принимают и передают данные. Поэтому до сеанса обмена регистр данных ведущего контроллера содержит подготовленные к передаче данные, а после сеанса обмена этот же регистр содержит принятые данные.

Регистр данных порта S

Порт S может работать в режиме порта ввода/вывода общего назначения. Работа каждой линии в режиме ввода или в режиме вывода определяется соответствующим разрядом регистра направления передачи DDRS. Если линия порта PSx установлена на ввод, то чтение регистра данных порта PORTS возвращает в разряде PORTSx значение сигнала на соответствующем выводе МК. Если линия PSx установлена на вывод, то запись в разряд PORTSx регистра данных PORTS 1 или 0 устанавливает на выводе МК сигнал с логическим уровнем 1 или 0. Альтернативной функцией порта S является обслуживание контроллеров асинхронного SCI и синхронного SPI последовательного обмена.

Регистр направления передачи порта S

Если порт S работает в режиме порта ввода/вывода общего назначения, то биты регистра направления передачи DDRS определяют режим ввода или режим вывода для каждой линии порта S. Если разряд DDRSx установлен в 0, то линия PSx работает в режиме ввода. При DDRSx = 1 линия PSx работает в режиме вывода.

Если активизированы контроллеры последовательных интерфейсов, то значения битов регистра DDRS определяют режим работы линий порта S по следующим правилам:

DDRS2, DDRS0. Если контроллер SCI конфигурирован для работы в обычном двухпроводном режиме, то линии порта PS2 и PS0 будут выполнять функцию входов приемников двух модулей SCI независимо от значения битов DDR2 и DDR0.

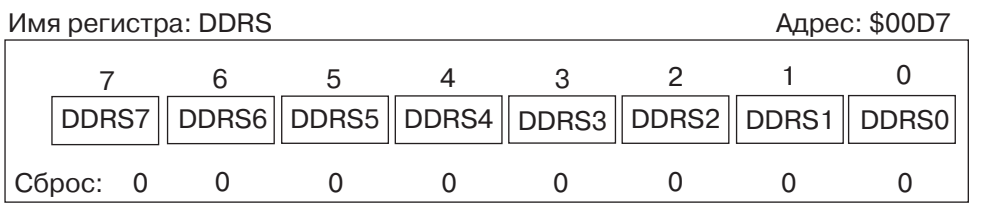


Рис. 4.77. *Формат регистра направления передачи порта PORTS*

DDRS3, DDRS1. Если контроллер SCI конфигурирован для работы в обычном двухпроводном режиме, то линии порта PS3 и PS1 будут выполнять функцию выходов передатчиков двух модулей SCI независимо от значения битов DDR3 и DDR1.

DDRS6…DDRS4. Если контроллер SPI активизирован и использует некоторые линии из PS6...PS4 для ввода данных, то эти линии будут работать в режиме ввода, независимо от значения соответствующих битов регистра DDRS. Если же по логике работы контроллера SPI некоторые линии из PS6...PS4 должны работать на вывод, то необходимо соответствующий бит регистра DDRS установить в 1.

DDRS7. Если контроллер SPI активизирован и работает в режиме ведомого, то линия PS7 будет выполнять функцию входа для выбора ведомого $\overline{SS}$, независимо от значения разряда DDR7. Если же контроллер SPI работает в режиме ведущего, то значение бита DDR7 определяет направление передачи линии PS7, независимо от того, используется ли она контролером SPI, или работает как линия ввода/вывода общего назначения (см. таблицу рис. 4.73 и примечание к ней).

Вопросы для самопроверки

1. Какова минимальная частота сигнала синхронизации SCK, если частота импульсной последовательности ECLK составляет 8 МГц?
   Ответ: При значении кодовой комбинации битов SPR2...SPR0 = 111 частота SCK составляет 31,2 кГц.
2. Какова максимальная частота сигнала синхронизации SCK, если частота импульсной последовательности ECLK составляет 4 МГц?
   Ответ: При значении кодовой комбинации битов SPR2...SPR0 = 000 частота SCK составляет 2 МГц.
3. Напишите выражение на Си для установки минимальной частоты SCK.
   Ответ: SP0BR = 0x07
4. Каково назначение флага SPIF?
   Ответ: Этот флаг устанавливается, когда контроллер SPI завершил выдачу на линию восьми бит данных. Чтобы сбросить бит SPIF необходимо сначала прочитать регистр состояния SP0SR, а затем выполнить операцию записи в регистр данных SP0DR.

4.19.4. Алгоритмы программного обслуживания контроллера SPI

Ранее в данном параграфе мы обсудили действия, которые необходимо совершить, чтобы инициировать пересылку одного байта с использованием интерфейса SPI. Эти действия отражены в блок-схемах алгоритмов, которые представлены на рис. 4.78. Далее мы покажем, как записать эти действия на Си.

Микроконтроллер и периферийные интегральные схемы. Одна из основных функций интерфейса SPI в микропроцессорных системах – обмен данными между МК и установленными на той же печатной плате интерфейсными ИС. Контроллер SPI в составе МК 68HC12 может быть инициализирован для работы как в режиме ведущего (мастера), так и в режиме ведомого (подчиненного). В процессе обмена с периферийными ИС микроконтроллер всегда является ведущим, а ИС – по умолчанию ведомой (без дополнительной инициализации). Формирование сигналов интерфейса SPI – дополнительная функция линий PS7..PS4 порта S. Сигналы SPI распределяются между линиями PS7..PS4 в следующем порядке:

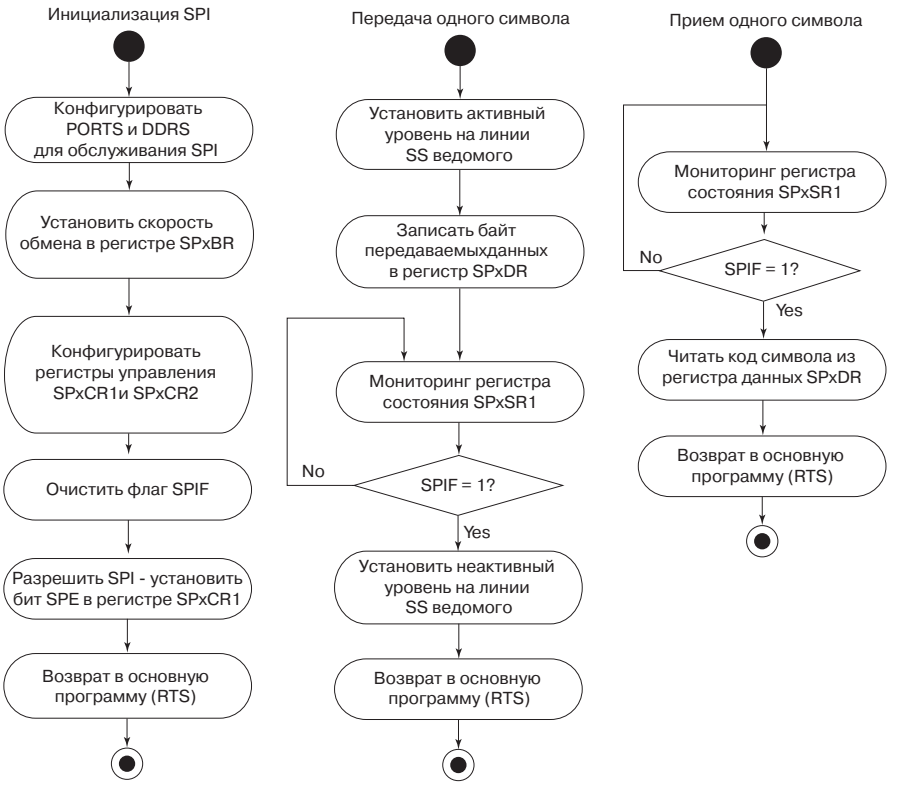


Рис. 4.78. *Блок-схемы алгоритмов программного обслуживания контроллера асинхронного обмена SPI*

- PS7 – сигнал $\overline{SS}$. Когда линия $\overline{SS}$ установлена в 0, происходит передача данных из ведущего устройства SPI;
- PS6 – сигнал SCK;
- PS5 – сигнал MOSI. По этой линии передаются данные от ведущего устройства к ведомому;
- PS4 – сигнал MISOПо этой линии передаются данные от ведомого устройства к ведущему.

Регистры управления, используемый в примере. Ниже перечислены биты и регистры управления, которые используются в данном примере:

- DDRS – регистр направления передачи порта S. Разряды DDRS7...DDRS4 должны быть установлены соответствующим образом при работе контроллера SPI в режиме ведущего. Это необходимо для правильного формирования сигналов обмена на линиях порта S;
- SP0BR – регистр скорости передачи. Разряды SPR2...SPR0 этого регистра задают скорость обмена по шине SPI;
- SP0CR1 – регистр управления (первый). Задает режимы работы контроллера SPI;

- SP0CR2 – регистр управления (второй). Задает режимы работы контроллера SPI;
- SP0SR – регистр состояния. Отражает текущее состояние процесса передачи информации по шине SPI;
- SP0DR – регистр данных. В этот регистр записываются данные, подлежащие передаче в интерфейсную ИС.

Пример программирования контроллера SPI. В нашем примере мы будем использовать встроенный контроллер SPI в режиме ведущего. Контролер будет непрерывно посылать шестнадцатеричное число $F0 в «воображаемую» периферийную ИС. Работоспособность приведенного программного кода может быть проверена с помощью осциллографа. При желании Вы можете собрать простейшую периферийную ИС – последовательный регистр со светодиодами , подключенными к параллельным выходам (рис. 4.79). В примере не использованы прерывания, контроль за состоянием флага SPIF ведется методом полинга.

```
/*-------------------------------------------------------------------*/
/* filename: SPI.c                                                   */
/* MAIN PROGRAM: Эта программа реализует непрерывную посылку кода   */
/* символа "S"с скоростью 9600 бод в 8-разрядном формате (кадр 10 бит)*/
/* с битом паритета                                                  */
/*-------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>
#include<stdio.h>

/*используемые функции*/
void initialize_spi(void);
void send_data(unsigned int);

void main(void)
{
int i, j;
unsigned int data;

initialize_spi();          /*инициализация модуля SCI*/
data = 0xF0;

while (1)                  /*передавать данные непрерывно*/
```

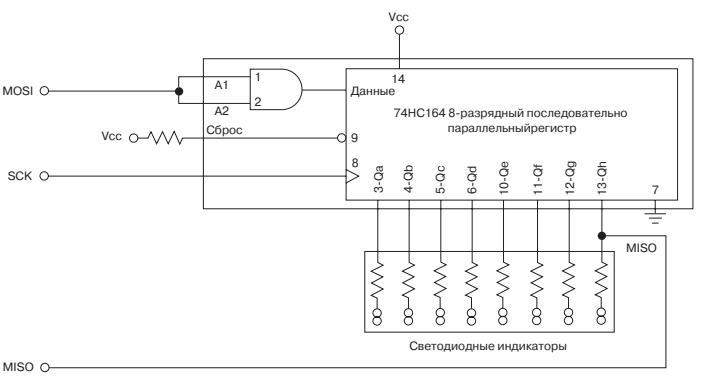


Рис. 4.79. *Функциональная схема периферийного устройства для тестирования обмена по SPI*

```
      {
      PORTS = PORTS & 0x7F; /*установить SS в 0 - активизировать ведомого*/
      send_data (data);
      PORTS = PORTS | 0x80; /*установить SS в 1 - "выключить" ведомый*/
      }
}
/*-------------------------------------------------------------------*/
/* Функция initialize_spi  производит инициализацию модуля SPI.     */
/*-------------------------------------------------------------------*/
void initialize_spi (void)
{
PORTS = 0x80;       /*установить SS - сделать ведомый неактивным*/
DDRS = 0xE0;        /*разряды 7, 6, 5 в 1 - линии PS7…PS5 на вывод*/
SP0BR = 0x04;       /*установить скорость обмена*/
SP0CR1 = 0x18;      /*запретить прерывания от SPI, назначить режим*/
                          /*ведущего, старшим битом вперед*/
SP0CR2 = 0x08;      /*PUPS = 1 - включить подтягивающий резистор */
SP0DR = 0x00;       /*очистить регистр данных */
SP0SR = 0x00;       /*очистить регистр состояния*/
SP0CR1 = 0x58;      /* разрешить SPI */
}
/*-------------------------------------------------------------------*/
/* Функция send_data производит инициализацию модуля SPI.            */
/*-------------------------------------------------------------------*/
void send_data(unsigned int data)
{
unsigned int status;

SP0DR = data;                    /*задать число для пересылки*/
while ((SP0SR&0x80) = = 0x00)    /*ожидать флага завершения передачи*/
       {
       ;
       }
status = SP0SR /*прочитать регистр состояния с целью сброса флага SPIF*/
}
/*-------------------------------------------------------------------*/
```

В приведенном примере мы показали лишь технику программирования обмена для контроллера SPI, однако мы не останавливались на особенностях подчиненного устройства, с которым происходит обмен.

4.19.5 Периферийные ИС с интерфейсом SPI

Интерфейс SPI обычно используется для расширения функциональных возможностей однокристального МК. Многие производители полупроводниковых компонентов выпускают периферийные интегральные схемы с интерфейсом SPI. По функциональному назначению эти схемы принадлежат к следующим группам устройств:

- Память типа EEPROM или FLASH;
- Дополнительные порты ввода/вывода;
- Часы реального времени;
- АЦП высокого разрешения (число разрядов преобразования превышает 8 бит);
- Драйверы светодиодных и жидкокристаллических дисплеев;
- Многоканальные ЦАП;
- Схемы фазовой автоподстройки частоты.

4.20. Введение в теорию аналого-цифрового преобразования

Встраиваемые микропроцессорные системы на основе МК семейства 68HC12 часто предназначаются для управления реальными промышленными объектами, в которых входные сигналы имеют аналоговую природу. Это сигналы различных датчиков: тока, напряжения, температуры, давления, ускорения, освещенности, загрязненности воздуха и т.д. Управляя каким либо технологическим агрегатом, МК должен обработать выходные сигналы этих датчиков, рассчитать по их значениям требуемые управляющие воздействия и сформировать необходимые управляющие сигналы для исполнительных устройств. Однако МК по своей сути является цифровым устройством, он способен преобразовывать данные только в цифровом виде. Поэтому для взаимодействия с аналоговыми датчиками микропроцессорная система должна быть оснащена аналого-цифровым преобразователем, который позволит представить аналоговые сигналы в виде цифровых кодов.

Процесс преобразования изменяющегося во времени аналогового сигнала в последовательность цифровых кодов предполагает выборку (запоминание) величины измеряемого аналогового сигнала через равноотстоящие во времени интервалы с последующим преобразованием каждого такого отсчета в цифровой код (рис. 4.80).

Известно много способов преобразования аналогового напряжения в цифровой код. Мы остановимся только на способе последовательного приближения, поскольку именно этот способ используется в модуле аналого-цифрового преобразования МК 68HC12.

Для представления некоторого изменяющего во времени аналогового сигнала в цифровом коде необходимо:

- Определить частоту дискретизации (выборки) аналогового сигнала;
- Определить необходимое число двоичных разрядов в кодовом представлении измеряемой аналоговой величины;
- Преобразовать напряжение входного сигнала в многоразрядный двоичный код.

Далее мы рассмотрим каждую из перечисленных задач более подробно.

4.20.1. Частота дискретизации сигнала

В процессе преобразования непрерывно изменяющийся аналоговый сигнал представляется конечным числом отсчетов этого сигнала, взятых в определенные моменты времени (рис. 4.70). Такой способ преобразования называют дискретизацией по времени. Как правило, моменты взятия отсчетов сигнала следуют с равными интервалами во времени. Поэтому можно говорить о частоте дискретизации сигнала или частоте выборки. Какова должна быть эта частота, чтобы на основе дискретных по времени отсчетов сигнала можно было безошибочно восстановить исходный сигнал? Итоги исследований, выполненных по этому вопросу, сформулированы в критерии Найквиста:

Минимальная частота дискретизации сигнала равна удвоенной частоте высшей гармоники в представлении исследуемого сигнала:

$$f_S \geqslant 2 f_h \text{, где}$$

f_S – частота дискретизации, f_h – частота высшей гармоники при разложении исследуемого сигнала в гармонический ряд.

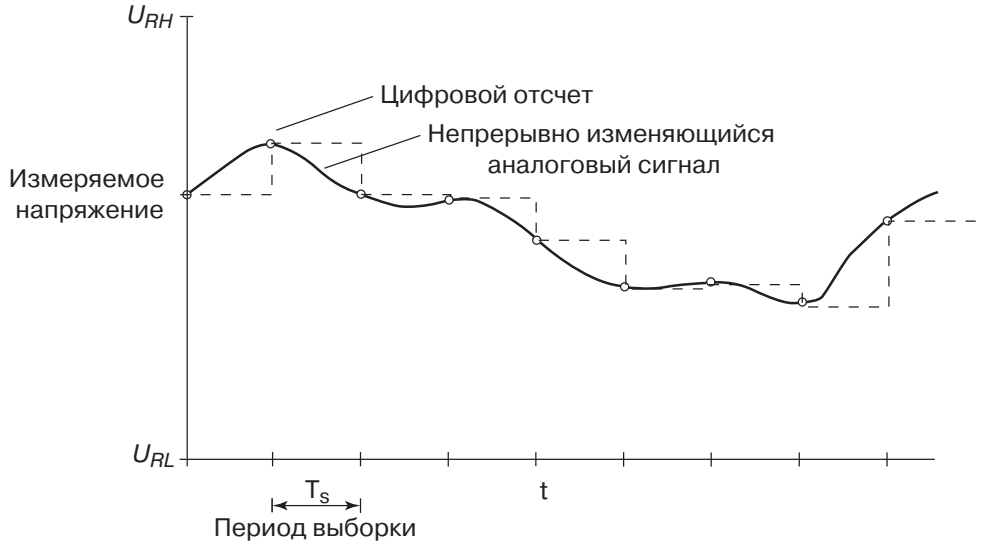


Рис. 4.80. *Временная диаграмма, поясняющая процесс преобразования аналогового сигнала в цифровой код.*

Одновременно с американским ученым Найквистом аналогичный результат был получен русским ученым академиком В.А. Котельниковым, поэтому теорему о минимальном значении частоты дискретизации в России именуют теоремой Котельникова (прим. переводчика).

Частота дискретизации определяет максимальный интервал времени T_S между соседними отсчетами (рис. 4.80):

$$T_S = 1/f_S$$

В соответствие с критерием Найквиста минимальная частота дискретизации чисто синусоидального сигнала должна быть в два раза выше частоты этого сигнала. Если исследуемый сигнал имеет более сложную форму, то следует провести гармонический анализ этого сигнала и определить частоты наивысшей значимой гармоники.

Пример. Верхняя граница частотного диапазона голоса человека примерно равна 4 кГц. Поэтому частота дискретизации в оборудовании телефонной компании должна составлять не менее 8 кГц.

На практике исследуемый сигнал перед оцифровкой должен быть преобразован фильтром низкой частоты, который устранит шумовую составляющую сигнала, а также нежелательные высокочастотные гармоники.

4.20.2. Представление аналоговой величины в цифровом коде

Полученные в процессе дискретизации по времени аналоговые отсчеты должны быть преобразованы в цифровой код. С технической точки зрения наиболее удобно преобразовывать в цифровой код сигналы в виде напряжения. Именно поэтому датчики различных физических величин по существу являются преобразователями типа ток-напряжение, температура – напряжение, давление – напряжение и т.д.

В процессе преобразования измеряемое напряжение соотносится с эталонным, которое называют опорным напряжением U_{REF}. Опорное напряжение U_{REF} формируется как разность потенциалов двух стабилизированных источников напряжения: источника с высоким уровнем U_{HL} и источника с низким уровнем U_{RL}:

$$U_{REF} = U_{RH} - U_{RL}$$

Величина измеряемого напряжения U_{INP} должна обязательно находиться в диапазоне $U_{RH} - U_{RL}$. Диапазон возможных значений аналогового сигнала $U_{RH} - U_{RL}$ разбивается на некоторое число уровней, с которыми сравнивается измеряемое напряжение (рис. 4.81). При двоичном кодировании число уровней составляет 2^n, где n – число разрядов двоичного кода в дискретном представлении промежуточных уровней напряжения для сравнения. Число n называют разрядностью аналого-цифрового преобразователя (АЦП). Чем больше n, тем большим числом уровней аналогового напряжения для сравнения с измеряемым напряжением обладает АЦП, и тем точнее будет отображена в цифровом представлении действительная величина измеряемого напряжения.

Пример. Модуль АЦП в составе МК 68HC12 – 8 разрядный. Это означает, что любая величина входного аналогового напряжения преобразуется этим АЦП в 8-разрядный двоичный код без знака. Число различных уровней напряжения, с которыми в ходе аналого-цифрового преобразования сравнивается входное напряжение, составляет $2^8 = 256$. МК B32 в составе семейства 68HC12 имеет дополнительный режим преобразования АЦП, в котором число разрядов цифрового кода равно 10. Измерение входного сигнала этим АЦП будет выполнено с большей точностью, поскольку его аппаратные средства образуют $2^{10} = 1024$ уровней сравнения напряжения.

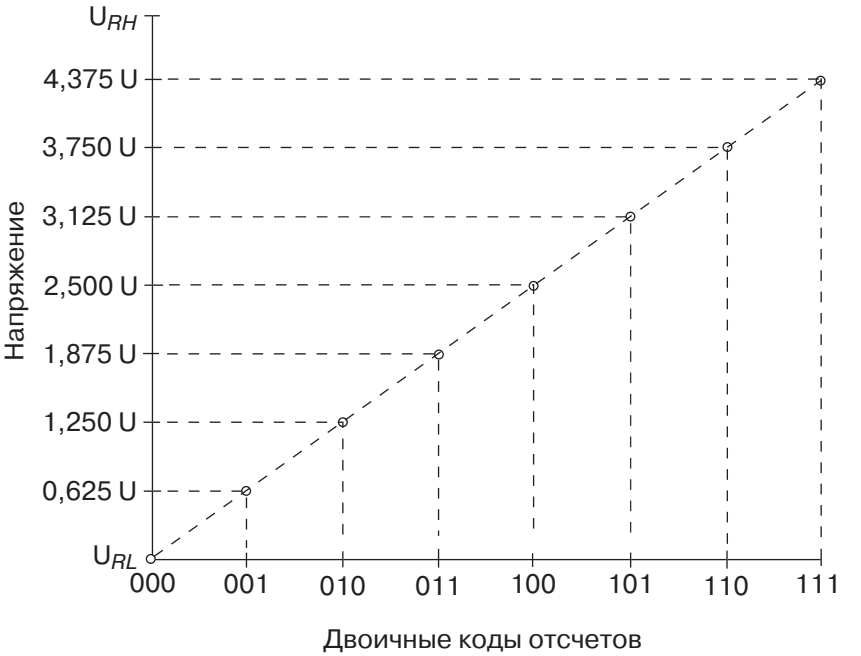


Рис. 4.81. *Прямая идеальной точности для аналого-цифрового преобразования*

4.20.3. Квантование по уровню и разрешающая способность

Преобразование величины напряжения аналогового отсчета в цифровой код называется дискретизацией или квантованием по уровню. Для получения цифрового кода, десятичный эквивалент которого прямопропорционален величине входного напряжения, АЦП сравнивает аналоговый сигнал с множеством эталонных аналоговых уровней, образованных его аппаратными средствами. Число этих уровней равно 2^n . Однако для сравнения необходимо знать величину каждого из этих уровней. Ее можно вычислить, используя понятие разрешающей способности АЦП.

Для примера предположим, что к выводу высокого уровня опорного напряжения VHL подключен источник стабилизированного напряжения 5.0 В, а к выводу низкого уровня опорного напряжения V_{RL} – источник напряжения 0 В. Если мы разделим разность этих напряжений на 256 уровней, то разность напряжений между любыми двумя соседними уровнями составит:

$$(5{,}0 - 0{,}0)/256 = 19{,}53 \text{ мВ}$$

При этом величина напряжения первого уровня сравнения составит 0 В, десятого уровня – 175,78 мВ, 256-го уровня – 4,980 В. Если мы увеличим число промежуточных уровней сравнения напряжения, шаг между уровнями уменьшится, а разрешающая способность АЦП увеличится. В общем виде разрешающая способность АЦП равна:

$$(U_{RH} - U_{RL})/ 2^n$$

Используя понятие разрешающей способности АЦП, измеряемое напряжение может быть вычислено по формуле:

$$U_{INP} = U_{RL} + x\,(U_{RH} - U_{RL})/ 2^n ,$$

где x – десятичный эквивалент двоичного кода результата преобразования.

Другой характеристикой АЦП является динамический диапазон измерения DR. Его величина измеряется в децибелах (dB). Величина динамического диапазона информирует пользователя о том, во сколько раз максимальное значение входного сигнала может превышать его минимальное значение:

$$DR\ (dB) = 20 \log 2^n = 20\ n\ (0{,}301) = 6{,}02\ n$$

Пример. Чему равен динамический диапазон 8-разрядного АЦП?

$$DR\ (dB) = 6{,}02\ n = 6{,}02 \times 8 = 48{,}16\ dB$$

4.20.4 Скорость потока данных оцифровки

Скорость потока данных оцифровки (d) отражает количество информации, которое поступает с выхода АЦП в единицу времени:

$$d = f_S \times n$$

Измеряется числом битов в секунду (бит/с).

Вопросы для самопроверки

1. Аналоговый сигнал содержит гармоники в диапазоне от 10 Гц до 4,2 кГц. Дискретные отсчеты сигнала производятся с частотой 10000 выборок в с. Достаточна ли частота выборки для полного восстановления исследуемого сигнала?
   Ответ: Минимальное значение частоты дискретных отсчетов, достаточное для полного восстановления исследуемого сигнала, равно удвоенной частоте высшей гармонической составляющей этого сигнала:

$$f_S \geqslant 2\, f_h = 2 \times 4{,}2 = 8{,}4\ кГц$$

   Поэтому 10000 выборок в с, которые эквивалентны частоте 10 кГц, достаточно для достоверного восстановления сигнала.
2. Число разрядов преобразования модуля АЦП в составе МК семейства HC12 равно восьми. Чему равна разрешающая способность этого АЦП в мВ, если U_{RH} = 5,0 В, а U_{RL} = 0 В.
   Ответ:

$$(U_{RH} - U_{RL})/\, 2^n = (5{,}0 - 0{,}0)/256 = 19{,}53\ мВ$$

3. Если модуль АЦП микроконтроллера 68HC12B32 установили в режим 10-разрядного преобразования. Чему равна его разрешающая способность при условии, что величина напряжения источников U_{RH} и U_{RL} осталась прежней?
   Ответ:

$$(U_{RH} - U_{RL})/\, 2^n = (5{,}0 - 0{,}0)/1024 = 4{,}88\ мВ$$

4. На вход АЦП поступает импульсный сигнал в форме меандра с частотой 2 кГц. Анализ амплитуд гармонических составляющих этого сигнала показал, что амплитуда 20-ой гармоники является значимой и должна быть учтена в цифровом представлении этого сигнала. Какой должна быть частота дискретизации в процессе измерения?
   Ответ:

$$f_S \geqslant 2\, f_h = 2 \times 20 = 40\ кГц$$

5. Сколько бит данных в секунду генерирует 8-разрядный АЦП при частоте выборке сигнала 40 кГц?
   Ответ:

$$f_S \times n = 40000 \times 8 = 320000\ бит/с$$

6. Предположим, что частотный диапазон музыкального сигнала составляет от 20 Гц до 20 кГц. Этот сигнал должен быть записан на компакт диск с использованием 16-разрядного АЦП, частоту дискретизации предполагается установить 44 кГц. Правильно ли выбрана частота дискретизации? Чему равна скорость потока данных при воспроизведении диска?
   Ответ: В соответствии с критерием Найквиста частота дискретизации выбрана верно. Она должна составлять не менее 2×20 кГц = 40 кГц. Выбранное значение 44 кГц превышает минимально необходимое значение частоты дискретизации. Скорость потока данных составляет:

$$f_S \times n \times (число\ каналов) = 44000 \times 16 \times 2 = 1{,}41\ Мб/с$$

4.21. Принцип действия АЦП

Известно несколько различных способов преобразования аналогового сигнала в цифровой код. Для большинства способов сигнал, подлежащий оцифровке, должен оставаться постоянным в течение всего времени аналого-цифрового преобразования. Поэтому большинство АЦП в качестве входной цепи используют так называемое устройство выборки и хранения (УВХ). УВХ – это электронная цепь с накопительным конденсатором, которой до начала преобразования заряжается до уровня измеряемого напряжения, затем напряжение на конденсаторе остается неизменным в течение времени преобразования. По способу преобразования различают следующие типы АЦП:

- последовательного приближения;
- двойного интегрирования;
- прямого счета;
- параллельные АЦП.

Мы рассмотрим только способ последовательного приближения, поскольку именно этот способ используется в модуле аналого-цифрового преобразователя МК семейства 68HC12/HCS12. Если у читателя возникнет желание познакомиться с другими способами преобразования, то он может обратиться к книге авторов Pack и Barret [2002].

4.21.1. АЦП последовательного приближения

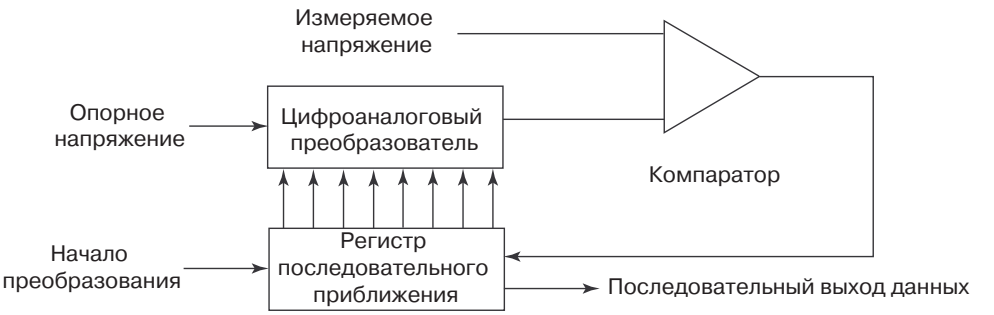
Структурная схема АЦП последовательного приближения представлена на рис. 4.82. Алгоритм функционирования АЦП последовательного приближения рассмотрим на примере. Предположим, что вывод низкого уровня опорного напряжения V_{RL} нашего АЦП подключен к потенциалу 0 В, а вывод высокого уровня опорного напряжения V_{HL} – к потенциалу 5,0 В. АЦП формирует на выходе 8-разрядный двоичный код. Число различных кодов, которыми может быть представлен результат оцифровки, составляет 28 = 256. Разрешающая способность нашего АЦП составляет:

$$(5{,}0 - 0{,}0)/256 = 19{,}53 \text{ мВ}$$

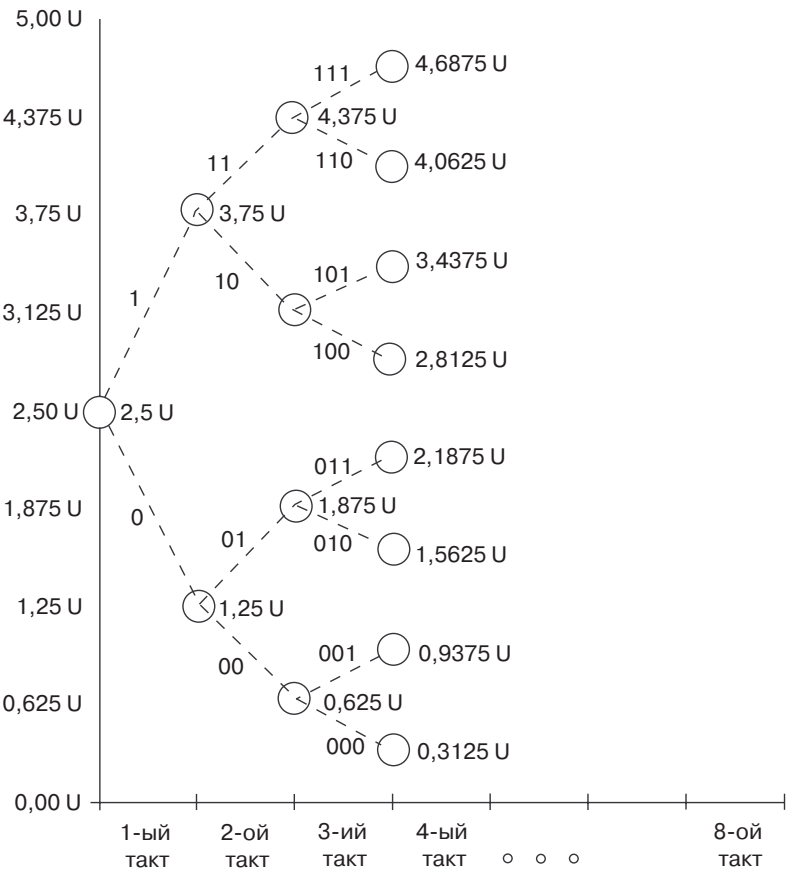
Процесс аналого-цифрового преобразования по способу последовательного приближения многотактный. Число тактов, необходимое для выполнения одного преобразования, равно числу двоичных разрядов в представлении результата. Таким образом, в нашем примере на выходе АЦП будет сформирован восьмиразрядный двоичный код результата после завершения восьмого такта преобразования. Процесс преобразования запускается по сигналу Start.

Диаграмма рис. 4.82 отражает процесс формирования цифрового кода в АЦП последовательного приближения. На каждом такте формируется один двоичный разряд результата: на первом такте – старший разряд D7, на втором такте – разряд D6 и т.д., заканчивая младшим разрядом D0 на восьмом такте.

На первом такте в регистре последовательного приближения устанавливается код К1 = 10000000b. Этот код поступает на вход цифро-аналогового преобразователя (ЦАП), опорное напряжение которого равно $U_{REF} = U_{RH} - U_{RL}$ = 5,0 В. На выходе ЦАП установится напряжение $U_{DAC} = U_{REF} \times$ К1 /28 = 2,5 В. Аналого-



а) Функциональная схема



б) Диаграмма, поясняющая принцип действия АЦП последовательного приближения

Рис. 4.82. *Аналого-цифровой преобразователь последовательного приближения*

вый компаратор сравнивает измеряемое напряжение U_{INP} с напряжением на выходе ЦАП U_{DAC}. Если $U_{INP} > U_{DAC}$, то на выходе компаратора формируется логическая 1. Если $U_{INP} < U_{DAC}$, на выходе компаратора устанавливается логический 0. Сигнал с выхода компаратора поступает на вход регистра последовательного приближения. Его значение определяет код на выходе регистра последовательного приближения в следующем такте преобразования. Если на выходе компаратора 1, то во втором такте преобразования регистр сформирует код К2 = 11000000b, если 0, то К2 = 01000000b. Иными словами, значение формируемого в текущем такте преобразования двоичного разряда равно значению логического сигнала на выходе компаратора.

На рис. 4.82 показаны возможные уровни напряжения на выходе ЦАП во втором, третьем и четвертом тактах работы АЦП. Количество уровней увеличивается с ростом номера такта, так как увеличивается число возможных комбинаций кодов на выходе регистра последовательного приближения. Так во втором такте при значении кода К2 = 11000000b на выходе ЦАП будет сформировано напряжение 3,75 В, а при коде К2 = 01000000b – 1,25 В. На восьмом такте число возможных уровней на выходе ЦАП составит 256.

АЦП последовательного приближения имеют ряд достоинств и недостатков. К достоинствам следует отнести следующие свойства:

Время преобразования не зависит от амплитуды входного сигнала и определяется только частотой тактирования и числом двоичных разрядов результата. Постоянное время преобразования удобно в микропроцессорной технике, поскольку позволяет разработчику точно рассчитать время выполнения отдельных программных фрагментов.

При получении каждого цифрового отсчета аналого-цифровое преобразование выполняется полностью сначала, результат предыдущего преобразования не оказывает влияния на последующее преобразование. Это свойство чрезвычайно важно в многоканальных модулях АЦП микроконтроллеров. Так при последовательном подключении двух измеряемых сигналов мультиплексором к одному АЦП оба измерения будут за два цикла преобразования, каждый цикл равен восьми тактам. Никаких дополнительных циклов не потребуется.

В качестве недостатков АЦП последовательного приближения следует отметить их относительно низкую скорость преобразования и достаточно сложную структуру.

Вопросы для самопроверки

1. Какому напряжению на входе АЦП соответствует код код результата «все единицы». Равно ли это измеряемое напряжение напряжению полной шкалы, т.е. $U_{REF} = U_{RH} - U_{RL}$?

   Ответ: Максимальный код на выходе АЦП соответствует значениям напряжения входного сигнала $U_{INP} > U_{REF}(1 - 2^{-n})$. Так для 12-разрядного преобразователя с напряжением опоры 10,0 В максимальный код на выходе, равный 1111 1111 1111b, будет соответствовать напряжению входного измеряемого сигнала $U_{INP} > U_{REF}(1 - 2^{-n}) = 10{,}0 \times (1 - 2^{-12}) = 9{,}99756$ В. Поскольку полученное значение крайне близко к U_{REF}, то часто говорят, что максимальный код соответствует напряжению полной шкалы U_{REF}. Однако, как мы установили, максимальный код в действительности несколько меньше напряжения полной шкалы.

4.22. Подсистема аналого-цифрового преобразования МК 68HC12

В этом разделе мы рассмотрим подсистему аналого-цифрового преобразования в составе микроконтроллеров семейства 68HC12. Эту подсистему именуют модулем ATD (Analog-To-Digital). Структура модуля представлена на рис. 4.83.

Модуль ATD - восьмиканальный, он имеет восемь входов AN0...AN7 для подключения аналоговых сигналов. В каждый момент времени аналоговый мультиплексор коммутирует один из восьми сигналов AN0...AN7 ко входу АЦП модуля ATD. Для преобразования аналогового сигнала в 8-разрядный цифровой код АЦП использует способ последовательного приближения. Ошибка преобразования составляет ±1 младшего разряда, т.е. ±1/256 полной шкалы преобразования: ±1/256 U_{REF} = ±1/256 ($U_{RH} - U_{RL}$).

Модуль ATD может работать как в режиме однократного преобразования, так и в режиме многократного преобразования. При однократном преобразовании модуль выполняется одно преобразование сигнала с заданным номером канала, после чего модуль ожидает следующего программного запуска. При многократном преобразовании модуль ATD ведет непрерывную оцифровку входного сигнала с заданным номером. После завершения одного преобразования немедленно автоматически запускается следующее. Третий режим работы модуля ATD – режим измерительной сессии. В этом режиме в регистры управления модуля ATD записывается последовательность номеров каналов, которые подлежат оцифровке. После поступления от программы сигнала запуска сначала оцифровывается канал с первым указанным номером. Затем аналоговый мультиплексор автоматически подключает ко входу АЦП канал со вторым запрограммированным номером. Запускается следующее преобразование. И так до окончания запрограммированной последовательности каналов.

Для управления процессом оцифровки восьми аналоговых сигналов предназначены регистры специальных функций модуля ATD, которые объединены в массив из 32 однобайтовых ячеек памяти. Диапазон измерения может варьироваться разработчиком, поскольку выводы для подключения опорных напряжений V_{HL} и V_{RL} выведены на ножки корпуса МК.

4.22.1 Структура и порядок функционирования

Модуль ATD использует линии порта AD в качестве входов AN0...AN7 для подключения измеряемых аналоговых сигналов. Обслуживание модуля ATD – альтернативная функция линий PAD0...PAD7 порта AD. Поэтому на рисунках функциональной схемы МК и модуля ATD (рис. 4.83) эти линии обозначаются AN0/PAD0... AN7/PAD7.

Аналоговые сигналы поступают на вход аналогового мультиплексора. Мультиплексор коммутирует один из аналоговых сигналов на вход АЦП последовательного приближения. Номер коммутируемого канала выбирается программно посредством установки определенных битов в регистрах специальных функций модуля ATD.

Запуск АЦП на преобразование осуществляется программной установкой бита в регистре управления ATDCTL5. После завершения преобразования полученный код загружается в один из регистров результата ADR0H...ADR7H. Номер регистра результата для каждого преобразования выбирается в соответствии с номером измеряемого канала. После завершения преобразования также устанавливаются флаги в регистре состояния ATDSTAT.

АЦП тактируется импульсной последовательностью с выхода делителя частоты f_{BUS}. Выводы V_{RH} и V_{RL} предназначены для подключения внешних источников прецизионного напряжения. Разность напряжений на этих входах $U_{RH} - U_{RL}$ определяет напряжение полной шкалы преобразования модуля ATD. Если напряжение входного аналогового сигнала $U_{INP} \leqslant U_{RL}$, то на выходе АЦП будет формироваться код \$00. При $U_{INP} \geqslant U_{RH}$, то результирующий код АЦП будет равняться \$FF. Формат представления кода – прямой однобайтовый беззнаковый.

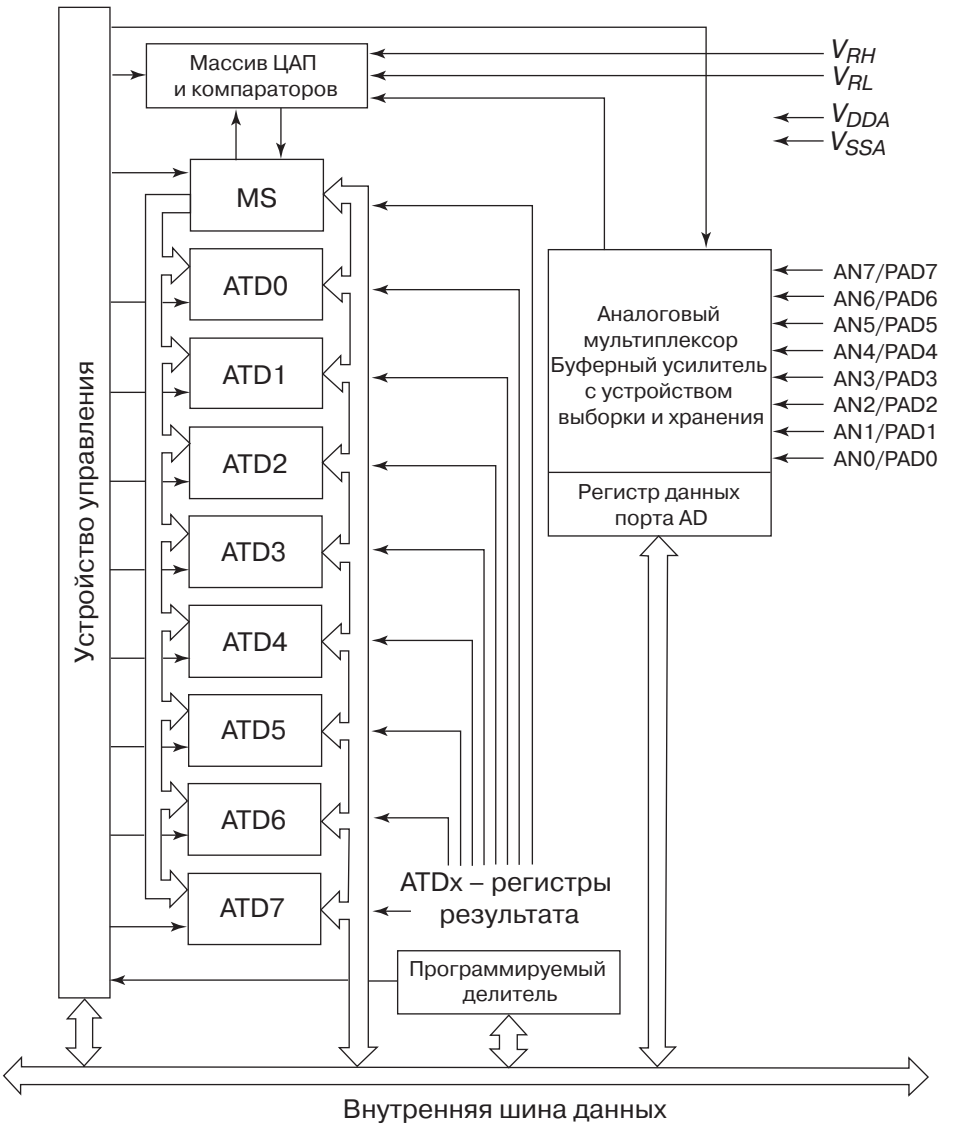


Рис. 4.83. *Структура модуля аналого-цифрового преобразователя ATD в составе МК 68HC12*

При изучении модуля ATD сначала может показаться, что процесс управления аналого-цифровым преобразованием достаточно сложен и состоит из многих операций. Мы покажем Вам, что это не так, и для этого проведем аналогию между изучаемым процессом и процессом выпечки пирожных. Для выпечки пирожных (или преобразования измеряемого напряжения в код) проделайте следующие действия:

- Включите духовку (для включения модуля ATD бит ADPU в регистре ATDCTL2 следует установить в 1);
- Прогрейте духовку в течение некоторого времени (после включения ATD следует организовать задержку на 100 мкс для завершения переходных процессов в аналоговых цепях и стабилизации системы тактирования АЦП);
- Определите, какую начинку Вы будете добавлять в пирожные: – грецкие орехи, мармелад и т.д. (определите, какой режим аналого-цифрового преобразования Вы будете использовать, и установите соответствующие биты регистров ATDCTL4 и ATDCTL5);
- Положите пирожные в духовку (установите биты регистра ATDCTL5). Они будут готовы через некоторое заранее установленное таймером время. Об окончании приготовления сообщит звуковой сигнал (В регистре ATDSTAT установится флаг SCF).
- Готовые пирожные выложите на блюдо (цифровой код буден помещен в регистры данных). Пирожные готовы для еды, а результат оцифровки для дальнейшего использования в расчетах.

4.22.2. Регистры управления модуля ATD

Ранее мы отметили, что все регистры специальных функций модуля ATD объединены в блок из шестнадцати 16-разрядных слов. Этот блок занимает в памяти 32 однобайтовых ячейки с последовательными адресами. По смысловому назначению все регистры можно разбить на четыре группы:

- ***Регистры управления.*** Эти регистры используются для выбора режима работы модуля ATD, для указания номера подлежащего оцифровке канала, для запуска АЦП на преобразование.
- ***Регистры состояния.*** Двухбайтовый регистр состояния содержит группу флагов, которые отражают текущее состояние модуля ATD: находится ли он в состоянии преобразования, какой канал преобразуется, или все преобразования уже завершены.
- ***Регистры результата.*** Модуль ATD содержит восемь 8-разрядных регистров результата (по числу аналоговых входов модуля). После завершения очередного преобразования полученный код загружается в соответствующий номеру канала регистр результата ADR0H...ADR7H.
- ***Тестовые регистры.*** Двухбайтовый регистр ATDTEST предназначен для специальных операций в процессе обязательного тестирования МК на фабрике-производителе. Доступ к этому регистру возможен только в специальном режиме работы МК. В обычном пользовательском режиме работы этот регистр для чтения и для записи недоступен.

Приступим теперь к более подробному рассмотрению формата регистров специальных функций модуля ATD.

Группа регистров управления

Эта группа регистров предназначена для управления режимами работы модуля аналого-цифрового преобразования. Группа объединяет шесть регистров управления ATDCTL0... ATDCTL5. Один из регистров (ATDCTL2) используется для включения модуля ATD в работу, поскольку аппаратные средства МК при выходе из состояния сброса отключают этот модуль для уменьшения энергии потребления. После того, как модуль ATD включили, следует выдержать паузу в 100 мкс, прежде чем подавать сигнал начала преобразования. Задержка в 100 мкс необходима, чтобы завершились переходные процессы включения в аналоговых цепях модуля. Регистры управления используются для выбора между режимами однократного или многократного преобразования, а также для назначения номера канала, подлежащего оцифровке. В режиме многократного преобразования может быть запрограммирована последовательность номеров каналов, которые будут оцифровываться друг за другом автоматически, без дополнительного вмешательства прикладной программы. Или многократные измерения могут проводиться для одного канала. Регистры управления 0, 1 и 3 по своему функциональному назначению используются достаточно редко, поэтому мы рассмотрим их кратко. Напротив, регистры 2, 4 и 5 активно используются в прикладных программах, поэтому будут рассмотрены подробно.

Регистры управления ATDCTL0 и ATDCTL1

Регистр управления ATDCTL0 располагается в памяти по адресу $0060. Этот регистр предназначен для принудительного останова аналого-цифрового преобразования, для чего в регистр под управлением программы должен быть записан определенный код. Этот режим крайне редко используется в прикладных программах. Не будем использовать его и мы в наших учебных примерах. Регистр управления ATDCTL1 располагается в памяти по адресу $0061. Он используется только в специальных случаях тестирования ИС МК.

Регистр управления ATDCTL2

Регистр управления ATDCTL2 располагается в памяти по адресу $0062. Этот регистр в первую очередь используется для включения модуля ATD. Он также содержит флаг окончания преобразования и несколько битов разрешения режимов. Формат регистра приведен на рис. 4.84. Назначение отдельных битов регистра следующее:

ADPU:

Бит разрешения работы модуля ATD. После сброса бит ADPU установлен в 0, и модуль ATD отключен от системы питания. Для включения модуля следует установить бит ADPU в 1. Модуль ATD будет готов к работе через 100 мкс.

AFFC:

Бит выбора режима сброса флагов модуля ATD. Если этот бит установлен в 0, то все установленные флаги сбрасываются обычным способом, т.е. посредством записи 1 в разряд установленного флага. Если же бит установлен в 0, то для всех флагов модуля разрешается как обычный способ сброса, так и ускоренный. Ускоренный способ сброса для каждого флага будет рассмотрен несколько позже.

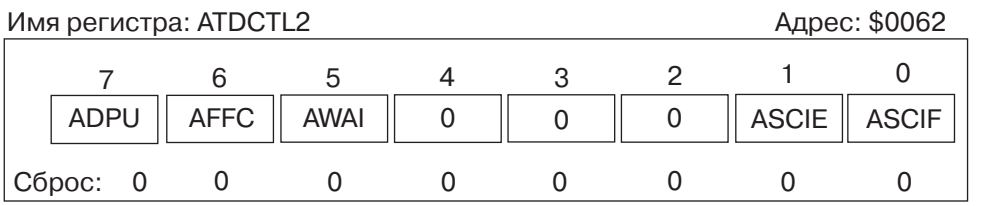


Рис. 4.84. *Формат регистра ATDCTL2*

AWAI:

Бит разрешения работы модуля ATD, когда микроконтроллер находится в состоянии пониженного энергопотребления типа WAIT. Если бит установлен в 0, то модуль ATD продолжает свою работу в состоянии WAIT. При AWAI = 1 работа модуля ATD с состоянии WAIT приостанавливается для понижения энергопотребления.

ASCIE:

Бит разрешения прерывания по флагу окончания преобразования АЦП. Если этот бит установлен в 1, и флаг окончания преобразования ASCIF установился в 1, то генерируется запрос на прерывание. Если бит ASCIE = 0, то прерывания по флагу ASCIF запрещены и его состояние следует контролировать программно.

ASCIF:

Флаг окончания преобразования АЦП. Автоматически устанавливается в 1, если ранее начатое аналого-цифровое преобразование завершено, и полученный код загружен в один из регистров результата. Если прерывания по данному флагу разрешены (ASCIE = 1), то генерируется запрос на прерывание.

Генерация запроса на прерывание с собственным вектором особенно полезна при работе модуля ATD в режиме измерительной сессии. В этом режиме АЦП выполняет несколько последовательных преобразований после единственного запуска под управлением программы. При использовании прерывания МК в течение всего времени преобразования может исполнять другие действия прикладной программы управления. Когда многократное аналого-цифровое преобразование будет завершено, установится флаг окончания преобразования ASCIF, будет выставлен запрос на прерывание, и МК, исполняя подпрограмму прерывания, считает коды оцифровки из регистров результата в память МК. Альтернативой такой организации взаимодействия прикладной программы с модулем ATD является многократный опрос флага ASCIF после программного запуска АЦП. Когда программа обнаружит, что флаг установился, регистры результата как и в предыдущем случае будут считаны. Однако в течение всего времени преобразования МК будет занят обслуживанием АЦП и не сможет исполнять других фрагментов прикладной программы.

Регистр управления ATDCTL3

Регистр управления ATDCTL3 располагается в памяти по адресу $0063. Этот регистр содержит два бита FRZ1:FRZ0, которые используются для организации работы АЦП в отладочном режиме работы.

Регистр управления ATDCTL4

Регистр управления ATDCTL4 располагается в памяти по адресу $0064. Формат регистра управления ATDCTL4 приведен на рис. 4.85. Этот регистр позволяет регулировать время выборки для устройства выборки и хранения на входе АЦП последовательного приближения. Полное время преобразования 8-разрядного АЦП в составе модуля ATD складывается из четырех интервалов:

- Интервал начального запуска, состоит из 2 периодов частоты тактирования модуля ATD;
- Неизменяемый интервал выборки аналогового сигнала, состоит из 4 периодов частоты тактирования модуля ATD;
- Регулируемый интервал выборки аналогового сигнала, его длительность программируется посредством установки битов SMP1:SMP0 в регистре управления ATDCTL4;
- Интервал преобразования напряжения на выходе устройства выборки и хранения аналого-цифровым преобразователем, состоит из 10 периодов частоты тактирования модуля ATD.

Имя регистра: ATDCTL4 Адрес: $0064

7	6	5	4	3	2	1	0
S10BM	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.85. *Формат регистра ATDCTL4*

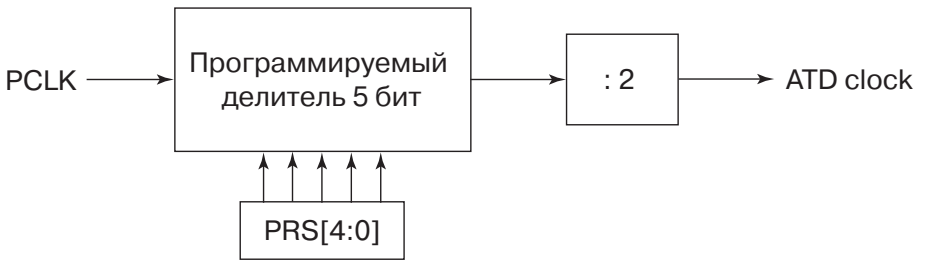


Рис. 4.86. *Структура системы тактирования модуля ATD*

Структурная схема подсистемы тактирования модуля ATD представлена на рис. 4.86. На вход подсистемы поступает импульсная последовательность PCLK с частотой f_{BUS}. Эта импульсная последовательность поступает на вход программируемого делителя частоты.

Частота тактирования модуля аналого-цифрового преобразования ATDclock формируется из PCLK путем деления двумя делителями частоты. Коэффициент деления первого делителя переменный, определяется пятью битами PRS4...PRS0 регистра ATDCTL4. Численное значение коэффициента равно десятичному эквиваленту пятиразрядного двоичного кода PRS4...PRS0 плюс 1. Например, если в разрядах PRS4...PRS0 записан код 00101 b, то коэффициент деления равен шести (5 + 1). Коэффициент деления второго делителя равен двум. Поэтому в рассматриваемом примере частота тактирования модуля ATD составит f_{BUS} /12.

Биты SMP1:SMP0 определяют регулируемый интервал выборки аналогового сигнала. Мы уже установили, что полное время одного аналого-цифрового преобразования равно 16 периодов частоты модуля ATD плюс регулируемый интервал выборки. На рис. 4.87 представлены четыре возможные комбинации кодов в разрядах SMP1:SMP0 и соответствующие им времена регулируемого интервала выборки и полного времени преобразования. Из рис. 4.87 следует, что минимальное время одного аналого-цифрового преобразования составляет 18 периодов, а максимальное - 32 периода частоты тактирования модуля ATD.

Бит S10BM определяет разрядность аналого-цифрового преобразования. При S10BM = 0, АЦП в составе модуля формирует 8-разрядный код измеряемого сигнала, при S10BM = 1 разрядность кода равна 10.

Регистр управления ATDCTL5

Регистр управления ATDCTL5 используется для выбора режима работы модуля ATD, его биты определяют номер канала измеряемого сигнала, а также осуществляют запуск АЦП на преобразование. Регистр ATDCTL5 располагается в памяти по адресу $0065, его формат представлен на рис. 4.88. Назначение отдельных битов регистра следующее:

S8CM:

Этот бит определяет число аналого-цифровых преобразований в одной последовательности. При S8CM = 0 число преобразований равно 4, при S8CM = 1 число преобразований составляет 8.

SCAN:

Бит включения сканирования. При SCAN = 0 производятся четыре или восемь последовательных измерений (одна измерительная последовательность) и заполняются четыре/восемь регистров результата. При SCAN = 1 измерения производятся непрерывно, после завершения одной последовательности автоматически запускается следующая. Регистры результата заполняются по мере поступления новых данных.

MULT:

Бит выбора многоканального или одноканального режима работы. При MULT = 0 АЦП выполняет четыре/восемь последовательных измерений по одному каналу, выбранному в зависимости от комбинации битов CD...CA. При MULT = 1 измерения производятся последовательно для группы из четырех/восьми каналов, выбираемых в зависимости от комбинации битов CD:CC. При этом каждому каналу соответствует определенный регистр результата.

CD, CC, CB. CA:

Эти биты предназначены для выбора каналов, подключаемых к АЦП. Соответствие различных комбинаций битов CD...CA и номеров каналов, подлежащих оцифровке, устанавливает таблица рис. 4.89.

SMP1:SMP0	**Время выборки** (периоды ADTclock)	**Полное время 8-разрядного преобразования** (периоды ADTclock)
00	2	18
01	4	20
10	8	24
11	16	32

Рис. 4.87. *Выбор времени аналого-цифрового преобразования в модуле ATD*

Анализируя таблицу рис. 4.89, обратите внимание, что выделенные цветом разря-

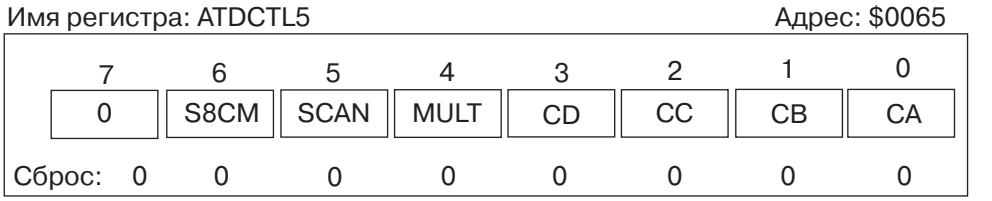


Рис. 4.88. *Формат регистра ATDCTL5*

S8CM	CD	CC	CB	CA	Номер канала	Регистр результата при MULT=1
0	0	0	0	0	AN0	ADR0
			0	1	AN1	ADR1
			1	0	AN2	ADR2
			1	1	AN3	ADR3
0	0	1	0	0	AN4	ADR0
			0	1	AN5	ADR1
			1	0	AN6	ADR2
			1	1	AN7	ADR3
0	1	0	0	0	Зарезервирован	ADR0
			0	1	Зарезервирован	ADR1
			1	0	Зарезервирован	ADR2
			1	1	Зарезервирован	ADR3
0	1	1	0	0	V_{RH}	ADR0
			0	1	V_{RL}	ADR1
			1	0	$(V_{RH} + V_{RL})/2$	ADR2
			1	1	Зарезервирован	ADR3
1	0	0	0	0	AN0	ADR0
		0	0	1	AN1	ADR1
		0	1	0	AN2	ADR2
		0	1	0	AN3	ADR3
		1	0	0	AN4	ADR4
		1	0	1	AN5	ADR5
		1	1	0	AN6	ADR6
		1	1	1	AN7	ADR7
1	1	0	0	0	Зарезервирован	ADR0
		0	0	1	Зарезервирован	ADR1
		0	1	0	Зарезервирован	ADR2
		0	1	1	Зарезервирован	ADR3
		1	0	0	V_{RH}	ADR4
		1	0	1	V_{RL}	ADR5
		1	1	0	$(V_{RH} + V_{RL})/2$	ADR6
		1	1	1	Зарезервирован	ADR7

Рис. 4.89. *Выбор номера канала для оцифровки*

ды не оказывают влияния на номер выбранного канала, если бит MULT установлен в 1. Таким образом, при MULT = 1 измерения проводятся для последовательности из четырех каналов, если S8CM = 0. Номера каналов определяются комбинацией битов CD:CC. Если S8CM = 1, то при MULT = 1 измерения проводятся для последовательности из восьми каналов. Номера каналов определяются значением бита CD.

Если бит MULT установлен в 0, то все CD...CA определяют номер канала, для которого будут произведены четыре (при S8CM = 0) или восемь (при S8CM = 1) последовательных преобразований.

Также обратите внимание, что при S8CM : CD = 01/10 b ко входу АЦП подключаются сигналы линий V_{HL} и V_{RL}, что позволяет измерить текущее значение опорного напряжения и тем самым повысить точность аналого-цифрового преобразования.

Перед рассмотрением следующих групп регистров специальных функций модуля ATD, проверьте степень усвоения материала в процессе ответа на следующие вопросы.

Вопросы для самопроверки

1. Какой код должен быть записан в регистр управления ATDCTL2 для того, чтобы включить модуль ATD и назначить обычный способ сброса флагов событий в модуле?
   Ответ: Для включения модуля флаг ADPU должен быть установлен в 1. Этот флаг располагается в разряде 7 регистра. Для назначения обычного способа сброса флагов событий модуля следует установить бит AFFC в 0. Бит AFFC располагается в разряде 6 регистра. Поэтому в регистр управления ATDCTL2, располагающийся по адресу $0062, должен быть записан код 10000000 b или $80.
2. Запишите выражение на Си для выполнения инициализации регистра ATDCTL2 кодом предыдущего вопроса.
   Ответ:

   ```
   ATDCTL2 = 0x80;
   ```

3. Какой код должен быть записан в регистр управления ATDCTL5 для того, чтобы назначить для модуля режим многократного преобразования, при этом измерительная последовательность должна производить оцифровку 8 входных сигналов.
   Ответ: Для организации заказанного режима следует установить бит S8CM в 1 (бит 6 регистра ATDCTL5), бит SCAN также в 1 (бит 5), бит MULT в 1 (бит 4), бит CD в 0 (бит 3), остальные биты CC, CB и CA значения не имеют. Обобщая, в регистр управления ATDCTL5, расположенный по адресу $0065, должен быть записан код 01110000 b или $70.
4. Запишите выражение на Си для выполнения инициализации регистра ATDCTL5 кодом предыдущего вопроса.
   Ответ:

   ```
   ATDCTL5 = 0x70;
   ```

5. Каковы преимущества использования АЦП в режиме преобразования с 10-разрядным выходным кодом относительно 8-разрядного кода оцифровки?
   Ответ: При 10-разрядном выходном коде разрешающая способность АЦП выше. Она составляет 4,88 мВ по сравнению с 19,53 мВ для 8-разрядного кода. Численные значения разрешающей способности приведены для U_{REF} = 5,0 В.

Регистр состояния ATDSTAT

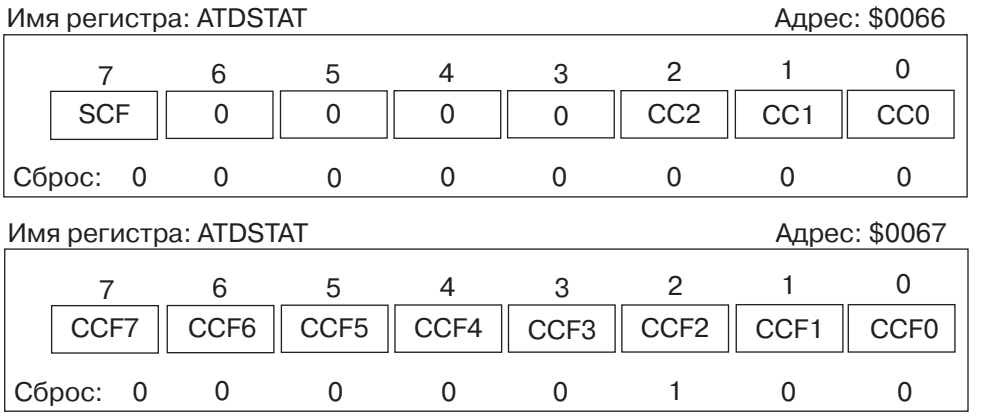


Рис. 4.90. *Формат регистра состояния ATDSTAT*

Регистр состояния ATDSTAT – это двухбайтовый регистр, который располагается в памяти по адресам $0066 и $0067. Регистр содержит в себе группу флагов, которые отражают текущее состояние аналого-цифрового преобразования в модуле ATD. Формат регистра представлен на рис. 4.90.

Флаг SCF (Sequence Complete Flag) автоматически устанавливается в 1, если все преобразования назначенной измерительной последовательности завершены. Если модуль находится в режиме однократного преобразования, то бит SCF установится после завершения исполнения измерительной последовательности из 4 или 8 преобразований. При работе модуля в режиме многократного преобразования, бит SCF установится после завершения первой измерительной последовательности.

Бит AFFC в регистре управления ATDCTL2 определяет способ сброса всех флагов в регистре состояния, в том числе и бита SCF. Если бит AFFC установлен в 0, то бит SCF обнуляется автоматически при выполнении операции записи в регистр управления ATDCTL5 для запуска новой измерительной последовательности. Если же бит AFFC установлен в 1, то бит SCF также сбрасывается автоматически при чтении одного из регистров результата.

Биты CC2...CC1 вместе составляют трехразрядный двоичный счетчик, который показывает номер текущего аналого-цифрового преобразования и, как следствие, номер регистра результата, в который будет занесен код очередного преобразования.

Каждый из восьми флагов завершения преобразования CCF7...CCF0 соответствует одноименному регистру результата ADR7H...ADR0H (бит CCF7 регистру ADR7H, бит CCF6 регистру ADR6H и т.д.). Биты CCF7...CCF0 устанавливаются автоматически, когда в процессе выполнения измерительной последовательности произошла запись в одноименный регистр результата. Если флаг AFFC установлен в 1, то биты CCF7...CCF0 сбрасываются автоматически при чтении одноименного регистра результата. При AFFC = 0 биты CCF7...CCF0 сбрасываются при чтении регистра состояния ATDSTAT, т.е. регистра с этими битами. Биты CCF7...CCF0 предназначаются для программного опроса с целью выяснения, завершилось или нет определенное аналого-цифровое преобразование в составе измерительной последовательности.

Регистр данных порта PORTAD

Обслуживание модуля ATD – альтернативная функция линий порта AD. Линии порта AD могут использоваться не только в качестве аналоговых входов модуля ATD, но и как линии порта ввода общего назначения. Каждый порт МК обладает регистром данных. Для порта AD 8-разрядный регистр данных PORTAD располагается в памяти по адресу $006F, формат этого регистра приведен на рис. 4.91. При выполнении операции чтения регистра данных порта логические сигналы на линиях AN7/PAD7... AN0/PAD0 отображаются в соответствующих разрядах регистра PORTAD.

Регистры результата ADR0H…ADR7H

После завершения каждого преобразования в составе измерительной последовательности результат преобразования помещается в один из восьми регистров результата ADR0H...ADR7H. Номер регистра определяется режимом работы модуля ATD и разрядами CD...CA регистра управления ATDCTL5 (рис. 4.89). Регистры результата ADR0H...ADR7H восьмиразрядные, в памяти располагаются по адресам $0070...$007E. Формат регистров ADR0H...ADR7H приведен на рис. 4.92.

В регистрах ADR0H...ADR7H результат аналого-цифрового преобразования представляется в прямом коде без знака. Диапазон возможных значений кодов – 0...255. Для представления результата в абсолютных единицах необходимо код результата разделить на число единиц полной шкалы, а затем умножить на напряжение полной шкалы:

$$U_{IZM} = K_{IZM}/256 \times U_{REF}$$

Вопросы для самопроверки

Имя регистра: PORTAD — Адрес: $006F

7	6	5	4	3	2	1	0
PAD7	PAD6	PAD5	PAD4	PAD3	PAD2	PAD1	PAD0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.91. *Формат регистра данных порта PORTAD*

Регистр данных канала 0: ADR0H:ADR0L	Адрес: $0070:0071
Регистр данных канала 1: ADR1H:ADR1L	Адрес: $0072:0073
Регистр данных канала 2: ADR2H:ADR2L	Адрес: $0074:0075
Регистр данных канала 3: ADR3H:ADR3L	Адрес: $0076:0077
Регистр данных канала 4: ADR4H:ADR4L	Адрес: $0078:0077
Регистр данных канала 5: ADR5H:ADR5L	Адрес: $007A:007B
Регистр данных канала 6: ADR6H:ADR6L	Адрес: $007C:007D
Регистр данных канала 7: ADR7H:ADR7L	Адрес: $007E:007F

Имя регистра: ADRxH — Адрес:

7	6	5	4	3	2	1	0
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Сброс: 0	0	0	0	0	0	0	0

Имя регистра: ADRxL — Адрес:

7	6	5	4	3	2	1	0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.92. *Формат регистров результата модуля ATD*

1. Код измеренного напряжения на выходе 8-разрядного АЦП равен 10001010 b. Потенциалы на входах опорных напряжений АЦП составляют U_{RH} = 5,0 В и U_{RL} = 0 В. Каково значение измеренного напряжения в Вольтах?
   Ответ: Десятичный эквивалент двоичного кода 10001010 b равен 138. Поэтому измеренное напряжение равно:

$$U_{IZM} = K_{IZM}/256 \times U_{REF} = (138/256) \times (5,0 - 0) = 2,69\ В$$

Тестовый регистр ATDTEST

Двухбайтовый регистр ATDTEST располагается в памяти по адресам $0068 и $0069. Доступ к этому регистру возможен только в специальном режиме работы МК. В обычном пользовательском режиме работы этот регистр для чтения и для записи недоступен.

4.22.3. Пример программирования модуля ATD

Для того чтобы воспользоваться модулем аналого-цифрового преобразования ATD для измерения уровня напряжения на нескольких аналоговых входах МК, необходимо выполнить следующие действия:

- Подключить источники стабилизированного напряжения ко входам опорного напряжения V_{RF} и V_{RL} . Необходимо помнить, что напряжение на входе высокого уровня опорного напряжения V_{RF} не должно превышать 5,0 В, а на входе низкого уровня *VRL* напряжение должно быть не менее 0 В. Кроме того, разность напряжений на входах V_{RF} и V_{RL}, равная напряжению полной шкалы АЦП $U_{REF} = U_{RH} - U_{RL}$, не должна быть менее 2,5 В;
- Подключить источники измеряемых аналоговых сигналов ко входам AN0...AN7. Напряжение измеряемых сигналов должно находиться в диапазоне 0 В = $U_{SS} \leqslant U_{IZM} \leqslant U_{DD}$ = 5,0 В;
- Осуществить внутреннюю коммутацию напряжения питания к модулю ATD. Для этого записать 1 в бит ADPU регистра управления ATDCTL2. Адрес регистра – $0062;
- Выдержать паузу в 100 мкс для завершения переходных процессов в модуле ATD. В рассматриваемом ниже программном фрагменте мы покажем, как организовать такую задержку;
- Назначить режим работы модуля ATD посредством записи необходимых слов инициализации в управляющие регистры модуля;
- Запустить измерительную последовательность посредством записи в регистр управления ATDCTL5;
- Контролировать ход преобразования, используя флаги регистра состояния модуля ATDSTAT;
- Когда измерительная последовательность будет завершена, считать данные из регистров результата ADR0H...ADR7H в память МК;
- Если экономия энергии важна для Вашего применения, следует выключить модуль ATD, установив бит ADPU регистра ATDCTL2 в 0.

Мы рассмотрим программную реализацию действий по управлению модулем аналого-цифрового преобразования на примере простейшего цифрового вольтметра.

Цифровой вольтметр

Программный фрагмент voltmeter.c производит измерение аналогового сигнала на входе AN6. Измерительная последовательность состоит из четырех преобразований. Режим измерения однократный. Результаты четырех последовательных преобразований одного и того же сигнала располагаются в четырех регистрах результата ADR0H...ADR3H. Эти измерения усредняются, что позволяет снизить влияние шумов. Полученный 8-разрядный двоичный код преобразуется к истинному значению измеряемого напряжения, умноженному на 100 d. Умножение на нормирующий коэффициент (100 в десятичной системе счисления) необходимо, чтобы использовать в программе целочисленные форматы представления данных. Полученный результат содержит одну десятичную цифру целой части измеренного напряжения, это единицы Вольт. А также две цифры десятичной дробной части. Это десятые и сотые доли Вольт в представлении результата. Промежуточные результаты исполнения программы, а также измеренное напряжение выводятся на экран персонального компьютера.

```
/*-------------------------------------------------------------------*/
/* filename: voltmeter.c                                             */
/* В этом примере реализован простейший цифровой вольтметр.          */
/* Программа выполняет одно измерение и выводит данные на экран     */
/* персонального компьютера. Для того, чтобы выполнить следующее измерение*/
/* следует перезапустить программу                                   */
/*-------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>
#include<stdio.h>

#define DECIMAL 0x2E       /*определить код точки на экране*/
#define V 0x56             /*определить код символа "V" для экрана*/

/*используемые функции*/
void delay_100us(void);
void ADC_convert(void);
void delay_5ms (void);

void main(void)
{
printf("HELLO\n");         /*вывести приветствие на экран*/
ATDCTL2 = 0x80;            /*включить питание модуля, запретить */
                           /*прерывания от модуля*/
printf ("ADC \n" );
delay_100us();             /*задержка 100мкс*/
printf ("warmed up\n");
ATDCTL3 = 0x00;     /*обеспечить доступ к модулю в отладочном режиме*/
ATDCTL4 = 0x01;            /*установить 2 такта для времени выборки*/
                           /*и коэффициент деления, равный 4*/
printf ( "ready\n" );
ADC_convert();             /*реализовать цифровой вольтметр*/
}
/*-------------------------------------------------------------------*/
```

```
/* Функция ADC_convert 4 измерения, усредняет их, вычисляет абсолютное  */
/* значение напряжения, а затем преобразует результат в коды ASCII для*/
/* вывода на экран                                                      */
/*----------------------------------------------------------------------*/
void ADC_convert (void)
{
unsigned int sumadr;
unsigned int avg_bin_voltage;
unsigned int int_voltage;
unsigned int ones_int;
unsigned int tenths_int;
unsigned int hundreths_int;
char ones;
char tenths;
char hundreths;

ATDCTL5 = 0x06;     /*4 преобразования в последовательности, канал 6*/

/* Ожидать завершения измерительной последовательности*/
while((ATDSTAT & 0x8000) != 0x8000)
      {
      ;
      }
/* Вывести коды преобразования на экран для контроля */
printf("%x  %x  %x  %x\n" , ADR0H, ADR1H, ADR2H, ADR3H);

/* Взять среднее от 4 измерений для устранения шума*/
sumadr = ADROH + ADRIH + ADR2H + ADR3H;
avg_bin_voltage = sumadr/4;

/*Преобразовать результат из двоичного кода к абсолютному значению,*/
/* умноженному на 100 D, получится число в диапазоне от 0 до 500 D*/
int_voltage = (100*avg_bin_voltage/255) *5;

/*Выделить старший разряд результата и преобразовать в код ASCII */
/* Правило преобразования: десятичная цифра +48*/
ones_int = int_voltage/l00;
ones = (char) (ones_int + 48);

/*Выделить второй разряд результата (десятые доли) и преобразовать в
код*/
/* ASCII */
tenths_int = (int_voltage - ones_int*100)/10;
tenths = (char) (tenths_int + 48);

/*Выделить третий разряд результата (сотые доли) и преобразовать в
код*/
/* ASCII */
hundreths_int = (int_voltage - ones_int*100 - tenths_int*10) /1;
hundreths = (char) (hundreths_int + 48);

/*Вывести значение измеренного напряжения на экран */
printf("%c.%c%cV\n", ones, tenths, hundreths);
}
/*----------------------------------------------------------------------*/
```

```
/* Функция delay_100us формирует задержку в 100 мкс, частота тактирования*/
/* межмодульных магистралей МК составляет 8 МГц                          */
/*-----------------------------------------------------------------------*/
void delay_100us (void)
{
int i;
for (i=0; i<50; i++)
      {
      asm( "nop") ;
      }
}
/*-----------------------------------------------------------------------*/
/* Функция delay_5ms формирует задержку в 5мс, частота тактирования      */
/* межмодульных магистралей МК составляет 8 МГц                          */
/*-----------------------------------------------------------------------*/
void delay_5ms (void)
{
int i;
for (i=0; i<50; i++)
      {
      delay_100us();
      }
}
/*-----------------------------------------------------------------------*/
```

4.22.4. Обслуживание прерываний от модуля ATD

В рассмотренном примере использовался метод программного опроса триггера завершения измерительной последовательности SCF в регистре состояния ATDSTAT. В этом же регистре располагаются флаги CCF7...CCF0, которые устанавливаются автоматически, когда в процессе выполнения измерительной последовательности произошла запись в одноименный регистр результата. Флаги CCF7...CCF0 могут быть программно опрошены с целью выяснения, завершилось или нет определенное аналого-цифровое преобразование в составе измерительной последовательности. В относительно медленных приложениях такая необходимость отсутствует, поскольку малое быстродействие объекта управления позволяет прикладной программе дождаться окончания всей измерительной последовательности, и лишь затем начать выборку из регистров модуля ATD кодов результатов для проведения дальнейших расчетов.

Если же прикладная программа обслуживает достаточно динамичный объект, то может возникнуть необходимость выборки из регистров результата данных в процессе исполнения измерительной последовательности. Именно тогда и следует воспользоваться флагами завершения преобразования в отдельных каналах CCF7...CCF0. Прикладная программа должна запустить измерительную последовательность на исполнение, а затем контролировать состояние флага первого преобразуемого канала. Когда флаг установится, соответствующий регистр результата может быть считан. При этом во время выборки из памяти первого результата, будет продолжаться аналого-цифровое преобразование следующего сигнала. И так до завершения всей измерительной последовательности.

Программный опрос или поллинг (в англоязычной терминологии) – не единственный способ взаимодействия прикладной программы с модулем аналого-цифрового преобразования. Модуль ATD способен генерировать запросы на прерывание по завершению исполнения измерительной последовательности. Прерывания разрешаются посредством установки бита ASCIE регистра ATDCTL2 в 1. При этом запрос на прерывание генерируется, если устанавливается бит ASCIF в том же регистре. Этот же бит следует сбросить в подпрограмме прерывания, чтобы по завершению следующей измерительной последовательности был сгенерирован новый запрос. Обратите внимание на некоторую особенность битов состояния модуля ATD. Существует два бита завершения исполнения измерительной последовательности: бит ASCIF в регистре управления ATDCTL2 и бит SCF в регистре состояния ATDSTAT. При этом первый бит используется в прерываниях, а второй при программном опросе.

4.23. Особенности модуля ATD в составе МК семейства HCS12

Модуль ATD в составе МК семейства HCS12 по своей архитектуре и исполняемым функциям крайне близок к рассмотренному модулю аналого-цифрового преобразования микроконтроллеров семейства 68HC12. Однако он имеет ряд отличий, которые позволили улучшить его функциональные характеристики:

- Программируемая разрядность аналого-цифрового преобразования: 8 или 10 бит;
- Несколько способов представления результата преобразования: в формате со знаком и без знака, со сдвигом кода 10-разрядного кода в старшие или младшие разряды 16-разрядного регистра результата;
- Возможность запуска измерительной последовательности от внешнего сигнала;
- Программируемое от 1 до 8 число преобразований в одной измерительной последовательности;
- В некоторых моделях МК число аналоговых входов увеличено до 16.

Далее мы кратко остановимся на каждом из этих дополнительных свойств модуля ATD в составе МК HCS12.

4.23.1. Выбор разрядности АЦП

Бит SRES8 в регистре управления ATDCTL4 (рис. 4.93) определяет число разрядов кода аналого-цифрового преобразования, которое выполняет встроенный АЦП модуля ATD. Использование 10-разрядного кода в представлении результата позволяет повысить разрешающую способность АЦП с 19,53 мВ до 4,88 мВ.

4.23.2. Представление результата измерения

Результат аналого-цифрового преобразования модуля ATD может быть представлен в формате без знака или со знаком. Бит DSGN регистра ATDCTL5 (рис. 4.93) осуществляет выбор между представлением без знака (при DSGN = 0) или со знаком

(при DSGN = 1). Код результата может размещаться со сдвигом влево, т.е. в старшие разряды 16-разрядного регистра результата (бит DJM = 0), или со сдвигом вправо, т.е. в младшие разряды регистра результата (бит DJM = 1).

Имя регистра: ATDCTL2 Адрес: $0002

	7	6	5	4	3	2	1	0
	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIGE	ASCIE	ASCIF
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: ATDCTL3 Адрес: $0003

	7	6	5	4	3	2	1	0
	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0
Сброс:	0	0	1	0	0	0	0	0

Имя регистра: ATDCTL4 Адрес: $0004

	7	6	5	4	3	2	1	0
	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
Сброс:	0	0	0	0	0	1	0	1

Имя регистра: ATDCTL5 Адрес: $0005

	7	6	5	4	3	2	1	0
	DJM	DSGN	SCAN	MULT	0	CC	CB	CA
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: ATDSTAT0 Адрес: $0006

	7	6	5	4	3	2	1	0
	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: ATDSTAT1 Адрес: $000B

	7	6	5	4	3	2	1	0
	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
Сброс:	0	0	0	0	0	0	0	0

Имя регистра: ATDDIEN Адрес: $000D

	7	6	5	4	3	2	1	0
	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	IEN0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.93. *Формат регистров модуля ATD в составе HCS12*

4.23.3. Запуск измерительной последовательности от внешнего сигнала

Аппаратные средства модуля аналого-цифрового преобразования в составе МК HCS12 позволяют производить запуск измерительной последовательности по изменению внешнего сигнала, который подключается на вход AN7/ETRIG/PAD7 модуля. Такой режим чрезвычайно полезен для синхронизации проводимых измерений с режимами работы объекта управления. Режим внешнего запуска разрешается установкой бита ETRIGE регистра ATDCTL2 в 1. При ETRIGE = 0 реализуется обычный программный режим запуска.

Параметры внешнего сигнала запуска настраиваются посредством задания кодовой комбинации двух битов управления в регистре ATDCTL2. Бит ETRIGLE назначает запуск по уровню или по перепаду внешнего сигнала, бит ETRIGP определяет активный уровень сигнала запуска (логический 0 или 1). Формат обновленного регистра управления ATDCTL2 с перечисленными битами представлен на рис. 4.93. При ETRIGLE:ETRIGP = 00 реализуется запуск измерительной последовательности по спадающему фронту внешнего сигнала, при ETRIGLE:ETRIGP = 01 – по нарастающему фронту, при ETRIGLE:ETRIGP = 10 запуск осуществляется при низком логическом уровне внешнего сигнала, при ETRIGLE:ETRIGP = 11 – при высоком уровне сигнала.

4.23.4. Программируемое число преобразований в измерительной последовательности

Рассмотренный ранее модуль ATD в составе МК 68HC12 мог исполнять измерительные последовательности, состоящие из 4 или из 8 преобразований. В МК HCS12 число измерений в последовательности может быть любым в диапазоне от 1 до 8. Если бит MULT регистра управления ATDCTL5 установлен в 0, то все измерения в последовательности выполняются для одного канала. При этом номер канала определяют биты CC:CB:CA регистра ATDCTL5, а число измерений в последовательности – биты S8C:S4C:S2C:S1C. Например, если MULT = 0, CC:CB:CA = 010 b, а S8C:S4C:S2C:S1C = 0101 b, то измерительная последовательность будет состоять из 5 преобразований сигнала канала 2. Обратите внимание, что формат регистра ATDCTL5 в модуле аналого-цифрового преобразования HCS12 изменен (рис. 4.93).

Если бит MULT установлен в 1, то в измерительной последовательности будет производиться оцифровка сигналов разных каналов. При этом биты CC:CB:CA будут определять номер первого канала в последовательности, а биты S8C:S4C:S2C:S1C – число измерений в последовательности. Например, MULT = 1, CC:CB:CA = 010 b, а S8C:S4C:S2C:S1C = 011 b. Измерительная последовательность будет состоять из трех преобразований, последовательно будут измерены напряжения в каналах 2, 3, 4.

Дополнительные возможности по выборке данных из регистров результата модуля ADT HCS12 предоставляет режим FIFO (First In First Out – первым пришел, первым вышел). Если бит FIFO в регистре ATDCTL3 равен 0, то реализуется ранее рассмотренный порядок записи кодов преобразования в регистры результатов. При FIFO = 1 регистры результата образуют накопительный банк, который хранит 8 последних измерений. При каждом следующем преобразовании, результат преобразования записывается в банк, при этом первое из 8 хранящихся в банке измерений теряется.

4.23.5. Увеличение числа аналоговых входов

Увеличение числа аналоговых входов для подключения измеряемых сигналов в МК семейства HCS12 ведется не путем изменения структуры модуля ATD, который имеет 8 аналоговых входов, а путем увеличения числа модулей аналого-цифрового преобразования в составе отдельных моделей МК. Так МК MC9S12DP256B имеет в своем составе два модуля аналого-цифрового преобразования (ATD0 и ATD1).

4.23.6. Регистры модуля ATD HCS12

На рис. 4.93 приведены форматы новых, дополнительных по сравнению с ранее рассмотренной версией модуля ATD регистров специальных функций ATDSTAT0, ATDSTAT1, ATDDIEN, а также тех регистров, которые, сохранив прежнее имя, претерпели изменение формата.

Регистр состояния ATDSTAT0

Регистр состояния ATDSTAT0 – это однобайтовый регистр. Флаг завершения измерительной последовательности SCF этого регистра устанавливается в 1, если все преобразования назначенной измерительной последовательности завершены. Флаг SCF может быть сброшен тремя способами:

- Программной установкой бита SCF в 1 (обычный способ);
- Программным запуском новой измерительной последовательности посредством записи в регистр ATDCTL5;
- Чтением регистра результата при установленном флаге разрешения режима быстрого сброса AFFC в регистре ATDCTL2.

Регистр состояния ATDSTAT1

Регистр состояния ATDSTAT1 – также однобайтовый регистр состояния. Содержит биты CCF7...CCF0, которые устанавливаются, когда в процессе выполнения измерительной последовательности произошла запись в одноименный регистр результата.

Регистр разрешения цифрового входа порта ATDDIEN

Регистр ATDDIEN содержит восемь битов IENn (n – номер канала). Если бит IENn равен 1, то соответствующая линия порта AD работает в режиме обычной линии ввода (цифровой ввод), при IENn = 0 линия порта AD с номером n работает в режиме аналогового ввода.

4.24. Подсистема широтно-импульсной модуляции

Микроконтроллер 68HC912B32 и все МК семейства HCS12 имеют в своем составе модуль широтно-импульсного модулятора PWM (Pulse Width Modulation). Широтно-импульсная модуляция (ШИМ) – это способ регулирования скорости вращения двигателя постоянного тока посредством изменения среднего значения напряжения, приложенного к обмоткам двигателя. ШИМ-сигнал может также использоваться для изменения направления движения радиоуправляемой модели автомобиля.

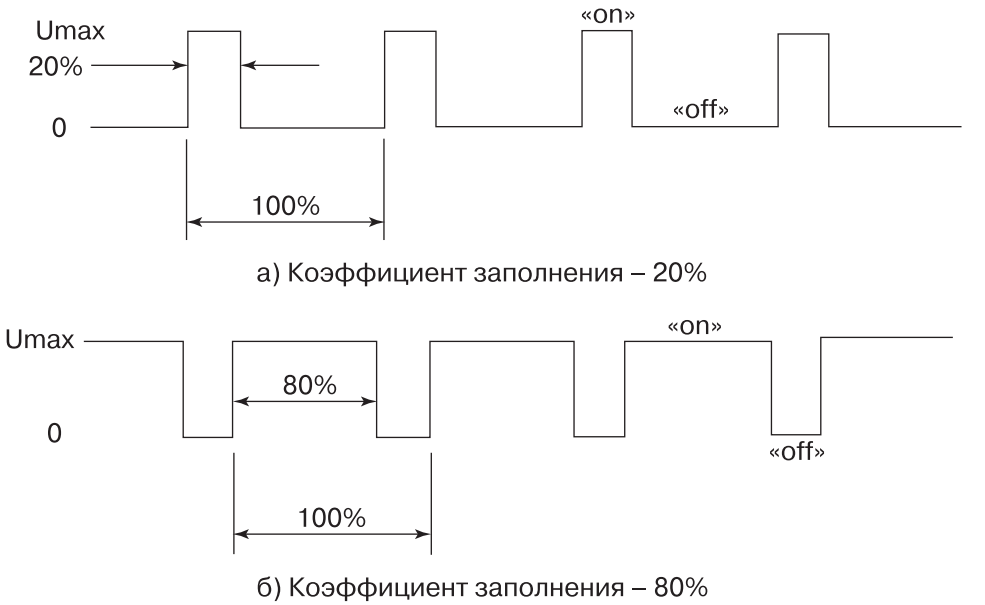


Рис. 4.94. *Временные диаграммы ШИМ-сигналов с различными коэффициентами заполнения*

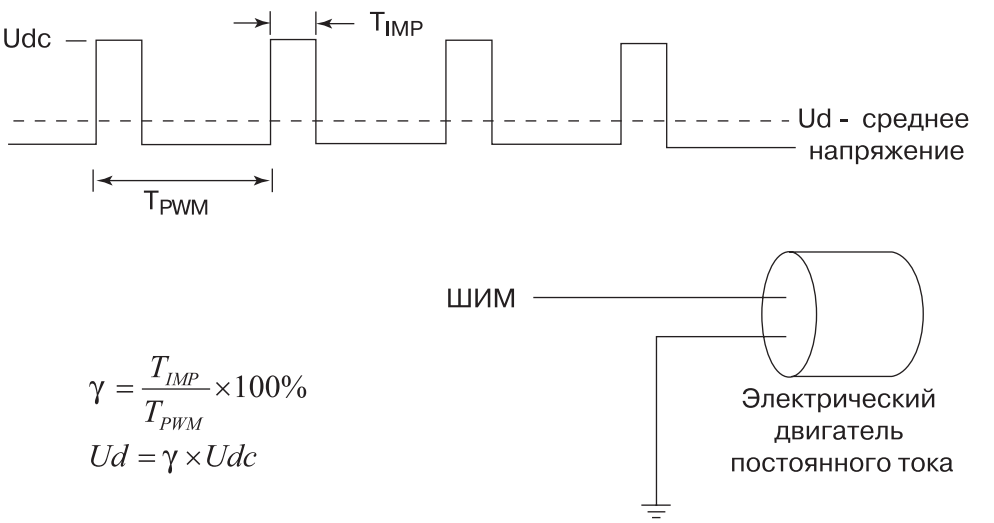


Рис. 4.95. *Управление электрическим двигателем методом ШИМ*

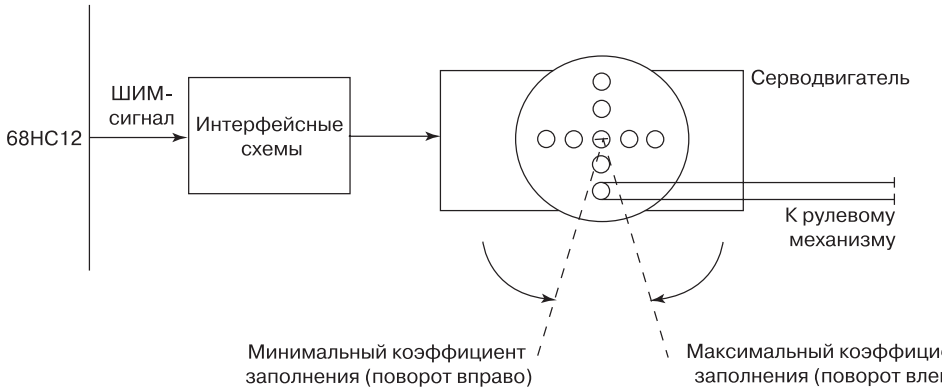


Рис. 4.96. *Использование ШИМ для поворота рулевого колеса радиоуправляемой модели автомобиля.*

Среднее значение напряжения на обмотках двигателя определяется отношением длительности импульса напряжения (состояние «on» на рис. 4.94) к длительности периода следования импульсов. Это отношение называют коэффициентом заполнения γ, который является величиной безразмерной. Например, если коэффициент заполнения равен 80%, то длительность импульса напряжения (состояние «on») составляет 80%, а длительность паузы (состояние «off») – 20% от длительности периода следования импульсов. На рис. 4.94 показана форма напряжения на обмотках двигателя постоянного тока при двух значениях коэффициента заполнения $\gamma = 20\%$ и $\gamma = 80\%$.

Рисунок 4.95 иллюстрирует способ управления двигателем постоянного тока с использованием ШИМ. Если коэффициент заполнения $\gamma = 80\%$, то к обмоткам двигателя будет приложено среднее напряжение, равное 80% от напряжения сети постоянного тока U_{dc}. Если коэффициент заполнения равен 100%, то это означает, что полупроводниковые ключи, используемые для коммутации напряжения к обмоткам двигателя, включены постоянно, и напряжение на обмотках двигателя не имеет импульсной формы. Отметим, что предельная нагрузочная способность выходов МК 68HC12/HCS12 недостаточна для коммутации токов и напряжений обмоток двигателя постоянного тока. Поэтому на выходах МК может быть воспроизведен только маломощный ШИМ-сигнал для управления двигателем, который затем должен быть усилен посредством специального полупроводникового коммутатора. Пример такого коммутатора будет рассмотрен в главе 5.

Широтно-импульсная модуляция используется не только для управления двигателем. Она имеет множество применений в технике управления различного рода исполнительными механизмами, в том числе в параграфе 4.24.5 будет рассмотрен пример формирования сигнала управления для серводвигателя, который приводит в движение рулевой механизм радиоуправляемой модели автомобиля.

4.24.1. Структура модуля PWM

Для генерации модулированных по длительности импульсных сигналов средствами модуля PWM необходимо рассчитать временные параметры этих сигналов в относительных единицах. Единицей измерения времени для модуля PWM служит период следования импульсов сигнала E_CLOCK, который является первичным сигна-

лом тактирования программируемых делителей частоты и счетчиков в составе модуля PWM. Частота импульсной последовательности E_CLOCK равна частоте системной шины МК. Поскольку частота системной шины МК для каждого нового проекта выбирается разработчиком по совокупности технических требований к изделию, то для каждого проектируемого изделия длительность кванта времени модуля PWM может оказаться различной.

После того, как временные параметры генерируемых сигналов, выраженные в числе периодов E_CLOCK, подсчитаны, следует загрузить их в регистры периода PWPERx и регистры коэффициента заполнения PWDTYx (x – номер канала модуля PWM, x = 0...3).

Модуль PWM имеет в своем составе четыре канала, которые могут генерировать на своих выходах ШИМ-сигналы с независимыми временными параметрами. Каждый канал состоит из двоичного 8-разрядного счетчика PWCNTx с системой предварительных делителей, двух схем сравнения и двух программно доступных регистров PWPERx и PWDTYx. Если работа канала разрешена, то счетчик канала PWCNTx считает непрерывно. Код в счетчике PWCNTx нарастает с $00 до значения, которое записано в регистр периода PWPERx. В момент сравнения счетчик автоматически сбрасывается, и счет продолжается, начиная с кода $00. Код счетчика PWCNTx непрерывно сравнивается с кодом регистра коэффициента заполнения PWDTYx. Если код в регистре PWDTYx превышает текущий код счетчика PWCNTx, то на выходе канала формируется логический сигнал высокого уровня. В момент, когда код счетчика превысит код в регистре коэффициента заполнения, на выходе установится низкий логический уровень (рис. 4.97).

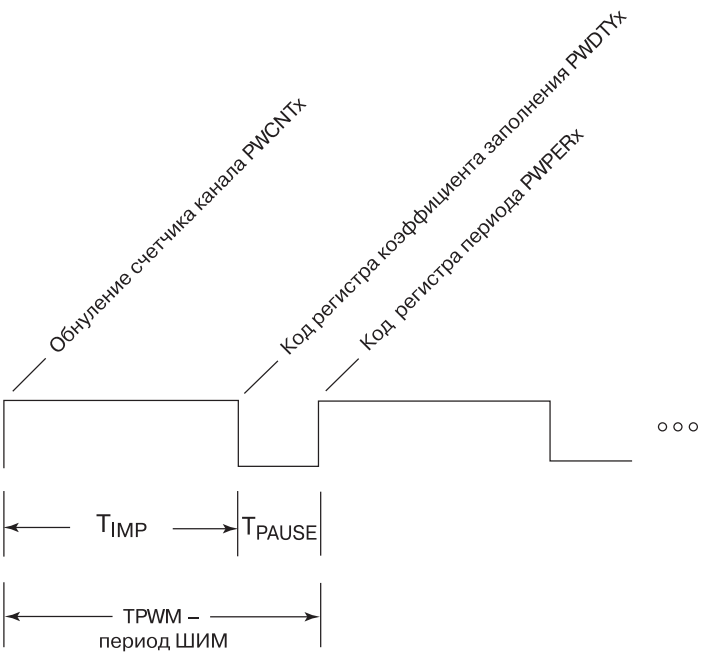


Рис. 4.97. *Временная диаграмма, поясняющая формирование ШИМ-сигнала в модуле PWM.*

Импульсная последовательность E_CLOCK, частота которой равна частоте системной шины, является базовой для тактирования всех каналов модуля PWM. Система тактирования модуля состоит из нескольких программируемых делителей, которые позволяют расширить диапазон частот генерируемых ШИМ-сигналов. Включение дополнительных делителей частоты между входом E_CLOCK модуля и счетчиком текущего кода канала PWCNTx приведет к изменению единицы измерения временных интервалов для данного канала. Поэтому коды периода следования и коэффициента заполнения должны быть пересчитаны в соответствии с суммарным коэффициентом деления программируемых делителей.

Как было отмечено ранее, модуль PWM в составе МК B32 имеет четыре 8-разрядных канала. Допускается объединение каналов попарно с целью получения 16-разрядного канала, способного генерировать ШИМ-сигнал с 16-разрядным разрешением по коэффициенту заполнения и периоду следования импульсов. Используя программно устанавливаемые опции конфигурации, на основе модуля PWM с четырьмя каналами могут быть получены следующие структуры:

- Четыре независимых 8-разрядных генератора ШИМ-сигнала (HCS12: восемь 8-разрядных);
- Два независимых 16-разрядных генератора ШИМ-сигнала (HCS12: четыре 16-разрядных);
- Два 8-разрядных и один 16-разрядный генератор ШИМ-сигнала.

В 8-разрядных каналах значения коэффициента заполнения и кода периода могут изменяться от 0 до 255 ($2^8 - 1$). Для 16-разрядных каналов эти же параметры могут принимать значения от 0 до 64535 ($2^{16} - 1$). Если задать код коэффициента заполнения равным коду периода, то выходной сигнал будет постоянным (без переключений между 1 и 0).

Объединенный 16-разрядный канал обладает всеми теми же свойствами, что и 8-разрядный. Он допускает регулирование периода и коэффициента заполнения, но отличается большей разрешающей способностью временной сетки в процессе регулирования. Например, минимальное значение коэффициента заполнения для 8-разряднго канала равно 1/256, в то время, как для 16-разрядного – 1/64356. Однако в 8-разрядном режиме достижимы большие частоты ШИМ-сигнала при той же базовой частоте тактирования канала.

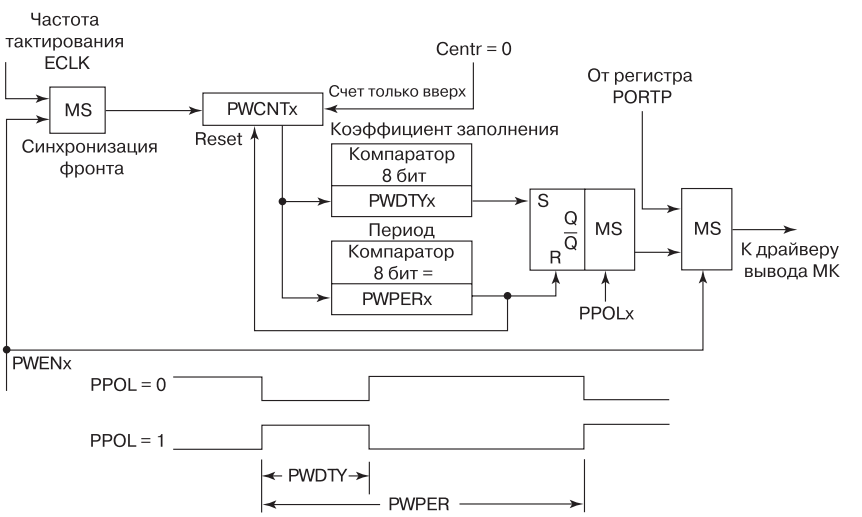


Рис. 4.98. Модуль PWM в режиме фронтовой ШИМ

4.24.2. Режимы центрированной и фронтовой ШИМ

Каждый из каналов может работать в режиме фронтовой ШИМ или в режиме центрированной ШИМ. В режиме фронтовой ШИМ счетчик текущего кода канала работает только в режиме инкрементирования. Его код изменяется от $00 до кода в регистре периода PWPERx. Далее счетчик сбрасывается, и процесс счета повторяется с кода $00.

В режиме центрированной ШИМ счетчик сначала считает от $00 до кода регистра периода, далее направление счета изменяется, и в вычитающем режиме код счетчика доходит до $00. Далее процесс повторяется.

Логика работы компараторов, которые участвуют в формировании импульса, длительность которого пропорциональна коду регистра коэффициента заполнения, полностью аналогична рассмотренной ранее. В результате, при фронтовой ШИМ импульс будет сформирован в начале периода сигнала, начиная с кода счетчика $00 (рис. 4.98). При центрированной ШИМ импульс с регулируемой длительностью будет формироваться в начале и в конце периода сигнала, в то время как в середине периода на выходе будет формироваться низкий логический уровень (рис. 4.99).

Для каждого из режимов работы (фронтовая или центрированная ШИМ) имеется возможность выбора активного уровня модулированного по длительности импульса ШИМ-сигнала. Длительность импульса с высоким логическим уровнем пропорциональна коду в регистре коэффициента заполнения PWDTYx при PPOL = 1. При PPOL = 0 аналогичным образом регулируется длительность временного интервала с низким логическим уровнем, как показано на рис. 4.98 и 4.99.

При одинаковой частоте тактирования и одинаковых кодах в регистрах периода и коэффициента заполнения длительность импульса и период формируемого сигнала для фронтовой и центрированной ШИМ будут различаться.

Для фронтовой ШИМ (CENTR = 0) расчет временных параметров выходного сигнала следует производить по следующим формулам:

- Период ШИМ-сигнала равен

$$T_{PWM} = (PWPERx + 1)/fx$$

- Длительность импульса с высоким активным уровнем при PPOLx = 1 равна

$$T_{IMP} = (PWDTYx + 1) / [(PWPERx + 1) \times fx]$$

Коэффициент заполнения в этом же режиме равен

$$\gamma = (PWPERx - PWDTYx)/[(PWPERx + 1) \times fx] \times 100\%$$

- Длительность импульса с высоким активным уровнем при PPOLx = 0 равна

$$T_{IMP} = (PWPERx - PWDTYx) / [(PWPERx + 1) \times fx]$$

Коэффициент заполнения в этом же режиме равен

$$\gamma = (PWDTYx + 1)/[(PWPERx + 1) \times fx] \times 100\%,$$

где PWPERx – десятичный эквивалент кода в регистре периода PWPERx, PWDTYx – десятичный эквивалент кода в регистре коэффициента заполнения PWDTYx, PPOLx – бит выбора полярности сигнала канала в регистре конфигурации PWPOL, CENTR – бит выбора формы сигнала в регистре управления PWCTL, *fx* – частота тактирования счетчика канала с номером «x».

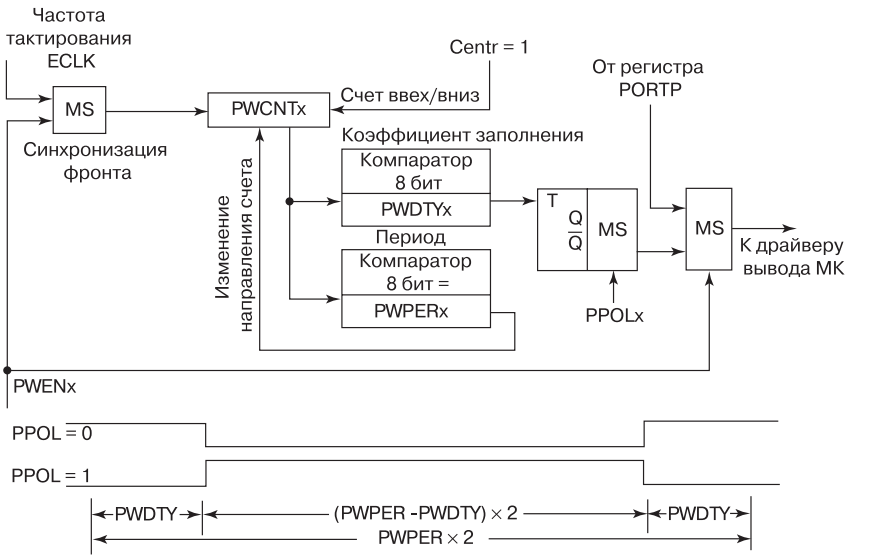


Рис. 4.99. *Модуль PWM в режиме центрированной ШИМ*

Для центрированной ШИМ (CENTR = 1) расчет временных параметров выходного сигнала следует производить по формулам:

- Период ШИМ-сигнала равен

$$TPWM = 2 \times (PWPERx + 1)/fx$$

- Длительность импульса с высоким активным уровнем при PPOLx = 1 равна

$$TIMP = 2 \times (PWDTYx + 1) / [(PWPERx + 1) \times fx]$$

Коэффициент заполнения в этом же режиме равен

$$\gamma = (PWPERx - PWDTYx)/[(PWPERx + 1) \times fx] \times 100\%$$

- Длительность импульса с высоким активным уровнем при PPOLx = 0 равна

$$TIMP = 2 \times (PWPERx - PWDTYx) / [(PWPERx + 1) \times fx]$$

Коэффициент заполнения в этом же режиме равен

$$\gamma = [(PWDTYx + 1)/(PWPERx + 1)] \times 100\%,$$

Обратите внимание, что при одинаковой частоте тактирования и одинаковых кодах в регистрах периода и коэффициента заполнения длительность импульса и период формируемого сигнала центрированной ШИМ будет в два раза больше, чем при фронтовой ШИМ. А коэффициент заполнения при этом остается одинаковым, поскольку является величиной относительной.

4.24.3. Система тактирования

Структурная схема системы тактирования модуля PWM представлена на рис. 4.100. Первичным генератором для тактирования счетчиков каналов модуля PWM служит импульсная последовательность E_CLOCK. На выходе программируемого делителя

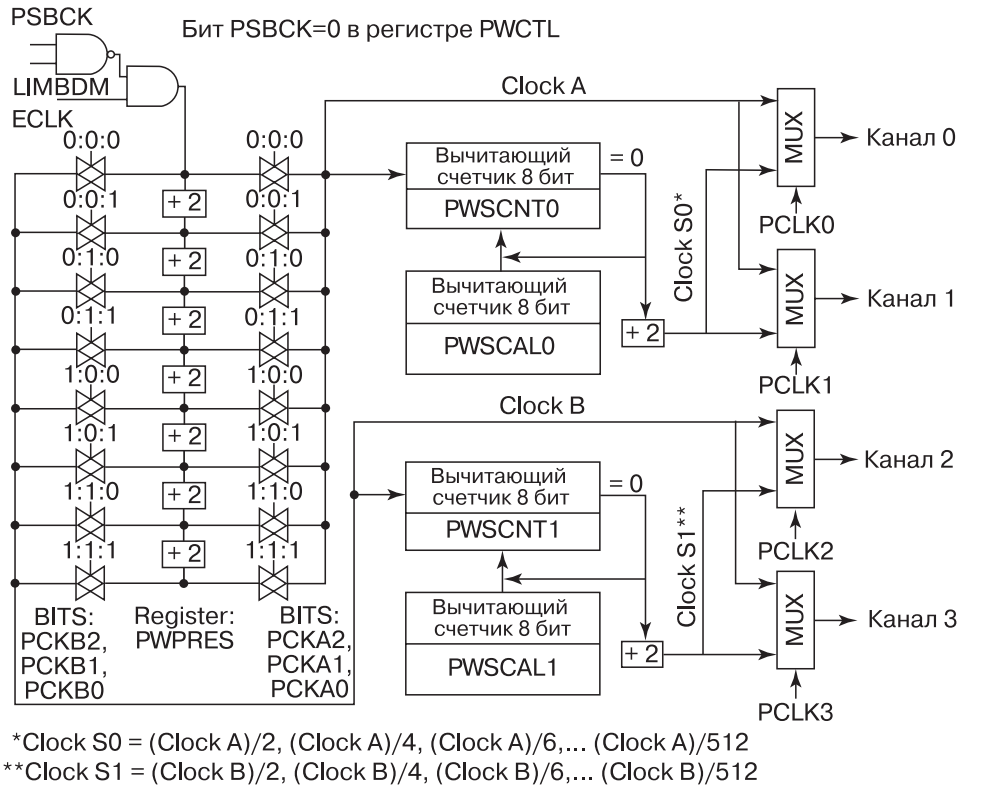


Рис. 4.100. *Структурная схема системы тактирования модуля PWM*

формируются две импульсные последовательности CLOCK_A и CLOCK_B. Причем сигнал CLOCK_A используется для тактирования каналов 0 и 1, а CLOCK_B – каналов 2 и 3. Коэффициенты деления для последовательностей CLOCK_A и CLOCK_B задаются битами PCKA[2:0] и PCKB[2:0] регистра конфигурации PWCLK. Возможные значения коэффициентов деления приведены в таблице рис. 4.101.

PCKA2 (PCKB2)	PCKA1 (PCKB1)	PCKA0 (PCKB0)	Частота CLOCK_A , CLOCK_B
0	0	0	E
0	0	1	E + 2
0	1	0	E + 4
0	1	1	E + 8
1	0	0	E + 16
1	0	1	E + 32
1	1	0	E + 64
1	1	1	E + 128

Рис. 4.101. *Выбор частоты тактирования счетчиков каналов модуля PWM*

При необходимости коэффициент деления может быть увеличен посредством подключения дополнительного делителя на входе каждой пары каналов. Делитель 0 обслуживает каналы 0 и 1 (рис. 4.100), он формирует свой выходной сигнал из импульсной последовательности CLOCK_A. Делитель 1 обслуживает каналы 2 и 3 (рис. 4.100) и формирует выходной сигнал из последовательности CLOCK_B. Коэффициент деления каждого из этих делителей определяет код в регистрах PWSCAL0 и PWSCAL1.

4.24.4. Регистры модуля PWM

Множество регистров специальных функций, которые обслуживают модуль PWM, можно разделить на следующие группы:

- Регистры конфигурации;
- Регистр разрешения работы каналов;
- Регистр дополнительного делителя;
- Регистры делителей 0 и 1;
- Регистры счетчика каналов;
- Регистры периода каналов;
- Регистры коэффициента заполнения каналов;
- Регистр управления;
- Регистр специальных режимов модуля;
- Регистры работы с портом P.

Рассмотрим перечисленные регистры более подробно.

Регистр конфигурации PWCLK

Регистр конфигурации PWCLK выполняет две функции. Во-первых, его биты определяют, будут ли каналы модуля использоваться в 8-разрядном режиме, или эти каналы объединят попарно для работы в 16-разрядном режиме. Во-вторых, биты регистра назначают два коэффициента деления частоты для образования из импульсной последовательности E_CLOCK сигнала тактирования каналов 0 и 1 (CLOCK_A) и сигнала тактирования каналов 2 и 3 (CLOCK_B).

Формат регистра PWCLK представлен на рис. 4.102. Биты CON23 и CON01 определяют режимы работы пар каналов 2:3 и 0:1 соответственно. Если значение бита CON23 равно 0, то каналы 2 и 3 работают независимо друг от друга с 8-разрядным разрешением. При CON23 = 1 каналы 2 и 3 объединяются в один 16-разрядный генератор ШИМ-сигнала. Действие бита CON01 по отношению к каналам 0 и 1 аналогично.

Если каналы объединены в пару, то ШИМ-сигнал пары 0:1 формируется на линии 0 порта PORT P, пары 2:3 – на линии 2 порта PORT P. Однако управление полярностью выходных сигналов 16-разрядных ШИМ-генераторов осуществляется битами PPOL1 для пары 0:1 и PPOL3 для пары 2:3. Биты управления полярностью PPOLx располагаются в регистре PWPOL.

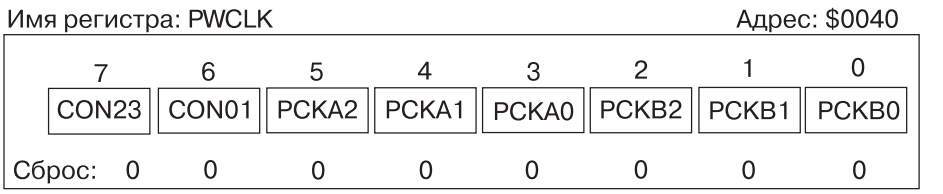


Рис. 4.102. *Формат регистра конфигурации PWCLK*

Имя регистра: PWPOL Адрес: $0041

7	6	5	4	3	2	1	0
PCLK3	PCLK2	PCLK1	PCLK0	PPOL3	PPOL2	PPOL1	PPOL0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.103. *Формат регистра конфигурации PWPOL*

Регистр конфигурации PWPOL

Регистр конфигурации PWPOL предназначен для задания активного уровня выходного сигнала каждого из четырех ШИМ-генераторов, а также для разрешения включения дополнительного делителя частоты для сигналов тактирования CLOCK_A и CLOCK_B. Формат регистра PWPOL приведен на рис. 4.103.

Каждому каналу поставлен в соответствие бит полярности сигнала PPOLx и бит выбора источника тактирования канала PCLKx (x – номер канала). Если бит PPOLx = 1, то активный уровень ШИМ-сигнала – высокий логический (рис. 4.98). При PPOLx = 0 на выходе канала формируется ШИМ-сигнал с низким логическим уровнем на интервале γT (рис. 4.98).

Если бит PCLKx равен 0, то для соответствующего канала в качестве источника тактирования назначается импульсная последовательность CLOCK_A (каналы 0 и 1) или CLOCK_B (каналы 2 и 3). При PCLKx = 0 соответствующий канал тактируется от дополнительного делителя частоты, который конфигурируется битами регистров PWSCNT0/PWSCNT1 и PWSCAL0/PWSCAL1.

Регистр разрешения работы каналов PWEN

Отдельные биты регистра конфигурации PWEN используются для активизации каналов модуля PWM. Формат регистра PWEN представлен на рис. 4.104. Если бит PWENx установлен в 1, то канал с номером «x» генерирует импульсную последовательность на соответствующем выводе порта PORT P. При PWENx = 0 канал находится в неактивном режиме, соответствующая линия порта PORT P может использоваться как линия ввода/вывода общего назначения.

Регистр дополнительного делителя PWPRES

Этот регистр используется только в специальных режимах работы МК. Его формат представлен на рис. 4.105. Более подробно мы не будем его рассматривать.

Прежде, чем приступить к рассмотрению двух следующих регистров, возвратимся к рис. 4.100 и вспомним, как организована система тактирования отдельных каналов модуля PWM. Источником тактирования каналов является импульсная последовательность E_CLOCK. Два программируемых делителя образуют из E_CLOCK сигналы тактирования CLOCK_A для каналов 0 и 1 и CLOCK_B для каналов 2 и 3. Причем каждый из каналов может тактироваться от указанного сигнала или напрямую, или через дополнительный делитель. Регистры PWSCNT0/PWSCNT1 и PWSCAL0/PWSCAL1 устанавливают режимы работы этих двух дополнительных делителей.

Имя регистра: PWEN Адрес: $0042

7	6	5	4	3	2	1	0
0	0	0	0	PWEN3	PWEN2	PWEN1	PWEN0
Сброс: 0	0	0	0	0	0	0	0

Рис. 4.104. *Формат регистра разрешения работы каналов PWEN*

Имя регистра: PWPRES Адрес: $0043

	7	6	5	4	3	2	1	0
	0	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.105. *Формат регистра дополнительного делителя PWPRES*

Регистры делителей PWSCNT0/PWSCNT1 и PWSCAL0/PWSCAL0

Делитель 0 обслуживает каналы 0 и 1 (рис. 4.100), он формирует свой выходной сигнал из импульсной последовательности CLOCK_A. Делитель 1 обслуживает каналы 2 и 3 (рис. 4.100) и формирует выходной сигнал из последовательности CLOCK_B. Коэффициент деления каждого из этих делителей определяет код в регистрах PWSCAL0 и PWSCAL1 соответственно. Коэффициент деления Kx (x – номер делителя) вычисляется по формуле:

$$Kx = (PWSCALx + 1)/2,$$

где PWSCALx – десятичный эквивалент кода в регистре PWSCALx

Код из регистров PWSCAL0 и PWSCAL1 автоматически загружается в вычитающие счетчики PWSCNT0 и PWSCNT1, когда последние обнуляются. Таким образом, счетчики выполняют функцию делителей частоты с плавно изменяющимся коэффициентом деления. Регистры счетчиков PWSCNT0 и PWSCNT1 доступны только для чтения. Форматы регистров PWSCAL0/PWSCAL1 и PWSCNT0/PWSCNT1 приведены на рис. 4.106 и 4.107.

Регистры счетчика каналов PWCNTx

При рассмотрении принципа действия отдельного канала генератора ШИМ-сигнала мы выяснили, что в формировании сигнала участвует непрерывно считающий 8-разрядный двоичный счетчик, текущий код которого сравнивается с кодом коэффициента заполнения и кодом периода. В регистрах PWCNTx (x – номер канала 0...3) отображается текущий код счетчика каждого канала. Формат регистров PWCNTx представлен на рис. 4.108. Регистры PWCNT0... PWCNT3 доступны только для чтения, попытка записи в регистр счетчика текущего кода вызывает сброс счетчика.

Регистры периода каналов PWPERx

Число регистров периода PWPERx равно числу каналов ШИМ в модуле PWM. Все регистры PWPER0...PWPER3 – 8-разрядные. Формат регистров PWPER0...PWPER3 приведен на рис. 4.109. Период следования ШИМ-сигнала каждого канала определяется кодом, который записан в соответствующем регистре периода PWPERx:

Имя регистра: PWSCAL0/PWSCAL1 Адрес: $0044/$0046

	7	6	5	4	3	2	1	0
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.106. *Формат регистров делителей PWSCALx*

Имя регистра: PWSCNT0/PWSCNT1 Адрес: $0045/$0047

	7	6	5	4	3	2	1	0
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.107. *Формат регистров вычитающих счетчиков PWSCNTx*

Имя регистра: PWCNT0/ PWCNT1/ PWCNT2/ PWCNT3
Адрес: $0048/$0049/$004A/$004B

	7	6	5	4	3	2	1	0
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.108. *Формат регистров счетчика каналов PWCNTx*

Имя регистра: PWPER0/ PWPER1/ PWPER2/ PWPER3
Адрес: $004C/$004D/$004E/$004F

	7	6	5	4	3	2	1	0
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.109. *Формат регистров периода каналов PWPERx*

- для фронтовой ШИМ период сигнала равен

$$T_{PWM} = (PWPERx + 1)/fx$$

- для центрированной ШИМ период сигнала равен

$$T_{PWM} = 2 \times (PWPERx + 1)/fx$$

где PWPERx – десятичный эквивалент кода в регистре периода PWPERx,
fx – частота тактирования счетчика канала с номером «x».

Регистры коэффициента заполнения каналов PWDTYx

Число регистров коэффициента заполнения PWDTYx также равно числу каналов ШИМ в модуле PWM. Все регистры PWDTY0...PWDTY3 – 8-разрядные. Формат регистров PWDTY0...PWDTY3 представлен на рис. 4.110. Длительность импульса в каждого канале определяется кодом, который записан в соответствующем регистре коэффициента заполнения PWDTYx:

- для фронтовой ШИМ длительность импульса равна

$$TIMP = (PWDTYx + 1) / [(PWPERx + 1) \times fx]$$

- для центрированной ШИМ длительность импульса равна

$$TIMP = 2 \times (PWDTYx + 1) / [(PWPERx + 1) \times fx]$$

Коэффициент заполнения в обоих режимах равен

$$\gamma = [(PWDTYx + 1)/(PWPERx + 1)] \times 100\%,$$

где PWDTYx – десятичный эквивалент кода в регистре коэффициента заполнения PWDTYx, PWPERx – десятичный эквивалент кода в регистре периода PWPERx, fx – частота тактирования счетчика канала с номером «x».

Регистр управления PWCTL

Формат регистра управления модулем PWMCTL представлен на рис. 4.111. В этом разделе мы рассмотрим только бит CENTR этого регистра. Этот бит определяет форму выходных сигналов всех каналов модуля PWM. Если бит CENTR = 1, то в каналах реализуется центрированная ШИМ, при CENTR = 0 выходные сигналы генерируются по способу фронтовой ШИМ.

Регистр специальных режимов PWTST

Этот регистр используется только в специальных режимах работы МК. Его формат представлен на рис. 4.112. Более подробно мы не будем его рассматривать.

Регистры работы с портом P

Обслуживание модуля PWM – альтернативная функция линий порта PORT P. Линии PP0...PP3 порта P могут использоваться не только в качестве выходов модуля PWM, но и как линии порта ввода/вывода общего назначения. Если работа соответствующих каналов модуля PWM разрешена, то линии порта P автоматически конфигуриру-

Имя регистра: PWDTY0/ PWDTY1/ PWDTY2/ PWDTY3

Адрес: $0050/$0051/$0052/$0053

	7	6	5	4	3	2	1	0
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.110. *Формат регистров коэффициента заполнения PWDTYx*

Имя регистра: PWCTL — Адрес: $0054

	7	6	5	4	3	2	1	0
	0	0	0	PSWAI	CENTR	RDPP	PUPP	PSBCK
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.111. *Формат регистра управления PWCTL*

Имя регистра: PWTST — Адрес: $0055

	7	6	5	4	3	2	1	0
	DISCR	DISCP	DISCAL	0	0	0	0	0
Сброс:	0	0	0	0	0	0	0	0

Рис. 4.112. *Формат регистра PWTST*

ются для работы в качестве выходов, на которых формируются ШИМ-сигналы. Если каналы модуля PWM не активизированы, то линии PP0...PP3 работают как обычные линии ввода/вывода. Направление передачи сигнала линиями PP0...PP7 определяется битами регистра направления передачи DDRP. Формат регистра DDRP приведен на рис. 4.113. Запись 1 в соответствующий разряд регистра DDRP конфигурирует линию на вывод. Порт P обслуживается регистром данных PORTP (рис. 4.113).

4.24.5. Примеры программирования модуля PWM

Изучая модуль PWM, читатель должен был заметить, что модуль требует инициализации достаточно большого числа регистров для определения параметров работы каналов ШИМ. Однако после активизации каналов их программное обслуживание состоит лишь в смене значений коэффициента заполнения и реже, периода следования выходной импульсной последовательности.

Рассмотрим последовательность действий, которую необходимо проделать для инициализации модуля PWM:

- Для конкретного приложения следует определить разрешающую способность генерируемого ШИМ-сигнала, т.е. число дискретных отсчетов частоты таткирования канала в периоде и длительности импульса выходного сигнала канала. На основании полученных данных следует определить, в каком режиме, 8-разрядном или 16-разрядном, Вы будете использовать каналы модуля PWM;
- Для конкретного приложения следует определить требуемую частоту генерируемого ШИМ-сигнала. На основе полученных данных определить структуру подсистемы тактирования каналов модуля PWM;
- Установить биты CON23 и CON01 в регистре PWCLK для выбора 8-разрядного или 16-разрядного режима работы;
- Определить, будете ли Вы использовать режим центрированной или режим фронтовой ШИМ. В соответствие с выбранным режимом установить бит CENTR в регистре PWCTL;
- Определить активный уровень ШИМ-сигнала, в соответствии с выбором установить биты PPOL0...PPOL3 в регистре PWPOL;

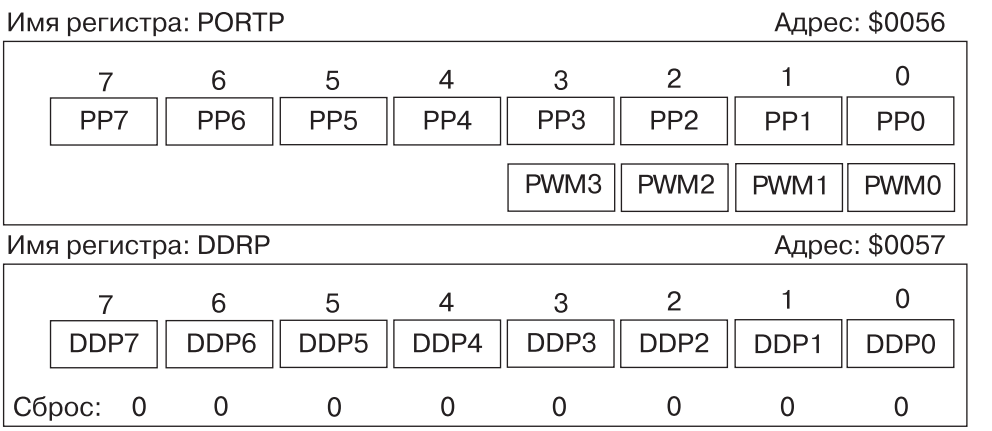


Рис. 4.113. *Формат регистров обслуживания порта PORTP*

- Назначить источники тактирования для каналов, для чего установить биты PCLK0... PCLK3 в регистре PWPOL;
- Установить коэффициенты деления для импульсных последовательностей CLOCK_A и CLOCK_B, используя для этого биты PCKA2... PCKA0 и PCKB2... PCKB0 регистра PWCLK;
- Установить для используемых каналов значения регистров периода и регистров коэффициента заполнения;
- Разрешить работу выбранных каналов модуля PWM, используя для этого биты PWEN0...PWEN3 регистра EPWM.

В конце данной главы в разделе самостоятельных заданий (задание 12) мы предложим Вам определить параметры инициализации модуля ШИМ самостоятельно. А пока приведем два примера.

Инициализация модуля PWM, пример 1

Определим параметры настройки модуля PWM для генерации ШИМ-сигнала с частотой 976 Гц и коэффициентом заполнения 66,7%.

Частоту тактирования канала ШИМ выберем равной 8МГц/32 = 250 кГц. Этой частотой будем тактировать 8-разрядный счетчик канала. Для формирования частоты 976 Гц потребуется 256 отсчетов частоты 250 кГц, что соответствует максимально возможному коэффициенту счета 8-разрядного счетчика периода канала.

Для формирования сигнала с коэффициентом заполнения 66,7% следует установить код периода, равный 256 отсчетам, а код коэффициента заполнения – 171 отсчету. Для формирования ШИМ-сигнала будем использовать канал 0 модуля PWM.

Программный фрагмент init_pwm.c производит начальную установку регистров специальных функций модуля PWM для генерации на выходе PP0 ШИМ-сигнала с частотой 976 Гц и коэффициентом заполнения 66,7%:

```
/*-------------------------------------------------------------------*/
/* Функция init_pwm задает начальные установки модуля PWM            */
/*-------------------------------------------------------------------*/
void ini_pwm(void)
{
PWTST = 0x00;       /*выбрать нормальный режим работы модуля PWM*/
PWCTL = 0x00;       /*выбрать режим фронтовой ШИМ*/
PWCLK = 0x28        /*канал 0 в 8-разрядном режиме, коэфф. деления*/
                    /* частоты E_CLOCK равен 32*/

PWPOL = 0x01;       /*установить высокий активный уровень сигнала*/
DDRP = 0xFF;        /*настроить порт P на вывод*/
PWEN = 0x01;        /*разрешить работу канала 0 модуля PWM */
PWPER0 = 255;       /*установить код периода*/
PWDTY0 = 171        /*установить код коэффициента заполнения*/
}
/*-------------------------------------------------------------------*/
```

Инициализация модуля PWM, пример 2

В начале раздела 4.24, мы рассматривали пример использования широтно-импульсной модуляции для управления серводвигателем. Для напоминания структурная схема управления серводвигателем приведена на рис. 4.114. В рассматриваемом примере серводвигатель приводит в движение рулевой механизм радиоуправляемой модели автомобиля. Вращение серводвигателя обеспечивает отклонение рулевого

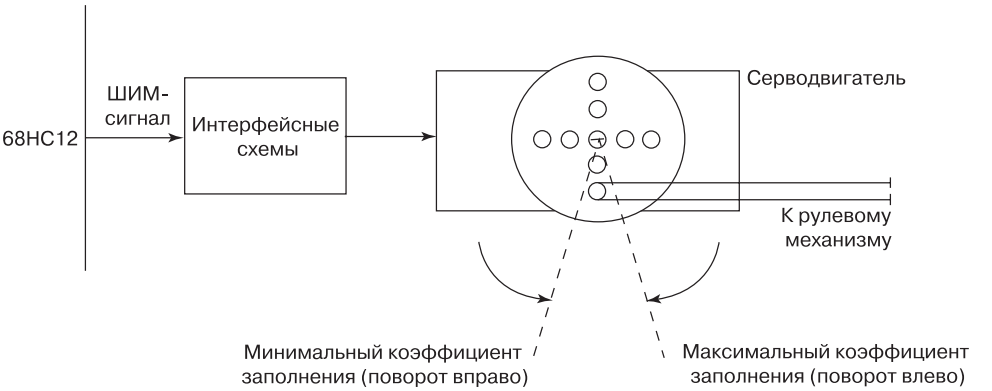


Рис. 4. 114. *Использование ШИМ для поворота рулевого колеса радиоуправляемой модели автомобиля.*

механизма на определенный угол от центрального положения. Этот угол определяется средним значением напряжения, которое прикладывается к двигателю. Напряжение регулируется способом широтноимпульсной модуляции, тогда среднее значение напряжения прямопропорционально коэффициенту заполнения ШИМ-сигнала. Значение коэффициента заполнения передается по радиоканалу. Изменение длительности импульсов напряжения, прикладываемого к серводвигателю, при сохранении частоты следования импульсов вызывает поворот рулевого механизма на определенный угол. Допустимый для рулевого управления диапазон регулирования коэффициента заполнения составляет от 4,5 до 10 %. При этом частота следования импульсов напряжения на двигателе должна составлять 50 Гц.

Определим параметры настройки модуля PWM для генерации ШИМ-сигнала с частотой 50 Гц и коэффициентом заполнения 4,5...10,0 %. Предположим, что частота импульсной последовательности E_CLOCK равна 8 МГц. Тогда период следования импульсов E_CLOCK составляет:

$$T_{ECLOCK} = 1/f_{ECLOCK} = 1/8\ МГц = 125\ нс$$

Период следования формируемого ШИМ-сигнала составляет:

$$T_{PWM} = 1/f_{PWM} = 1/50\ Гц = 20\ мс$$

Число периодов импульсной последовательности E_CLOCK, которое должно быть отсчитано для формирования периода ШИМ-сигнала, составляет:

$$K_{E\_CLOCK} = T_{PWM} / T_{E\_CLOCK} = 20\ мс/125\ нс = 160\ 000$$

Подсчитанное число слишком велико, чтобы его можно было разместить не только в 8-разрядном, но и 16-разрядном регистре. Поэтому при конфигурации модуля PWM необходимо воспользоваться программируемым делителем частоты E_CLOCK. Максимальное значение коэффициента этого делителя составляет 128: *160 000/128 = 1250*. Именно такое число отсчетов следует использовать для генерации ШИМ-сигнала с частотой 50 Гц. Число 1250 может быть размещено в 16-разрядном регистре, поэтому мы будем использовать объединение двух 8-разрядных каналов модуля PWM в один 16-разрядный. Аналогичным способом следует вычислить коды, соответствующие диапазону возможных коэффициентов заполнения 4,5...10,0 %.

Запишите код инициализации модуля PWM для управления серводвигателем рулевого механизма самостоятельно, используя блок схему рис. 4.115 и программный фрагмент init_pwm.

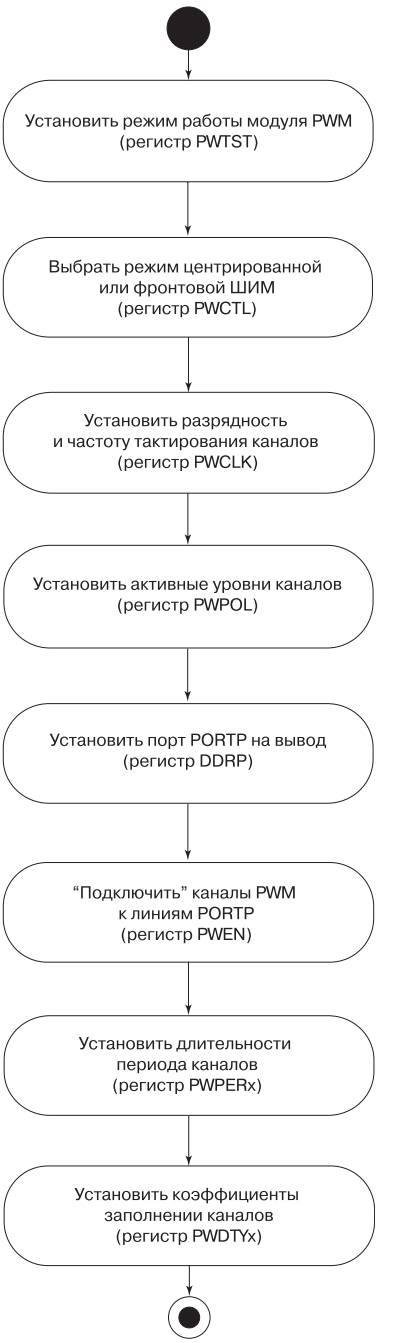


Рис. 4.115. *Последовательность действий МК при обслуживании модуля PWM*

4.25. Ограничение энергии потребления

Используя мобильный телефон, Вы должны были заметить, что он переходит в режим экономичного расхода энергии аккумуляторной батареи после того, как Вы перестали им пользоваться. Все современные телефоны оснащены красочным дисплеем с подсветкой. После завершения разговора подсветка дисплея автоматически выключается, и будет оставаться в выключенном состоянии до тех пор, пока пользователь не начнет нажимать на кнопки панели управления. Однако через некоторое время после последнего нажатия дисплей опять выключится.

Все описанные действия имеют своей целью экономию энергии аккумуляторной батареи в те промежутки времени, когда телефоном не пользуются. Телефон переходит в режим низкого энергопотребления («sleep mode» – режим сна) и остается в нем до тех пор, пока не поступит звонок или пользователь нажмет на кнопки. В главе 6 мы подробно рассмотрим, как экономить энергию батареи в микропроцессорных системах. Мы также увидим, что энергия потребления МК, выполненных по CMOS технологии, пропорциональна частоте системной шины МК. Некоторые значения для токов потребления в различных режимах работы МК приведены в таблице рис. 4.116.

Анализируя приведенные в таблице цифры, Вы можете увидеть ярко выраженную зависимость тока потребления МК от частоты тактирования. А также, что МК значительно снижает ток потребления в режимах ожидания WAIT и останова STOP.

4.25.1. Как остановить МК 68HC12

МК семейства 68HC12 переходит в режим ожидания WAIT сразу после исполнения команды WAI языка ассемблер. Перед тем, как перейти в режим ожидания, МК записывает в стек содержимое регистров центрального процессора и адрес возврата. Затем модуль системной интеграции отключает генератор тактирования центрального процессора, в то время как тактирование периферийных модулей МК продолжается. МК может быть выведен из состояния ожидания WAIT поступлением запроса на прерывание. Этот запрос может быть внешним, или внутренним, например, по какому либо событию в модуле таймера. После поступления запроса, МК перейдет из состояния WAIT в активный режим работы, восстановит значения кодов регистров центрального процессора из стека, а затем перейдет к исполнению подпрограммы прерывания.

МК переводится в режим останова STOP командой STOP языка ассемблер. Команда STOP исполняется только в том случае, если бит S в регистре признаков CCR центрального процессора сброшен. После поступления команды STOP центральный процессор сохраняет все регистры в стеке, а затем модуль системной интеграции полностью выключает систему тактирования МК. При полном отсутствии тактовых импульсов все логические элементы в составе полупроводниковой ИС микроконтроллера находятся в статическом режиме, т.е. в состоянии 0 или 1. Известно, что основные потери энергии в логических элементах происходят при переключениях из одного установившегося состояния в другое. Именно поэтому в состоянии STOP микроконтроллера, когда переключений элементов внутренней структуры нет, энергия потребления резко падает. Данные таблицы рис. 4.116 показывают, что в режиме останова STOP ток потребления МК снижается на три порядка, т.е. примерно в тысячу раз.

Частота внутренней системной шины			
Максимальные значения токов потребления	**2 МГц**	**4 МГц**	**8 МГц**
Активный режим работы Однокристальный режим Расширенный режим	15 мА 25 мА	25 мА 45 мА	45 мА 70 мА
Режим ожидания Wait (все периферийные модули отключены) Однокристальный режим Расширенный режим	1,5 мА 4 мА	3 мА 7 мА	5 мА 10 мА
Режим останова STOP Однокристальный режим, (система тактирования выключена) –40 +85 (°C) +85 +105 (°C) +105 +125 (°C)	10 мА 25 мА 50 мА	10 мА 25 мА 50 мА	10 мА 25 мА 50 мА

Рис. 4.116. *Ток, потребляемый МК семейства 68HC12 в различных режимах работы*

4.25.2. Как вывести МК 68HC12 из состояния пониженного энергопотребления

Поступление запроса на внешнее прерывание способно вывести МК семейства 68HC12/HCS12 из режима WAIT или STOP. Однако для подключения сигналов внешних запросов могут быть использованы не только входы IRQ и XIRQ. Линии портов PORT D, PORT Y и PORT J могут быть конфигурированы таким образом, что перепад сигнала на них из 1 в 0 вызовет переход МК из режима пониженного энергопотребления в активный режим работы.

Для реализации функции пробуждения МК все перечисленные регистры снабжены двумя дополнительными регистрами управления. Это регистр разрешения функции прерывания пробуждения KWIEx и регистр флагов прерывания пробуждения KWIFx (x – буквенное обозначение порта). Поскольку формат этих дополнительных регистров для всех перечисленных портов одинаков, мы рассмотрим только регистры порта D. На рис. 4.117 приведен формат всех четырех регистров специальных функций для обслуживания линий порта PORT D:

- Регистра данных PORTD;
- Регистра направления передачи DDRD;
- Регистра разрешения функции прерывания пробуждения KWIED;
- Регистра флагов прерывания пробуждения KWIFD.

Установка одного из битов регистра KWIED в 1 разрешает генерацию запроса на прерывание в момент перехода сигнала на одноименной линии из 1 в 0. Одновременно устанавливается флаг в регистре KWIFD. Прерывания от порта PORT D не имеют собственного вектора. Эти прерывания в МК связаны с прерыванием по входу IRQ. Поэтому, чтобы прерывания от линий порта PORT D были обслужены,

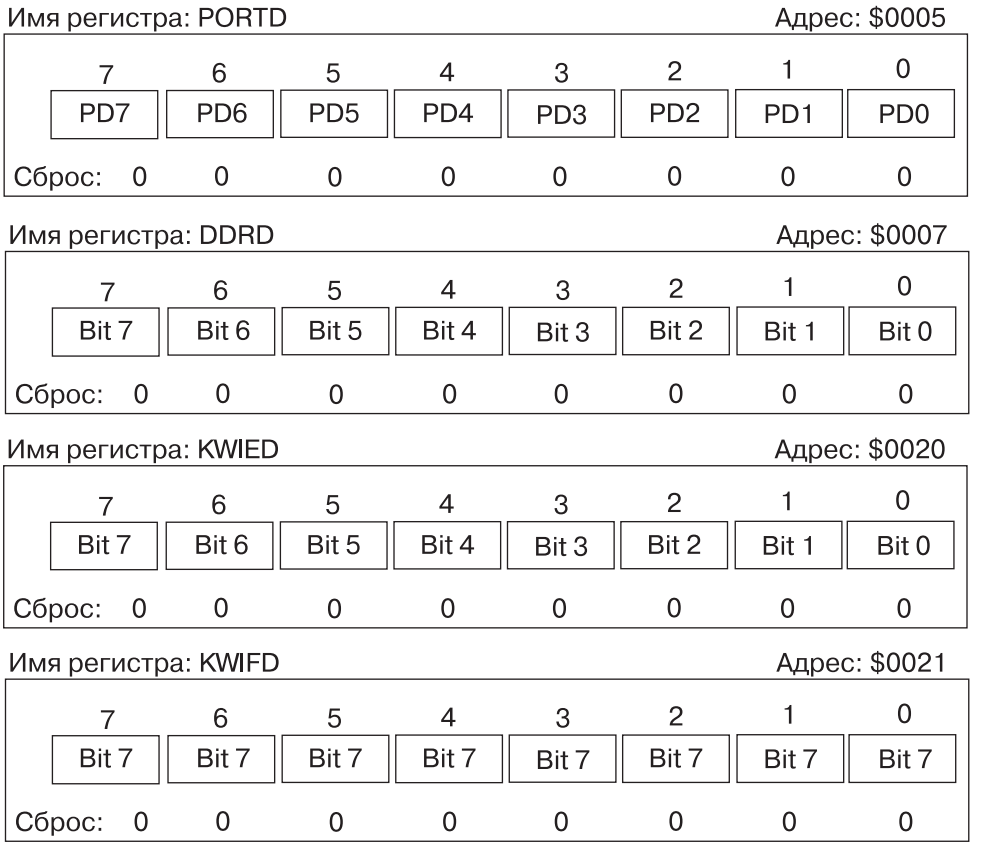


Рис. 4.117. *Формат регистров обслуживания порта PORD*

необходимо установить бит IRQEN и разместить подпрограмму прерывания в памяти по вектору внешнего прерывания IRQ. Различные модели МК семейства 68HC12/HCS12 несколько отличаются по конфигурации функции внешнего прерывания пробуждения. Так в МК 68HC12A4 порты PORT J и PORT H обладают собственными векторами прерывания, что позволяет упростить подпрограмму прерывания по входу IRQ.

Обсудив возможность вывода МК из режима пониженного энергопотребления посредством дополнительных внешних прерываний, не следует забывать, что любой МК может быть переведен в активный режим работы подачей сигнала внешнего сброса.

Уменьшение потребляемой микроконтроллером в процессе работы мощности возможно также снижением частоты тактирования посредством перепрограммирования коэффициентов умножителя частоты или выбора специального режима тактирования «slow mode». Такое решение позволит оставить центральный процессор в работе, в отличие режимов WAIT и STOP, в которых центральный процессор останавливается.

4.26. Советы по использованию платы отладки MC68EVB912B32

В этом параграфе мы более подробно рассмотрим аппаратные средства отладочной платы MC68EVB912B32, в том числе, какие возможности предоставляет конструкция платы по присоединению к ней внешних интерфейсных компонентов.

Плата развития (evaluation board) MC68EVB912B32 разработана компанией Motorola для практического изучения МК семейства 68HC12 и отладки прикладных программ управления для относительно несложных приложений. На плате установлен МК MC68HC912B32 и дополнительные схемы, обеспечивающие функционирование МК и его связь с персональным компьютером для целей загрузки кодов разработанной программы и ее отладки. Внешний вид платы MC68EVB912B32 и схема размещения на ней компонентов представлены на рис. 4.118.

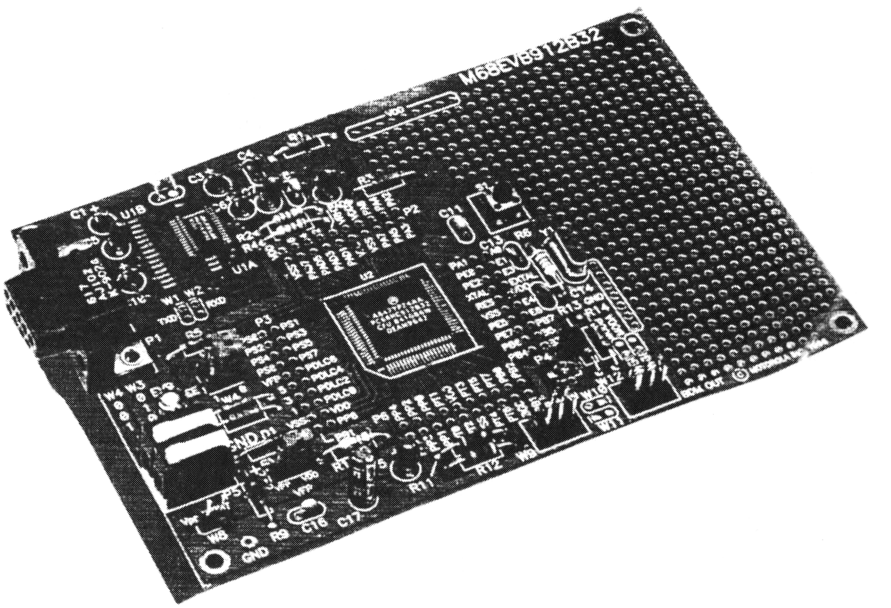
МК B32 схемотехнически установлен в режим взаимодействия с персональным компьютером по последовательному интерфейсу. Для организации взаимодействия необходимо установить соединение между портом P1 платы и COM-портом компьютера, используя для этого стандартный кабель с разъемами DB-9. Обмен производится со скоростью 9600 бод, в формате 8 бит данных, один стоповый бит, без контроля паритета. Обмен данными на более высоких скоростях невозможен.

Для работы с платой пользователь должен загрузить в персональный компьютер программный пакет класса «Интегрированная среда разработки ПО для микропроцессорных систем», например ICC12 от компании ImageCraft. Этот пакет позволит вам ввести исходный текст программы на языке ассемблера или на Си, произвести его компиляцию, обработать программой линковщика и получить файл исполняемых кодов. Далее следует загрузить полученный программный код в память МК отладочной платы, после чего выполнить внешний сброс МК. И МК начнет исполнять загруженную программу без участия персонального компьютера.

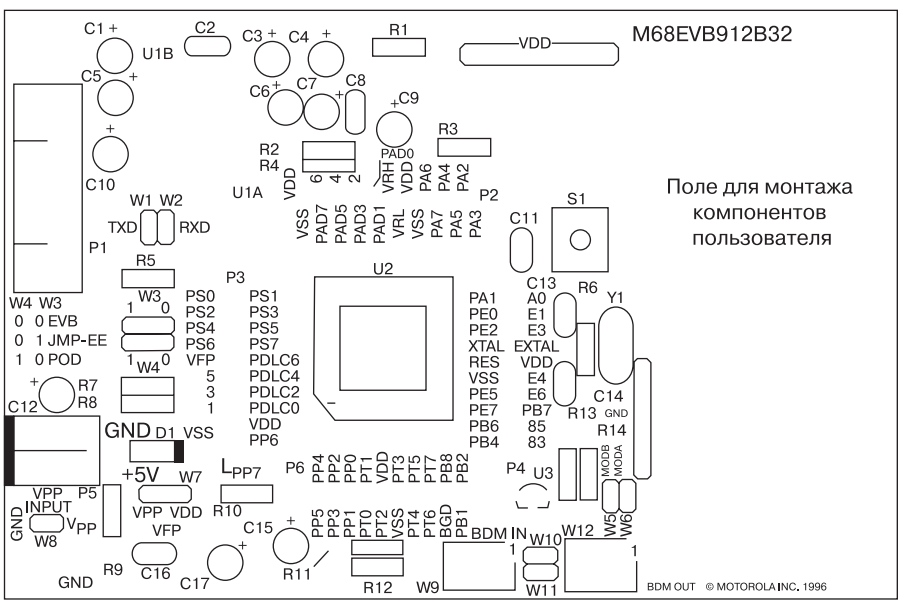
Отладочная плата MC68EVB912B32 имеет встроенную программу монитора отладки D-Bug12. Эта программа размещается в области Flash ПЗУ МК. Если пользователь желает поместить отлаживаемую программу также в область Flash, то он должен сначала стереть область Flash ПЗУ, а затем записать в эту область коды прикладной программы. При этом функции отладки будет невозможно реализовать.

МК B32 и другие ИС, размещенные на плате MC68EVB912B32, выполнены на основе CMOS технологии. Эти ИС достаточно чувствительны к статическому электрическому заряду, поэтому следует соблюдать повышенную осторожность при присоединении к плате внешнего оборудования. Более подробно вопросы безопасной коммутации внешних устройств к платам с МК мы рассмотрим в главе 5.

Отладочная плата MC68EVB912B32 имеет свободное монтажное поле для размещения дополнительных ИС, например ИС ЦАП, и установочных изделий, таких как дополнительные переключатели и светодиодные индикаторы. Для подключения этих компонентов к выводам МК можно использовать группу штыревых контактов 2×10, которую следует впаять в подготовленную линейку металлизированных отверстий. На каждый штыревой контакт может быть надета пружинящая клемма с подпаянным к ней монтажным проводом (рис. 4.119).



а) Внешний вид



б) Схема размещения компонентов

Рис. 4.118. *Отладочной плата MC68EVB912B32*

Инженер кафедры вычислительной техники университета в Вайоминге Lew Sircin усовершенствовал данную технологию. Он предложил произвести автоматизированную разводку платы с дополнительными компонентами с выводом необходимых соединений на аналогичную линейку контактов 2×10, но только на дополнительной плате установить ответную часть разъема для штыревых контактов платы отладки. Соединив две платы через этот разъем, Вы получите мезонинную конструкцию, в которой одна плата расположена над другой (рис. 4.119).

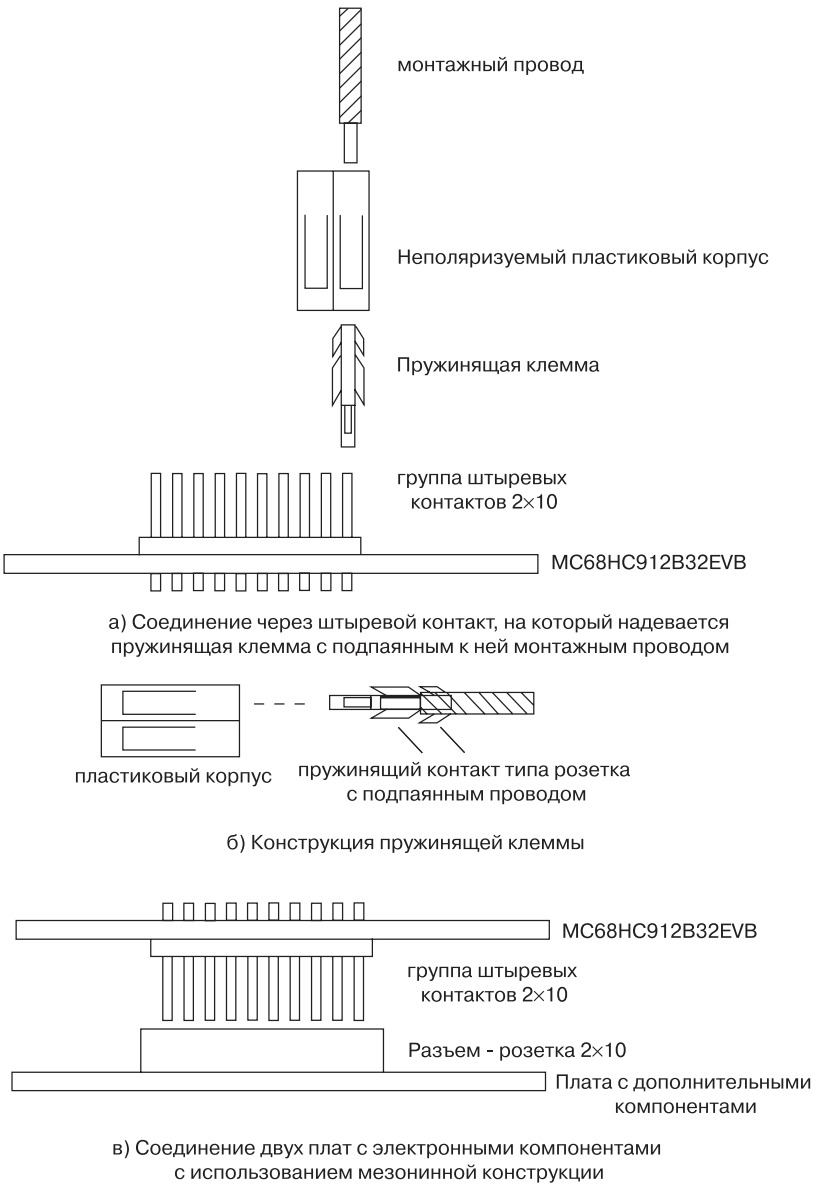


Рис. 4.119. *Способы подключения дополнительных компонентов к отладочной плате*

4.27. Заключение по главе 4

В данной главе мы достаточно подробно рассмотрели структуру, принцип действия и регистры управления периферийных модулей в составе МК семейства 68HC12/HCS12. Мы изучили резидентную память МК, систему тактирования, порты ввода/вывода, подсистемы таймера, аналого-цифрового преобразования, контроллеры последовательных интерфейсов, модуль ШИМ. Мы также рассказали Вам об особенностях обслуживания прерываний и о состоянии сброса в МК семейства 68HC12/HCS12.

Вы научились измерять временные интервалы и генерировать импульсные последовательности с использованием режимов входного захвата и выходного сравнения таймера. Вы также освоили проведение измерений аналоговых сигналов с использованием встроенного в МК модуля аналого-цифрового преобразования. Вы познакомились с принципами организации обмена между интеллектуальными электронными устройствами в последовательном коде и со способом широтно-импульсной модуляции, который позволяет регулировать напряжение, прикладываемое к обмоткам исполнительного двигателя.

Для каждого из периферийных модулей мы рассмотрели эталонный учебный пример программного кода на Си, который далее может быть использован Вами при разработке учебных и реальных прикладных программ.

4.28. Что еще почитать?

1. Motorola. 68HC12 M68EVB912B32 Evaluation Board User's Manual (68EVB912B32-UM/D). Motorola Inc., 1997.
2. Motorola. HC12 M68HC12B Family Advance Information, (M68HCI2B/D), Motorola Inc., 2000.
3. ImageCraft. ImageCraft C Compiler and Development Environment for Motorola HC12, . Version 6. Palio Alto, CA: ImageCraft, Inc.
4. Pack, Daniel, and Steven Barrett. 68HC12 Microcontroller: Theory and Application, Upper Saddle River, NJ: Prentice Hall 2002.
5. Wakerly, John. Digital Design principles and Practices, Upper Saddle River, NJ: Prentice Hall, 2001.

4.29. Вопросы и задания

Основные

1. Объясните назначение каждого функционального модуля МК 68HC12B32.
2. Семейство 68HC12 включает несколько моделей МК. Каковы различия между отдельными моделями?
3. Сколько ячеек памяти занимает блок регистров специальных функций в МК семейства 68HC12? В какой области памяти, и по каким адресам этот блок располагается?

4. Опишите все функции линий порта PORT AD.
5. Каков объем встроенной памяти типа EEPROM в микроконтроллере семейства 68HC12?
6. Какое состояние МК называют прерыванием? Кратко опишите последовательность действий МК по обслуживанию прерывания.
7. Каковы различия между маскируемыми и немаскируемыми прерываниями. Какие из этих прерываний имеют более высокий приоритет?
8. Какова разрешающая способность встроенного АЦП для МК семейства 68HC12? Может ли она изменяться и при каких условиях?
9. Каково назначение регистра направления передачи порта ввода/вывода?
10. Число разрядов счетчика временной базы равно 16. Чему равен его коэффициент счета?
11. Счетчик временной базы тактируется частотой 0,5 МГц. Чему равен период переполнения счетчика?

Исследовательские

1. При изучении подсистемы входного захвата мы рассмотрели пример измерения длительности импульса с высоким логическим уровнем, которая по условию задачи не должна была превышать периода переполнения счетчика временной базы таймера. Разработайте блок схему алгоритма для этого примера.
2. Напишите программу на Си для реализации задачи предыдущего вопроса. Программа должна включать все операции по инициализации модуля таймера. Считайте, что таймер тактируется частотой 2 МГц, исследуемый импульсный сигнал подается на вход канала 2 модуля таймера.
3. При изучении подсистемы выходного сравнения мы рассмотрели пример генерации на одном из выходов МК единичного импульса с низким логическим уровнем. По условию задачи длительность этого импульса не превышала периода переполнения счетчика временной базы таймера. Разработайте блок схему алгоритма для этого примера.
4. Напишите программу на Си для реализации задачи вопроса № 3. Программа должна включать все операции по инициализации модуля таймера. Считайте, что таймер тактируется частотой 2 МГц, импульсный сигнал генерируется на на выходе канала 2 модуля таймера.
5. Напишите программу на Си, которая изменяет код на выходе порта А в порядке инкрементирующего двоичного счетчика. Временной интервал между двумя соседними изменениями кодов должен составлять 30 мс.
6. Напишите программу на Си, которая изменяет код на выходе порта А сначала в порядке инкрементирующего двоичного счетчика от 0x00 до 0xFF, а затем в обратном порядке 0xFF...0x00. Временной интервал между двумя соседними изменениями кодов должен составлять 30 мс.
7. В рассмотренном при изучении подсистемы таймера примере измерения периода некоторого импульсного сигнала мы предположили, что этот период будем меньше, чем период переполнения счетчика временной базы. Измените предложенный ранее алгоритм таким образом, чтобы измерения проводились для сигналов с периодом, превышающим период переполнения счетчика временной базы.

8. Укажите ограничения по минимальному и максимальному значению периода сигнала для предложенного Вами алгоритма в задании № 7.
9. В параграфе 4.14.5, посвященном счетчику внешних событий в составе модуля таймера, мы обсудили способ измерения скорости движения велосипеда с использованием датчика Холла. Разработайте блок схему алгоритма, напишите текст программы на Си для реализации такого измерителя скорости.
10. В примере использования модуля меток реального времени (раздел 4.15) мы программно отсчитывали 122 периода модуля RTI по 8,196 мс для формирования интервала в 1 с. Сколько отсчетов необходимо было бы сделать для отсчета 24 часов? Сколько 8-разрядных ячеек памяти понадобилось бы задействовать для этой задачи ?
11. В разделе 4.18 мы рассмотрели программный фрагмент для инициализации контроллера SCI и передачи с его помощью ASCII кодов некоторых символов. Предположите, что Вам требуется организовать связь по последовательному асинхронному интерфейсу двух МК семейства 68HC12. Нарисуйте функциональную схему соединения микроконтроллеров. Разработайте две блок схемы алгоритмов для передающего и принимающего контроллеров. Напишите программы для обоих МК на Си.
12. В параграфе 4.24 мы обсудили способ управления исполнительным двигателем рулевого управления радиоуправляемой модели автомобиля с использованием ШИМ. Частота импульсного напряжения, прикладываемого к двигателю, составляет 50 Гц. Диапазон изменения коэффициента заполнения при полном ходе механизма рулевого управления составляет 4,5...10,0 %. Разработайте блок схему алгоритма, напишите текст программы на Си для управления рулевым механизмом. Для решения задачи предположите, что код задания угла поворота рулевого механихма в 8-разрядном формате поступает на входы порта A.

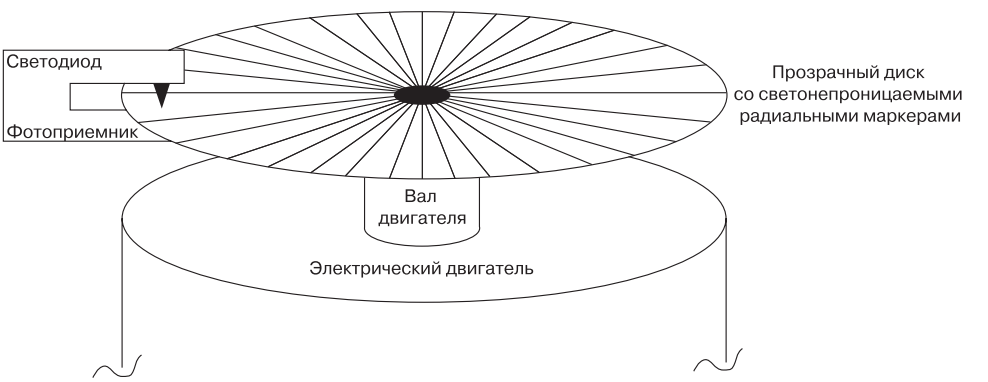


Рис. 4.120. *Электрический двигатель с оптическим датчиком скорости*

13. Разработайте блок схему алгоритма и напишите текст программы на Си для расчета скорости вращения двигателя (об/мин) по сигналам оптического датчика скорости. Принцип действия датчика поясняет рис. 4.120. Прозрачный пластмассовый диск закреплен на валу двигателя. Диск поделен светонепроницаемой краской на секторы. Диск помещен между излучающим светодиодом и фотоприемником. Если напротив светодиода находится черная полоса диска, то на выходе электронной схемы фотоприемника формируется логический 0. Если напротив светодиода располагается прозрачная полоса диска, то выход электронной схемы фотоприемника устанавливается в 1. В результате, когда двигатель вращается, на выходе электронной схемы формируется импульсная последовательность, частота которой прямопропорциональна скорости вращения двигателя. Для решения задачи предположите, что оптический датчик скорости формирует 200 импульсов на один оборот двигателя.

Глава 5

ОСНОВЫ СОПРЯЖЕНИЯ МК С УСТРОЙСТВАМИ ВВОДА/ВЫВОДА

ПОСЛЕ ИЗУЧЕНИЯ ГЛАВЫ ВЫ СМОЖЕТЕ:

ЦЕЛИ

- Описать электрические характеристики портов ввода/вывода МК 68HC12.
- Определить требования к электрическим характеристикам периферийных компонентов для подключения их к портам ввода/вывода МК;
- Разработать схему подключения и программу для ввода в МК информации о состоянии переключателей и матричных клавиатур;
- Грамотно применить методы программной и аппаратной защиты от механического дребезга контактов;
- Выполнить подключение к МК 68HC12 устройств индикации, таких как светодиоды, светодиодные семисегментные индикаторы, жидкокристаллические цифро-буквенные и графические индикаторы. Написать программы для вывода информации на перечисленные устройства индикации;
- Разработать схему подключения и алгоритм программного обслуживания датчиков различных физических величин.

5.1. Электрические характеристики МК 68HC12 328
5.2. Устройства дискретного ввода: кнопки, переключатели, клавиатуры 334
5.3. Устройства индикации: светодиодные индикаторы, индикаторы выхода с тремя состояниями 342
5.4. Программное обслуживание дискретных входов и выходов 346
5.5. Подавление механического дребезга переключателей 346
5.6. Жидкокристаллические индикаторы 352
5.7. Управление электрическим двигателем 367
5.8. Кодовый замок 371
5.9. Интерфейс МК с аналоговыми датчиками 378
5.10. Интерфейс RS-232 380
5.11. Заключение по главе 5 381
5.12. Что еще почитать? 382
5.13. Вопросы и задания 382

Какие устройства Вы можете подключить к выводам микроконтроллера 68HC12 напрямую, без дополнительных цепей согласования? Если у Вас уверенность, что входные и выходные буферы внутри МК при этом останутся работоспособными? Ваша интуиция подсказывает, что, наверное, без дополнительных цепей согласования могут быть подключены логические ИС, выполненные по той же технологии, что и МК семейства HC12. Например, можно без опаски подключить элемент 74HC00, выполняющий функцию 2И-НЕ. А что Вы скажете о возможном числе этих элементов? Можно ли одновременно подключить 20, 30 или 50 таких элементов? А как насчет элементов другой серии? Например, можно ли подключить к выходу МК 68HC12 всего один элемент И-НЕ модели 74LS00? Основываясь на примерах предыдущей главы, Вы также должны предположить, что управление электрическим двигателем вряд ли возможно непосредственно с выходов МК. Для этого потребуются дополнительные цепи согласования и дополнительный источник электрической энергии, которая будет преобразована двигателем механическую энергию. Как определить технические требования к подобным цепям, а затем разработать их?

В этой главе мы ответим на поставленные вопросы. Мы подробно исследуем электрические характеристики выходов МК семейства 68HC12. Вы должны детально освоить этот материал, поскольку он является основой для проектирования электронных цепей сопряжения МК с различными устройствами ввода и вывода. Далее мы рассмотрим типовые способы подключения к МК наиболее часто используемых устройств ввода и вывода, научимся составлять программы для взаимодействия МК с этими устройствами. В завершении главы мы рассмотрим пример реализации на МК кодового замка. В процессе изучения мы, как и предыдущих главах, будем использовать МК семейства 68HC12, однако все рассмотренные методики полностью применимы и к МК семейства HCS12.

5.1. Электрические характеристики МК 68HC12

Микроконтроллеры 68HC12 принадлежат к семейству интегральных схем HC, выпускаемых компанией Motorola по технологии «high-speed CMOS». CMOS (Complementary Metal Oxide Semiconductor) или в русскоязычной терминологии КМОП – это технология производства цифровых интегральных схем на основе Комплементарных полевых транзисторов со структурой Металл – Окисел – Полуп-

роводник. Семейство HC объединяет цифровые ИС различной степени интеграции: от простых логических элементов, счетчиков, дешифраторов до микроконтроллеров с архитектурой различной сложности. Все элементы, принадлежащие к семейству HC, электрически совместимы, поэтому сопряжение МК 68HC12 с другими элементами семейства HC не вызовет у Вас затруднений. Однако, если в процессе проектирования Вам потребуется подключить к выводам МК 68HC12 интегральные схемы, которые не принадлежат к семейству HC, то Вы должны провести анализ на совместимость электрических характеристик МК и этих ИС.

Производители электронных компонентов обычно указывают их электрические и динамические характеристики в справочном листе, Для проведения анализа о возможности сопряжения компонентов Вам потребуются следующие восемь параметров (рис. 5.1):

- V_{OH} – минимальное выходное напряжение логической 1;
- V_{OL} – максимальное выходное напряжение логического 0;
- I_{OH} – максимальный выходной ток логической 1;
- I_{OL} – максимальный выходной ток логического 0;
- V_{IH}: – минимальное входное напряжение логической 1;
- V_{IL}: – максимальное входное напряжение логического 0;
- I_{IH} – максимальный входной ток логической 1;
- I_{IL} – максимальный входной ток логического 0.

Необходимо знать численные значения параметров из приведенного списка для всех ИС, которые подлежат объединению в систему. В нашем случае одной из таких ИС обязательно является МК семейства 68HC12, численные значения обсуждаемых параметров для которого приведены ниже:

- V_{OH} = 4,2 В;
- V_{OL} = 0,4 В;
- I_{OH} = – 0,8 мА;
- I_{OL} = 1,6 мА;
- V_{IH} = 3,5 В;
- V_{IL} = 1,0 В;
- I_{IH} = 10 мкА;
- I_{IL} = – 10 мкА.

Обратите внимание, некоторые токи в этом списке указаны со знаком минус. Это означает, что соответствующий ток вытекает из МК. А токи с положительным знаком втекают в МК. Очень легко запомнить это условное обозначение, воспользовавшись аналогией с собственными денежными средствами. Финансовые поступления на ваш счет Вы рассматриваете со знаком плюс, а траты, т.е. вытекающие финансы, – со знаком минус.

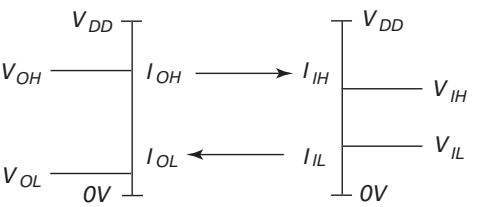


Рис. 5.1. *Номограмма электрической совместимости логических элементов*

5.1.1. Нагрузочные характеристики

Предположим, что некая периферийная ИС присоединена к выводу МК. Если входной ток этой ИС превышает максимальный выходной ток вывода МК, то могут возникнуть проблемы с формируемыми микроконтроллером уровнями напряжений логической 1 или логического 0. Выходные токи МК различаются в состоянии логической 1 и логического 0, поэтому рассмотрим эти два случая по отдельности.

Если на выходе МК формируется напряжение низкого логического уровня V_{OL}, то ток нагрузки I_{OL} положительный, т.е. ток нагрузки втекает в МК. Интегральная схема, присоединенная к выводу МК, формирует вытекающий ток I_{IL}, который и является током нагрузки вывода МК. Если величина входного тока ИС I_{IL} превышает указанное в листе электрических характеристик значение максимального тока нагрузки I_{OL}, то выходное напряжение V_{OL} на выходе МК может повыситься. Это явление иллюстрируют выходные характеристики МК в состоянии логического 0 (рис. 5.2,а). При определенном токе нагрузки I_{OL} напряжение на выводе МК может превысить значение входного напряжения логического нуля V_{IL} для периферийной ИС. Тогда периферийная ИС будет поставлена в ненормированный режим работы, ее работа в соответствии с техническим описанием не гарантируется.

Если на выходе МК формируется напряжение высокого логического уровня V_{OH}, то ток нагрузки I_{OH} отрицательный, т.е. ток нагрузки вытекает из МК. Входной ток I_{IL} присоединенной ИС является втекающим. Таким образом, ток нагрузки I_{OH} вытекает из МК и втекает в периферийную ИС. Если величина входного тока ИС I_{IH} превышает значение максимального тока нагрузки I_{OH}, то выходное напряжение V_{OH} на выходе МК может понизиться. Это явление иллюстрируют выходные характеристики МК в состоянии логической 1 (рис. 5.2,б). При определенном токе нагрузки I_{OH} напряжение на выводе МК может стать ниже минимально допустимого значения входного напряжения логической единицы V_{IH} для периферийной ИС. И мы опять поставим периферийную ИС в ненормированный режим работы, при котором ее функционирование может не соответствовать желаемому.

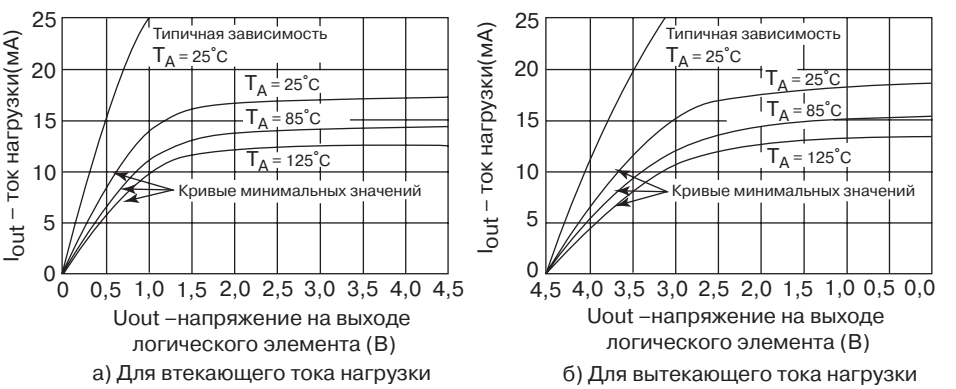


Рис. 5.2. *Нагрузочные характеристики логического элемента, выполненного по технологии HC CMOS*

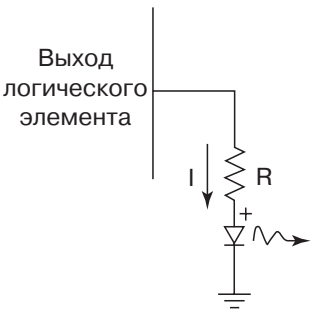


Рис. 5.3. *Неудачная схема подключения светодиода к логическому элементу*

В справочных данных компания Motorola/Freescale Semiconductors обычно указывает, что максимальный ток нагрузки для каждого из выводов портов равен ±25 мА. Однако следует понимать, что это всего лишь максимальный ток, который могут выдержать транзисторы выходного буфера линии порта. При таком токе нормированные значения выходного напряжения линии в состоянии логического 0 или логической 1 не гарантируются.

Во многих приложениях МК должен управлять некоторым устройством, входные напряжения и токи которого, превышают выходные параметры МК. Что делать в этом случае? Ответ категоричный. Вы, как разработчик системы, должны убедиться, что подключаемое устройство не превышает нагрузочных характеристик используемых линий МК. В противном случае нельзя выполнять непосредственное соединение, и необходимо разработать цепи усиления и согласования.

5.1.2. Что произойдет, если Вы должным образом не учтете электрические характеристики периферийных ИС?

Давайте рассмотрим очень простой пример. Мой близкий друг, доктор Jim Rasmussen (J.R.) работал инженером-электроником в течение 20 лет. В свое время, возвратясь из рядов вооруженных сил, он поступил на работу по контракту в одну известную фирму. Его первым заданием было спроектировать панель аварийных состояний. Эта панель должна была иметь несколько логических выходов, состояние которых отображалось светодиодными индикаторами. Одна из возможных схем подключения светодиода к выходу логической ИС показана на рис. 5.3. Давайте предположим, что вместо этой логической схемы используется выход МК семейства 68HC12. Видите ли Вы какие-либо проблемы в этой схеме? Для ее обсуждения давайте сначала более подробно остановимся на электрических характеристиках индикаторных светодиодов.

Светодиоды видимого спектра излучения в настоящее время доступны во множестве цветовых решений. Они могут светиться красным, желтым, зеленым, синим, белым и оранжевым цветами. Светодиодные индикаторы характеризуются относительно малой мощностью потребления в сочетании с большим сроком службы. Светодиод имеет два вывода: анод и катод. Во время излучения напряжение между анодом и катодом должно быть положительным.

Светодиоды характеризуются двумя основными параметрами: рабочим током и напряжением прямого смещения. Типичные значения рабочих токов светодиодов лежат в диапазоне от 10 до 15 мА, при этом прямое падение напряжения составляет 1,5 В. Для того, чтобы светодиод излучал, необходимо подключить его к источнику напряжения более 1,5 В и при этом обеспечить протекание прямого рабочего тока 10...15 мА.

Посмотрим, как эти условия выполняются в приведенной схеме (рис. 5.3)? Если на выходе МК формируется напряжение логической 1, то протекание тока через светодиод в требуемом направлении возможно. Величина выходного напряжения логической единицы V_{OH} превышает 1,5 В, поэтому условие по прямому напряжению в этой схеме выполнено. А как насчет тока? Можно рассчитать величину сопротивления R, при которой в цепи светодиода будет протекать ток 10...15 мА, и светодиод будет светиться. Но тогда понизится выходное напряжение V_{OH}, и его величина может стать меньше, чем входное напряжение логической 1 V_{IH}. Тогда подключение к такому выходу других логических схем невозможно, поскольку их надежная работа не гарантируется. Они просто могут не распознать наличие на выходе сигнала высокого логического уровня, и защитные исполнительные устройства не сработают.

Как выйти из сложившейся ситуации? Мы расскажем, как правильно подключать к МК светодиоды в разделе 5.3.1. А сейчас займемся грамотным сопряжением МК и цифровых ИС.

5.1.3. Входные и выходные характеристики логических элементов

В этом параграфе мы рассмотрим, как правильно подключить логический элемент одной серии к элементу другой серии. Электрические характеристики логических элементов, принадлежащих к разным сериям, отличаются друг от друга. Для того, чтобы ответить на вопрос о возможности соединения некоторого количества элементов разных серий, необходимо проделать следующие шаги:

- Убедиться, что выходные напряжения логической 1 и логического 0 элемента – источника сигнала совместимы со входными напряжениями 1 и 0 подключаемого элемента.
- Убедиться, что выходные токи элемента – источника сигнала превышают входные токи подключаемого элемента. Необходимо проанализировать как состояние логического 0, так и логической 1.
- Определить, сколько логических элементов будет подсоединяться к одному выходу элемента – источника сигнала.

Давайте покажем анализ совместимости электрических характеристик логических элементов на примере.

Пример 1. Необходимо проанализировать совместимость логических элементов двух серий DP1 и SB2. Входные и выходные параметры элементов приведены ниже:

DP1:

$$V_{IH} = 2{,}0\ \text{В},\ V_{IL} = 0{,}8\ \text{В}.\ I_{OH} = -0{,}4\ \text{мА},\ I_{OL} = 16\ \text{мА}$$
$$V_{OH} = 3{,}4\ \text{В},\ V_{OL} = 0{,}2\ \text{В},\ I_{IH} = 40\ \text{мкА},\ I_{IL} = -1{,}6\ \text{мА}$$

SB2:

$$V_{IH} = 2{,}0\ \text{В},\ V_{IL} = 0{,}8\ \text{В}.\ I_{OH} = -0{,}4\ \text{мА},\ I_{OL} = 8\ \text{мА}$$
$$V_{OH} = 2{,}7\ \text{В},\ V_{OL} = 0{,}4\ \text{В},\ I_{IH} = 20\ \text{мкА},\ I_{IL} = -0{,}4\ \text{мА}$$

Могут ли логические элементы серии SB2 быть подключены к элементам серии DP1? Если да, то в каком количестве? Иными словами, чему равен коэффициент разветвления DP1 для SB2?

Решение. Для ответа на вопрос необходимо сравнить уровни входных и выходных напряжений элементов этих серий, оценить нагрузочную способность элементов серии DP1 по отношению к элементам SB2. Коэффициент разветвления должен быть вычислен как для состояния логического 0, так и для состояния логической 1. Далее следует выбрать наихудший вариант сочетания параметров, и по нему необходимо дать заключение.

- Сравним напряжения при высоком уровне выходного сигнала. Минимальная величина выходного напряжения логической 1 для серии DP1 составляет V_{OH} = 3,4 В. Минимальный уровень входного напряжения логической 1 для серии SB2 – V_{IH} = 2,0 В. Сравнивая эти числа, можно сделать вывод, что по напряжению высокого логического уровня эти элементы совместимы. Если элемент серии DP1 сгенерирует на выходе наименьшее возможное напряжение V_{OH}, его величина обязательно превысит минимально необходимый уровень входного напряжения элемента серии SB2. В результате, ошибка распознавания единичного выходного уровня при исправных элементах невозможна.
- Далее сравним напряжения элементов при низком уровне выходного сигнала. Максимальное значение выходного напряжения логического 0 для элементов серии DP1 составляет V_{OL} = 0,2 В. Элементы серии SB2 распознают как уровень логического 0 сигналы с напряжением ниже V_{OL} = 0,8 В. Поскольку V_{OL} (DP1) < V_{IL} (SB2), ошибки распознавания низкого логического уровня также невозможна. Следовательно, элементы совместимы по уровням, подключение элемента серии SB2 к выходу элемента серии DP1 не вызовет нарушения передачи логического сигнала.
- Коэффициент разветвления в состоянии логической 1 равен:

$$K1 = I_{OH}(DP1)/I_{IH}(SB2) = 400мкА/20мкА = 20$$

- Коэффициент разветвления в состоянии логического 0 равен:

$$K0 = I_{OL}(DP1)/I_{IL}(SB2) = 16мА/0{,}4мА = 40$$

- Для заключительного вывода выбираем меньшее из двух полученных чисел. Коэффициент разветвления DP1?SB2 равен 20. Таким образом, к выходу элемента серии DP1 можно подключить 20 элементов серии SB2.

Пример 2. Определим, чему равен коэффициент разветвления элементов серии HC? То есть, сколько элементов серии HC можно подключить к выходу такого же элемента? Напомним входные и выходные параметры серии HC:

$$V_{IH} = 3{,}5\ В,\ V_{IL} = 1{,}0\ В.\ I_{OH} = -0{,}8\ мА,\ I_{OL} = 1{,}6\ мА$$
$$V_{OH} = 4{,}2\ В,\ V_{OL} = 0{,}4\ В,\ I_{IH} = 10\ мкА,\ I_{IL} = -10\ мкА$$

Решение. Ход рассуждений при решении этой задачи аналогичен предыдущей.

- Сравним напряжения при высоком уровне выходного сигнала. Минимальная величина выходного напряжения логической 1 серии HC составляет V_{OH} = 4,2 В. Минимальный уровень входного напряжения логической 1 этих же элементов – V_{IH} = 3,5 В. Сравнивая эти числа, можно сделать вывод, что по напряжению высокого логического уровня элементы серии HC выполне-

ны совместимыми, в чем и требовалось убедиться. Иного результата быть не может, иначе из элементов одной серии невозможно было бы создать схему.

- Максимальное значение выходного напряжения логического 0 для элементов серии HC составляет V_{OL} = 0,4 В. В то время как входное напряжение логического 0 сигналы для этих же элементов не должно быть ниже V_{OL} = 1,0 В. Поскольку $V_{OL} < V_{IL}$, ошибки распознавания низкого логического уровня быть не может. Следовательно, элементы серии HC совместимы сами с собой и по низкому логическому уровню также.
- Коэффициент разветвления в состоянии логической 1 равен:

$$K1 = I_{OH}/I_{IH} = 0{,}8 \text{ мА}/10 \text{ мкА} = 80$$

- Коэффициент разветвления в состоянии логического 0 равен:

$$K0 = I_{OL}/I_{IL} = 1{,}6 \text{ мА}/10 \text{ мкА} = 160$$

- Делаем вывод, что коэффициент разветвления серии элементов HC равен 80.

Итак, мы научились оценивать входные и выходные электрические характеристики логических элементов. На основе этих характеристик делать выводы о возможности подключения элементов различных серий друг к другу при составлении схемы какого-либо управляющего устройства. Далее мы рассмотрим различные интерфейсные компоненты, которые достаточно часто работают совместно с МК в микропроцессорных системах управления.

5.2. Устройства дискретного ввода: кнопки, переключатели, клавиатуры

Любая встраиваемая микропроцессорная система принимает данные из «внешнего мира», преобразует эти данные в управляющие воздействия, а затем «выдает» эти воздействия во внешний мир. Ни одна микропроцессорная система управления не обходится без механических переключателей, которые активируются пользователем, а также без различного рода устройств индикации, которые информируют пользователя о режимах работы самой системы и о состоянии объекта управления. В данном параграфе мы рассмотрим, как подключить к МК различные типы переключателей и простейшие светодиодные индикаторы. Несколько позже, в параграфе 5.6, мы подробно остановимся на вопросах сопряжения МК с жидкокристаллическим дисплеем (далее ЖК индикатор или ЖК дисплей).

5.2.1. Кнопки и переключатели

На рис. 5.4,а приведена схема подключения одиночного механического переключателя к МК. Механическая часть переключателя может быть выполнена таким образом, что его замкнутое состояние удерживается только тогда, когда человек нажимает на клавишу. Такой переключатель называют кнопкой. Другие переключатели обладают свойством удерживать замкнутое и разомкнутое состояние контактов при отсутствии внешнего воздействия. Последнее потребуется лишь для того, чтобы изменить состояние переключателя с разомкнутого на замкнутое и наоборот. Такие переключатели называются переключателями с фиксацией положения или тумблерами.

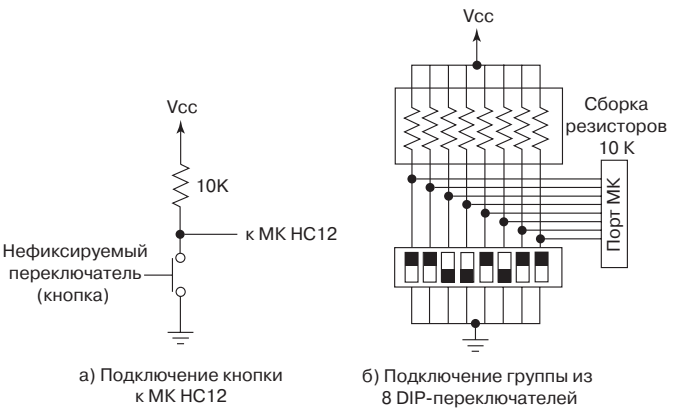


Рис. 5.4. *Примеры схем подключения механических переключателей к МК*

Возвратимся к нашей схеме (рис. 5.4,а). Когда клавиша отжата, переключатель находится в разомкнутом состоянии, и на входе МК формируется высокий логический уровень сигнала. Когда клавиша нажата, переключатель замыкает контакты, вывод МК подсоединяется к потенциалу общего провода системы GND, и на входе МК формируется логический 0. Резистор R = 10 кОм ограничивает силу тока в цепи переключателя.

Если бы переключатель был идеальным, то переход потенциала вывода МК из состояния 1 в состояние 0 при нажатии клавиши происходил бы мгновенно. На самом деле это не так. Переход механического переключателя из одного состояния в другое сопровождается механическим дребезгом контактов. Эффект механического дребезга состоит в том, что при смене состояния, например, с разомкнутого на замкнутое, контакты, прежде чем перейти из установившегося разомкнутого состояния в установившееся замкнутое, многократно замыкаются и размыкаются. Тогда, установив щуп осциллографа на вход МК, мы увидим сначала высокий уровень сигнала, затем многократное нерегулярное во времени переключение с 1 на 0, и, наконец, установится низкий уровень сигнала. Частота работы МК (от единиц до сотен МГц) чрезвычайно высока по сравнению с временами переключения механических контакторов (десятки - сотни мс). Поэтому механический дребезг контакта может быть воспринят управляющей программой как его многократное переключение. Существуют аппаратные и программные методы защиты от эффекта дребезга контактов. Один из программных методов заключается в том, что после обнаружения первого изменения логического уровня сигнала программа формирует задержку на 100-200 мс. В течение этого времени дребезг контактов прекращается, и переключатель переходит в новое устойчивое состояние. Мы рассмотрим аппаратные и программные методы противодребезгвой защиты в параграфе 5.5.

5.2.2. DIP переключатели

На рис. 5.4,б показана схема, в которой мы распространили идею подключения одного механического переключателя к МК сразу на восемь переключателей. Эти микропереключатели смонтированы на заводе-производителе в корпусе, который по размерам и расположению выводов совпадает с корпусом типа DIP (Dual In-line Package) для интегральных схем. Поэтому их называют DIP-переключатели.

Каждый переключатель из блока подсоединен к отдельному выводу порта МК. Так же, как и в предыдущем случае, ток цепи каждого переключателя ограничивается резистором. Для блока из восьми переключателей потребуется восемь резисторов. Для уменьшения габаритов печатной платы изделия целесообразно использовать сборку резисторов в одном корпусе. Однако возможно использование и одиночных резисторов.

В параграфе 5.8 мы рассмотрим пример использования подобной сборки DIP-переключателей для выбора режима работы микропроцессорной системы.

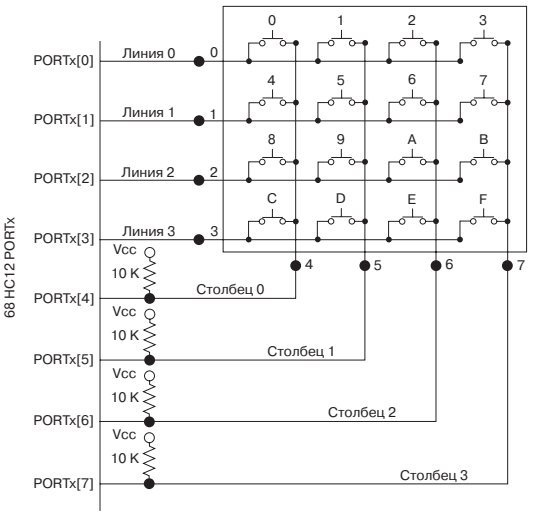
5.2.3. Клавиатуры

Во многих приложениях микропроцессорные системы требуют ввода цифровой и буквенной информации. Для таких случаев могут быть использованы блоки из нескольких кнопок, которые объединены конструктивно и соединены электрически по стандартной матричной схеме. Такие блоки называют клавиатурами. На рис. 5.5 показана клавиатура из 16 клавиш, которая позволяет вводить данные в микропроцессорную систему в шестнадцатеричном коде. Изучим представленную на рис. 5.5 схему соединения МК с клавиатурой подробно.

Кнопки клавиатуры собраны в матричную схему. Первый ряд составляют кнопки 0, 1, 2 и 3. Второй ряд – кнопки 4. 5. 6. и 7. Следующие восемь кнопок составляют ряд 3 и ряд 4. Выводы левых контактов кнопок одного ряда соединены между собой и подключены к одному из выводов порта PORTx (x - имя порта МК, например, PORTA, PORTB и т.д.). В нашем примере для обслуживания четырех линий клавиатуры использованы четыре линии порта PORTx[0].....PORTx[3]. Все эти линии работают в режиме вывода.

Выводы правых контактов кнопок также соединены между собой. Но при этом объединены каждые четыре кнопки по вертикали, которые зрительно составляют единый столбец. Приведенное соединение называют матричным. Из рис. 5.5. можно видеть, что блок клавиатуры из 16 клавиш имеет всего восемь линий связи для подключения к МК. А если бы мы использовали ранее рассмотренные схемы соединения, то для подключения 16 кнопок портебовалось бы 16 линий связи. Четыре вывода столбцов подключаются ко входам порта PORTx[4].....PORTx[7]. Эти линии порта работают в режиме ввода. Каждая линии PORTx[4].....PORTx[7] снабжена подтягивающим к напряжению питания резистором R = 10 кОм.

Если все клавиши, принадлежащие к одной колонке, отжаты, то на соответствующем входе порта PORTx[4].....PORTx[7] формируется логическая 1. МК использует специальную программу опроса клавиатуры, чтобы обнаружить, какая клавиша в данный момент времени нажата. Эта программа сначала выставляет на линию PORTx[0] логический 0. Тогда все левые выводы клавиш первого ряда оказываются под потенциалом логического 0. Если одну из клавиш этого ряда нажать, то уровень сигнала на одном из входов PORTx[4].....PORTx[7] изменится с 0 на 1. Например, если нажали клавишу «0», то на входе PORTx[4] установится низкий логический уровень, а все остальные входы PORTx[5].....PORTx[7] останутся в 1. Или, если нажать клавишу «2», то в 0 установится вывод PORTx[6], а линии PORTx[4], PORTx[5] и PORTx[7] будут в 1. Таки образом, программа, прочитав состояние линий PORTx[4].....PORTx[7] и обнаружив 0 в каком-либо разряде, сможет установить, какая клавиша нажата.



	Код на выходах PORTx[3:0]				Код на входах PORTx[7:4]				
Нажатая клавиша	3	2	1	0	7	6	5	4	Код порта PORTx[7:0]
0	1	1	1	0	1	1	1	0	0xEE
1	1	1	1	0	1	1	0	1	0xDE
2	1	1	1	0	1	0	1	1	0xBE
3	1	1	1	0	0	1	1	1	0x7E
4	1	1	0	1	1	1	1	0	0xED
5	1	1	0	1	1	1	0	1	0xDD
6	1	1	0	1	1	0	1	1	0xBD
7	1	1	0	1	0	1	1	1	0x7D
8	1	0	1	1	1	1	1	0	0xEB
9	1	0	1	1	1	1	0	1	0xDB
A	1	0	1	1	1	0	1	1	0xBB
B	1	0	1	1	0	1	1	1	0x7B
C	0	1	1	1	1	1	1	0	0xE7
D	0	1	1	1	1	1	0	1	0xD7
E	0	1	1	1	1	0	1	1	0xB7E
F	0	1	1	1	0	1	1	1	0x77
Не нажата	X	X	X	X	1	1	1	1	0xXF

Рис. 5.5. *Схема подключения матричной клавиатуры к МК 68HC12*

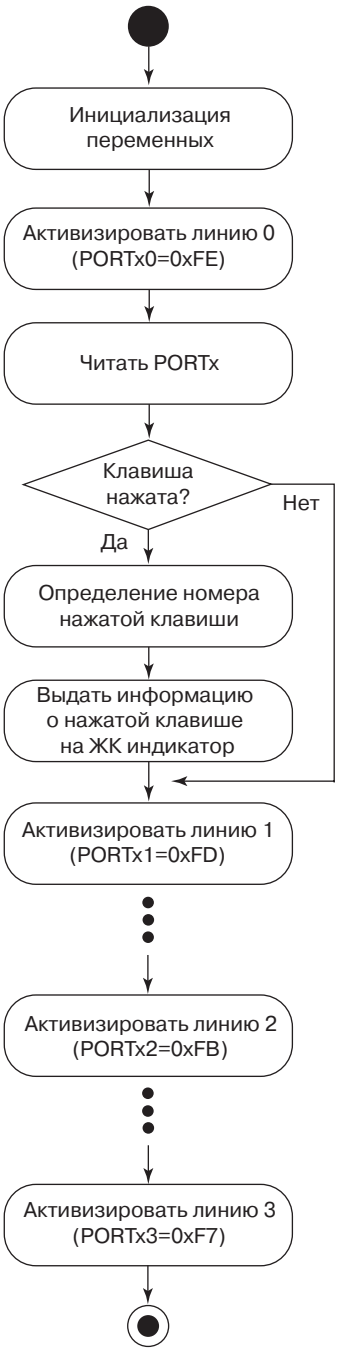


Рис. 5.6. *Блок-схема алгоритма опроса матричной клавиатуры*

Программа опроса клавиатуры последовательно анализирует состояние клавиш каждого ряда, последовательно выставляя логический 0 на выводы PORTx[0].....PORTx[3]. На рис. 5.5 приведена таблица, в которой показаны коды, которые будут на линиях порта при нажатии каждой из клавиш. Эти коды должны быть использованы программой опроса для приведения кода нажатой клавиши к одному из стандартных представлений, например к коду ACSII.

Рассмотренная нами клавиатура может иметь иные символы на кнопках, в соответствие с функциональным назначением кнопки в устройстве. Например, подобная клавиатура используется для управления насосом бензоколонки. Тогда клавиши могут задавать тип отпускаемого бензина, форму приема платежа и т.д. Микроконтроллер должен распознать, какая клавиша нажата, и перейти к соответствующей подпрограмме.

На рис. 5.6 приведена блок-схема алгоритма опроса матричной клавиатуры из 16 клавиш. Этот алгоритм состоит из четырех одинаковых блоков, в которых сначала устанавливается в 0 одна из линий PORTx[0].....PORTx[3], а затем контролируется состояние линий PORTx[4].....PORTx[7]. Если на входах линий PORTx[4].....PORTx[7] логические 1, то ни одна кнопка активизированного ряда не нажата, и следует перейти к опросу следующего ряда. Если на какой -либо из линий PORTx[4].....PORTx[7] обнаружен потенциал логического 0, то по номеру линии и по номеру активизированного ряда программа должна восстановить код нажатой клавиши. Для этого удобно использовать таблицу рис. 5.5. Мы привели лишь общую структуру алгоритма. Детальное ее рассмотрение последует после примера программного кода, и описания ЖК дисплея.

```
/*--------------------------------------------------------------------*/
/* filename: keypad.c                                                 */
/* MAIN PROGRAM: Эта программа производит анализ                      */
/*состояния матричной клавиатуры из 16 клавиш                         */
/*Для подключения клавиатуры использован PORTB                        */
/*Разряды PORTB[0]…PORTB[3] активизируют линии рядов                  */
/*разряды PORTB[4]…PORTB[7]используются для считывания кодов          */
/*колонок                                                             */
/*--------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>
#include<stdio.h>
#include<math.h>

/*используемые функции*/
char keypad(unsigned char keypress, unsigned char row);
char which_key(unsigned char keypress);       /*Function to select key*/

void main{void)
{
unsigned char keypress;

/*задание кодов активизации рядов клавиатуры*/
unsigned char first = 0xFE;      /*код активизации первого ряда*/
unsigned char second = 0xFD;     /*код активизации второго ряда*/
unsigned char third = 0xFB;      /*код активизации третьего ряда*/
unsigned char fourth = 0xF7;     /*код активизации четвертого ряда*/
```

```
DDRB = 0x0f;                /*линии 0…3 на вывод, линии 4…7 на ввод*/

while (1)
        {
        PORTB = 0xFE;
        keypress = PORTB;
        key = keypad(keypress, first);
        PORTB = 0xFD;
        keypress = PORTB;
        key = keypad(keypress, second);
        PORTB = 0xFB;
        keypress = PORTB;
        key = keypad(keypress, third);
        PORTB = 0xF7;
        keypress = PORTB;
        key = keypad(keypress, fourth);
        }
}

/*-------------------------------------------------------------------*/
/*Функция keypad определяет, была ли нажата какая-либо клавиша.     */
/*Функциясравнивает прочитанное значение порта PORTB с тем значением,  */
/*которое было выдано на порт PORTB. Если они равны, то ни одна клавиша в */
/*ряду не нажата                                                    */
/*-------------------------------------------------------------------*/
char keypad(unsigned char keypress, unsigned char row)
{
char key1;

if(keypress != row)
{                                /*какая-то клавиша нажата*/
key1 = which_key(keypress);      /*определить клавишу*/
putchars (key1);                 /*передать символ на дисплей*/
}
else if(keypress = = row)
{                                /*ни одна из клавиш не нажата*/
key1 = 'Z';
}
return (key1);
}
/*-------------------------------------------------------------------*/
/* Функция char witch_key определяет код нажатой клавиши методом перебора */
/*табличных значений                                                 */
/*-------------------------------------------------------------------*/
char witch_key(unsigned char keypress)
{
char key;

switch(keypress)                 /*распознавание кода клавиши*/
{
case 0xEE: key = '0' ;           /*нажата клавиша "0"*/
break;

case 0xDE: key = '1' ;           /*нажата клавиша "1"*/
break;
```

```
case 0xBE: key = '2' ;     /*нажата клавиша "2"*/
break;

case 0x7E: key = '3' ;     /*нажата клавиша "3"*/
break;

case 0xED: key = '4' ;     /*нажата клавиша "4"*/
break;

case 0xDD: key = '5' ;     /*нажата клавиша "5"*/
break;

case 0xBD: key = '6' ;     /*нажата клавиша "6"*/
break;

case 0x7D: key = '7' ;     /*нажата клавиша "7"*/
break;

case 0xEB: key = '8' ;     /*нажата клавиша "8"*/
break;

case 0xDB: key = '9' ;     /*нажата клавиша "9"*/
break;

case 0xBB: key = 'A' ;     /*нажата клавиша "A"*/
break;

case 0x7B: key = 'B' ;     /*нажата клавиша "B"*/
break;

case 0xE7: key = 'C' ;     /*нажата клавиша "C"*/
break;

case 0xD7: key = 'D' ;     /*нажата клавиша "D"*/
break;

case 0xB7: key = 'E' ;     /*нажата клавиша "E"*/
break;

case 0x77: key = 'F' ;     /*нажата клавиша "F"*/
break;

default: key = 'Z';
}                          /*конец распознавание кода клавиши*/

return (key);
}
/*-------------------------------------------------------------------*/
```

Представленный программный код не является полностью завершенной рабочей программой, поскольку в нем отсутствуют элементы защиты от дребезга контактов, отсутствует текст функции вывода символов на ЖК дисплей. Мы исправим эти недостатки несколько позже, в разделе 5.6.

5.3. Устройства индикации: светодиоды, семисегментные индикаторы, индикаторы логического выхода с тремя состояниями

В этом параграфе мы научимся подключать различные типы светодиодных индикаторов к выходам микроконтроллера. Основываясь на полученных ранее сведениях об электрических характеристиках светодиода видимого спектра излучения, мы рассмотрим, как подключить к МК линейку из восьми светодиодов и семисегментный индикатор. Далее мы обсудим, проблемы сопряжения с семисегментным индикатором больших размеров. Завершим параграф рассмотрением оригинальной схемы индикации состояния типичных для микропроцессорной техники выходных логических буферов с тремя состояниями.

5.3.1. Светодиоды

В процессе наладки любой микропроцессорной системы крайне удобно использовать светодиоды для индикации состояния тех или иных логических выходов. Светодиод имеет два вывода: анод (+) и катод (–). Для того, чтобы светодиод излучал, напряжение между анодом и катодом должно быть положительным. Светодиоды характеризуются двумя основными параметрами: рабочим током и напряжением прямого смещения. Типичные значения рабочих токов светодиодов лежат в диапазоне от 10 до 15 мА, при этом прямое падение напряжения составляет 1,5 В.

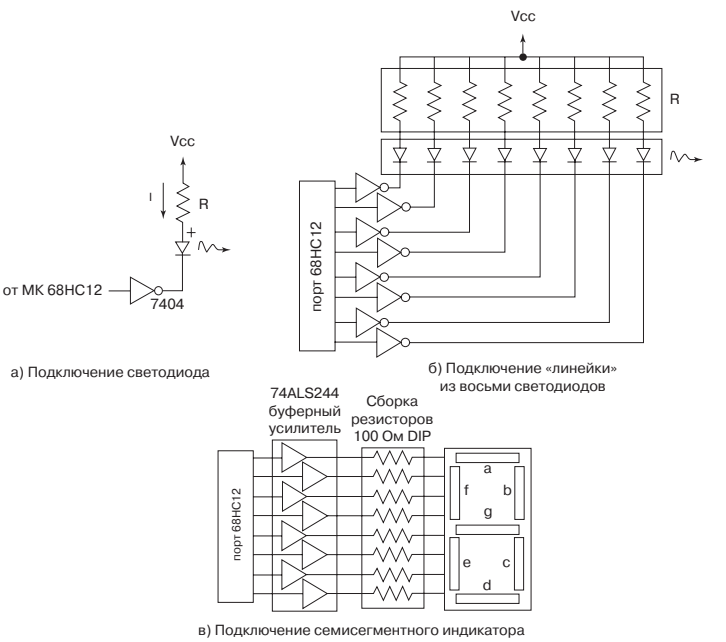


Рис. 5.7. *Примеры схем подключения светодиодных индикаторов к МК*

На рис. 5.7,а показана схема подключения светодиода к выводу МК. В параграфе 5.1.2 мы убедились в невозможности подключения светодиода непосредственно к выводу МК. Выходные буферы МК не способны обеспечить величины тока светодиода, достаточной для свечения с приемлемой для человеческого глаза яркостью. Поэтому в приведенной схеме использован буферный элемент 7404. При выборе буферного элемента следует убедиться, что максимальное значение выходного тока нуля I_{OL} этого элемента не ниже номинального значения прямого тока светодиода 10...15 мА. Так для выбранной модели элемента 7404 максимальное значение выходного тока логического нуля составляет 16 мА.

В схеме рис. 5.7,а логический элемент должен обязательно быть с инверсией на выходе. Тогда если выход МК в состоянии логической 1, выход элемента в 0, по цепи светодиода протекает ток, и светодиод светится. Наоборот, если на выходе МК логический 0, выход логического элемента в 1, и светодиод погашен. Для правильной работы схемы напряжение V_{CC} должно обязательно превышать напряжение прямого смещения светодиода 1,5 В. Для выбранного значения V_{CC} подбирается резистор R, величина которого ограничивает ток светодиода. Выбирая величину V_{CC} отличной от напряжения питания МК, необходимо помнить, что, если в цепи светодиода ток отсутствует, то напряжение V_{CC} прикладывается к выходному буферу логического элемента. Для обычных логических элементов это напряжение не превышает 5,0 В, а для элементов с повышенным коллекторным напряжением – 15 В.

Пример. Определите величину сопротивления резистора R на рис. 5.7,а. Предположите, что прямой ток светодиода должен быть равным 15 мА.

Решение. Падение напряжения на резисторе R в цепи рис. 5.7,а составляет:

$$V_R = V_{CC} - VD - V_{OL} = 5{,}0 - 1{,}5 - 0{,}4 = 3{,}1 \text{ В}$$

Тогда величина сопротивления резистора определяется неравенством:

$$R \geqslant V_R / I_D = 3{,}1 \text{ В}/15 \text{ мА} = 206 \text{ Ом}.$$

Выбираем ближайший к расчетному значению номинал резистора 220 Ом.

Очень часто разработчик желает индицировать состояние всех линий какого-либо порта. Для этой цели удобно использовать так называемые линейки светодиодов, которые подобно DIP переключателям сгруппированы по 8 и смонтированы в корпусе, который размещается на посадочное место для DIP. Схема подключения такой линейки светодиодов показана на рис. 5.7,б. С точки зрения конструктивного исполнения для такой схемы целесообразно использовать резисторную сборку, что и отражено на рис. 5.7,б.

5.3.2. Семисегментные индикаторы

Многие микропроцессорные устройства требуют вывода информации в символах десятичной или шестнадцатеричной системы счисления. Наиболее подходящим индикатором для этого является светодиодный семисегментный индикатор. Внешний вид семисегментного индикатора показан на рис. 5.7,в. Каждый сегмент индикатора – это светодиод, вмонтированный в пластмассовый корпус с поверхностью рассеивания света в виде того или иного сегмента. Сегменты именуются стандартным способом, латинскими буквами от a до g. Комбинация светящихся сегментов образует цифру, или букву. Например, чтобы высветить цифру 1, необходимо зажечь сегменты b и c, цифру 2 – сегменты a, b, d, e, g, букву F – сегменты a, e, f, g. Подключение семисегментного индикатора к

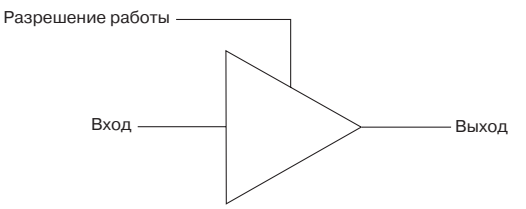
выходам порта МК показано на рис. 5.7,в. В схеме для усиления по току использованы не отдельные логические элементы, а специальная буферная ИС 74ALS244. Ток в цепи светодиодов каждого сегмента ограничивают резисторы сопротивлением 100 Ом. В этом примере использован семисегментный индикатор с общим анодом. В соответствии с названием, аноды всех сегментов индикатора объединены внутри корпуса и выведены на одну ножку. Выпускаются также индикаторы с общим катодом.

5.3.3. Индикаторы для логического выхода с тремя состояниями

В этом параграфе мы рассмотрим схему отображения для восьми линий порта с тремя состояниями. Сначала напомним читателю, как работают выходные логические буферы с тремя состояниями.

Выходной логический буфер с тремя состояниями может формировать на выходе напряжение высокого уровня, которое соответствует логической 1, напряжение низкого уровня, которое соответствует логическому 0, и может находиться в так называемом третьем состоянии или иначе «Z» состоянии, которое характеризуется очень высоким выходным сопротивлением (рис. 5.8.). Это сопротивление столь велико, что можно считать, что внутренние цепи логического элемента отсоединены от соответствующего вывода корпуса этого элемента. Это свойство буферов с тремя состояниями позволяет подсоединять к одному электрическому проводнику печатной платы сразу несколько выводов ИС.

Такое решение используется при подключении к магистралям микропроцессорной системы различных ИС внешней памяти. В каждый момент времени МК ведет обмен только с одной ИС памяти. Он активизирует работу этой ИС установкой в 0 сигнала выбора кристалла $\overline{CS}$. Все остальные ИС памяти в этот момент времени находятся в неактивном состоянии, буферы их линий данных пребывают в «Z» состоянии с высоким выходным сопротивлением. Таким образом, эти ИС в данный момент времени как бы отсоединены от линий магистрали данных и не мешают обмену данными МК с единственной активизированной ИС памяти. В другой момент времени положение дел изменится. В активное состояние будет переведена другая схема памяти, в то время как выходные буферы первой ИС будут переведены в «Z» состояние, и эта ИС как бы выключится из работы. Но при этом не следует забывать, что все неактивизированные ИС памяти находятся под питанием и надежно хранят ранее записанную в них информацию.



Вход разрешения работы	Вход	Выход
L	L	Z
L	H	Z
H	L	L
H	H	H

Рис. 5.8. *Логические элементы с тремя состояниями на выходе*

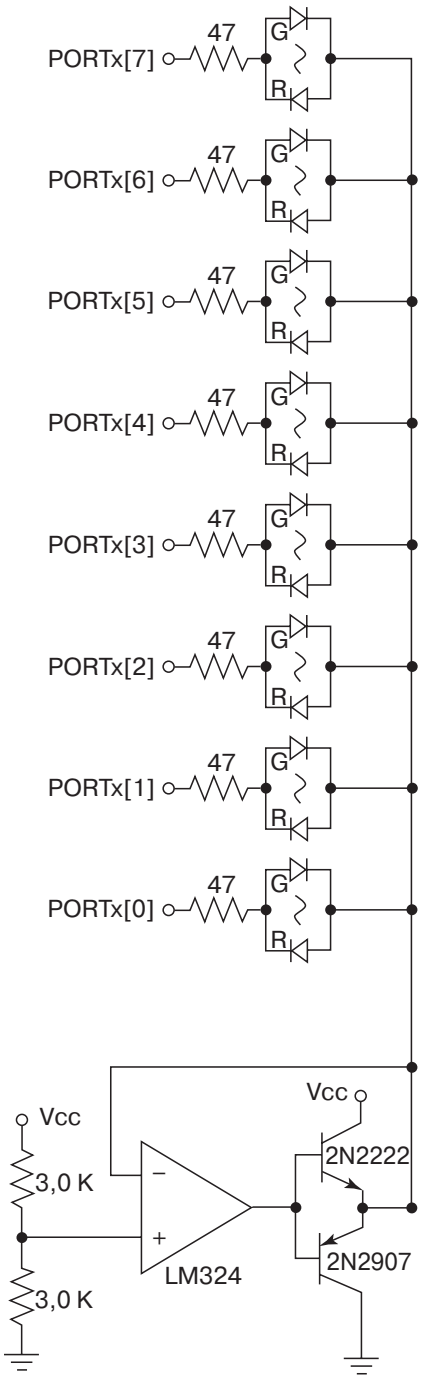


Рис. 5.9. *Функциональная схема 8 индикаторов для выходов с тремя состояниями*

Вернемся к рассмотрению схемы индикации. Она представлена на рис. 5.9. Индикатор каждого разряда состоит из двух светодиодов: зеленого и красного. Если на выходе порта PORTx[i] формируется высокий логический уровень сигнала, то благодаря усилителю с большим коэффициентом усиления, который выполнен на операционном усилителе LM324, открывается транзистор 2N2907. Для зеленого светодиода создается прямое смещение напряжения и путь для протекания тока. В результате, зеленый светодиод светится и информирует пользователя о наличии высокого логического уровня на соответствующем выходе. Если на выходе порта PORTx[i] формируется низкий логический уровень сигнала, то открывается транзистор 2N2222, и светится красный светодиод. Если же выход порта PORTx[i] установлен в третье состояние, то пути для протекания токов светодиодов нет, они оба погашены.

5.4. Программное обслуживание дискретных входов и выходов

Для написания программного кода предположим, что входы порта PORTA подключены к блоку DIP переключателей, а выходы порта PORTB - к 8-разрядному светодиодному индикатору по схеме рис. 5.9. Необходимо считать байт данных с порта PORTA и вывести этот байт данных на порт PORTB.

```
.
.
unsigned char INMASK = 0x00;
unsigned char OUTMASK = 0xff;
unsigned char PORTA_value;

DDRA = INMASK;              /*порт PORTA на ввод*/
DDRB = OUTMASK;             /*порт PORTB на вывод*/
PORTA_value = PORTA;        /*считать портPORTA*/
PORTB = PORTA_value;        /*записать в порт PORTB*/
.
.
```

5.5. Подавление механического дребезга контактов переключателей

Ранее, в параграфе 5.2, Вы познакомились с различными типами механических переключателей. Переключатели обладают двумя устойчивыми состояниями, в которых замыкаются или размыкаются электрические контакты. Включенный по схеме рис. 5.4,а переключатель обеспечивает высокий логический уровень сигнала на входе МК, если пара его контактов разомкнута. В нажатом состоянии контакты замыкаются, и на входе МК формируется напряжение низкого логического уровня.

Идеальный переключатель изменяет состояние контактов с разомкнутого на замкнутое и наоборот моментально, так, что на входе МК переключение между двумя уровнями логических сигналов происходит за время, которое стремится к нулю.

Однако реальные переключатели ведут себя иначе. Так при нажатии клавиши рис. 5.4,а контакты многократно замыкаются и размыкаются, пока не окажутся в установившемся замкнутом состоянии. Это явление называется механическим дребезгом переключателя. Микроконтроллеры работают на частотах в несколько МГц, что остаточно для того, чтобы зафиксировать многократное изменение состояния переключателя во время дребезга. Однако реакция программы управления на каждое переключение во время дребезга было бы ошибкой. Поэтому следует применять специальные методы подавления эффекта механического дребезга. Различают аппаратные и программные способы подавления дребезга контактов. Мы рассмотрим несколько вариантов для каждого из этих способов.

5.5.1. Аппаратная защита от механического дребезга контактов

На рис. 5.10 представлена схема, которая осуществляет защиту от дребезга механических переключателей [Horowitz и Hill, 1989]. Основным элементом этой схемы является триггер Шмитта (74HC14).

Триггер Шмита отличается от других логических элементов, во-первых, тем, что на его входы можно подавать аналоговые сигналы. И если для обычного логического элемента время изменения входного сигнала из состояния 0 в состояние 1 должно составлять всего несколько наносекунд, то для триггера Шмитта это время может быть любым, в том числе несколько десятков или сотен миллисекунд, которые потребуются нам для устранения дребезга контактов. Во-вторых, передаточная характеристика триггера Шмитта обладает гистерезисом: уровень входного напряжения, при котором выход триггера переключается из 0 в 1, превышает уровень напряжения переключения из 1 в 0 примерно на 0,5...0,8 В. Эффект гистерезиса позволяет подавить звон входного сигнала. Незначительные по амплитуде высокочастотные колебания, наложенные на монотонно изменяющуюся постоянную составляющую входного сигнала, не будут приводить к многократному изменению выходного сигнала, поскольку абсолютная величина входного сигнала окажется внутри петли гистерезиса.

Рассмотрим работу схемы подавления дребезга контактов переключателя. При разомкнутых контактах напряжение конденсатора равно V_{CC}, на выходе триггера Шмита формируется низкий логический уровень, поскольку все триггеры Шмитта в интегральном исполнении инвертируют входной сигнал. Если клавишу только что нажали, то конденсатор начинает разряжаться через резистор и замкнутые контакты переключателя. Время его разряда определяется постоянной времени RC. Для приведенных на рис. 5.10,а номиналов постоянная времени равна 47 мс. Напряжение на конденсаторе будет убывать немонотонно, поскольку на интервале дребезга контакты то замкнутся, то разомкнутся. В соответствии с их положением конденсатор то разряжается, то заряжается. Но большая постоянная времени цепей разряда и заряда не позволяет напряжению на конденсаторе измениться во время дребезга столь сильно, чтобы произошло переключение триггера Шмитта. И лишь когда дребезг закончится и произойдет длительное замыкание контактов переключателя, конденсатор разрядится до нуля, и на выходе триггера Шмитта установится высокий логический уровень. При размыкании контактов переключателя процесс будет происходить в обратном порядке. Эффект подавления дребезга в рассмотренной схеме будет наблюдаться только тогда, когда постоянная времени цепей разряда и заряда конденсатора

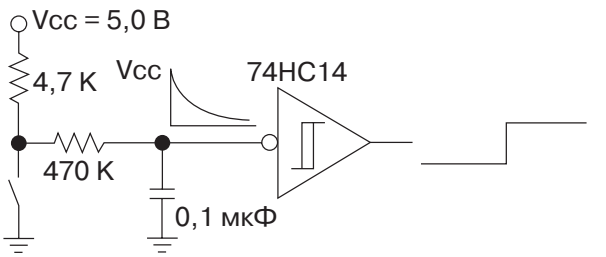
будет сравнима с длительностью самого эффекта механического дребезга. Для маломощных переключателей его длительность измеряется единицами и десятками мс, поэтому в нашем примере номиналы резисторов и конденсатора выбраны правильно.

Если Вы желаете иметь на выходе цепи подавления дребезга инверсную логику, то следует использовать дополнительный инвертор (рис. 5.10,б). Тогда при разомкнутом переключателе на выходе инвертора будет низкий логический уровень, а при замкнутом – высокий.

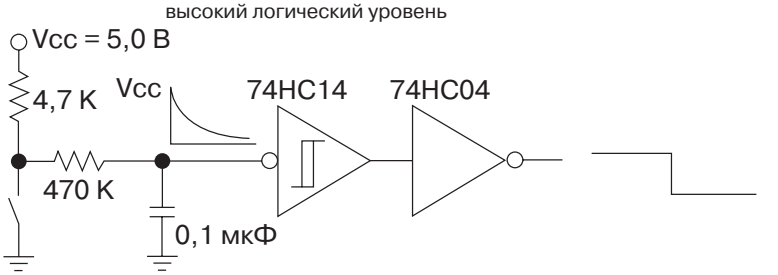
5.5.2. Программная защита от механического дребезга контактов

Существует несколько методов программной защиты от эффекта дребезга механических контактов. Мы кратко остановимся на двух наиболее часто используемых:

- Прикладная программа проводит мониторинг состояния контакта, опрашивая линию порта и контролируя уровень логического сигнала. Как только программа зафиксировала изменение сигнала, она фиксирует это изменение и формирует задержку 100...200 мс. За это время дребезг закончится, и контакт зафиксируется в установившемся состоянии. Если это новое состояние совпадет с первым зафиксированным изменением, то произошло переключение контакта. В противном случае имела место помеха.
- Прикладная программа также производит мониторинг контакта. После обнаружения первого переключения программа продолжает постоянно опрашивать контакт в течение тех же 100...200 мс. Если к концу интервала опроса состояние не стабилизировалось и постоянно считывается то 0 то 1, то это помехи, а если установилось в одном из состояний, то произошло переключение.



а) При нажатии клавиши на выходе формируется высокий логический уровень



б) При нажатии клавиши на выходе формируется низкий логический уровень

Рис. 5.10. *Примеры схем подавления механического дребезга контактов*

5.5.3. Пример программной защиты

Для того чтобы продемонстрировать пример реального программного кода для ввода и вывода дискретных управляющих воздействий, предположим, что восемь DIP переключателей подключены к порту PORTB микроконтроллера, а состояние выходов порта PORTC индицируются с использованием схемы двухцветного индикатора. Общая структура микропроцессорной системы нашего примера представлена на рис. 5.11. В примере мы используем аппаратную защиту и программную защиту от дребезга контактов. Программная защита реализуется по второму способу.

```
/*-------------------------------------------------------------------*/
/*filename: debounced_swith.c                                        */
/*осуществляет опрос 8 переключателей с программной защитой от дребезга  */
/*выполняет специальные действия для каждого нажатого переключателя */
/*зажигает зеленый светодиод в разряде активизированного переключателя*/
/*-------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>

/*используемые функции*/
int process_valid_input(unsigned char input_value); //управление
                                //процессом опроса в реальном времени
void initialize_ports(void);    //инициализация портов
void timer_init(void);          //инициализация таймера

//глобальные переменные
int keep_going=1;               //переменная цикла
unsigned char old_PORTB= 0xff;  //предыдущее значение порта PORTB
unsigned char new_PORTB;        //новое значение порта PORTB

void main{void)
{
initialize_ports();
timer_init();

while(keep_going)
       {
       new_PORTB=PORTB;         //читать порт PORTB
       if (new_PORTB != old_PORTB)
              {           //выполнять, если значение порта изменилось
              swith(new_PORTB)
                     {
                     case 0xFE:   //переключатель PB0
                     if(process_valid_input(new_PORTB))//процедура
                                                  //антидребезга
                     {                  //выполнять действия, связанные
                     :                  //с нажатием клавиши PB0
                     :
                     PORTC = 0x01;//зажечь зеленый в разряде PB0
                     keep_going=1;
                     }
                     break;
```

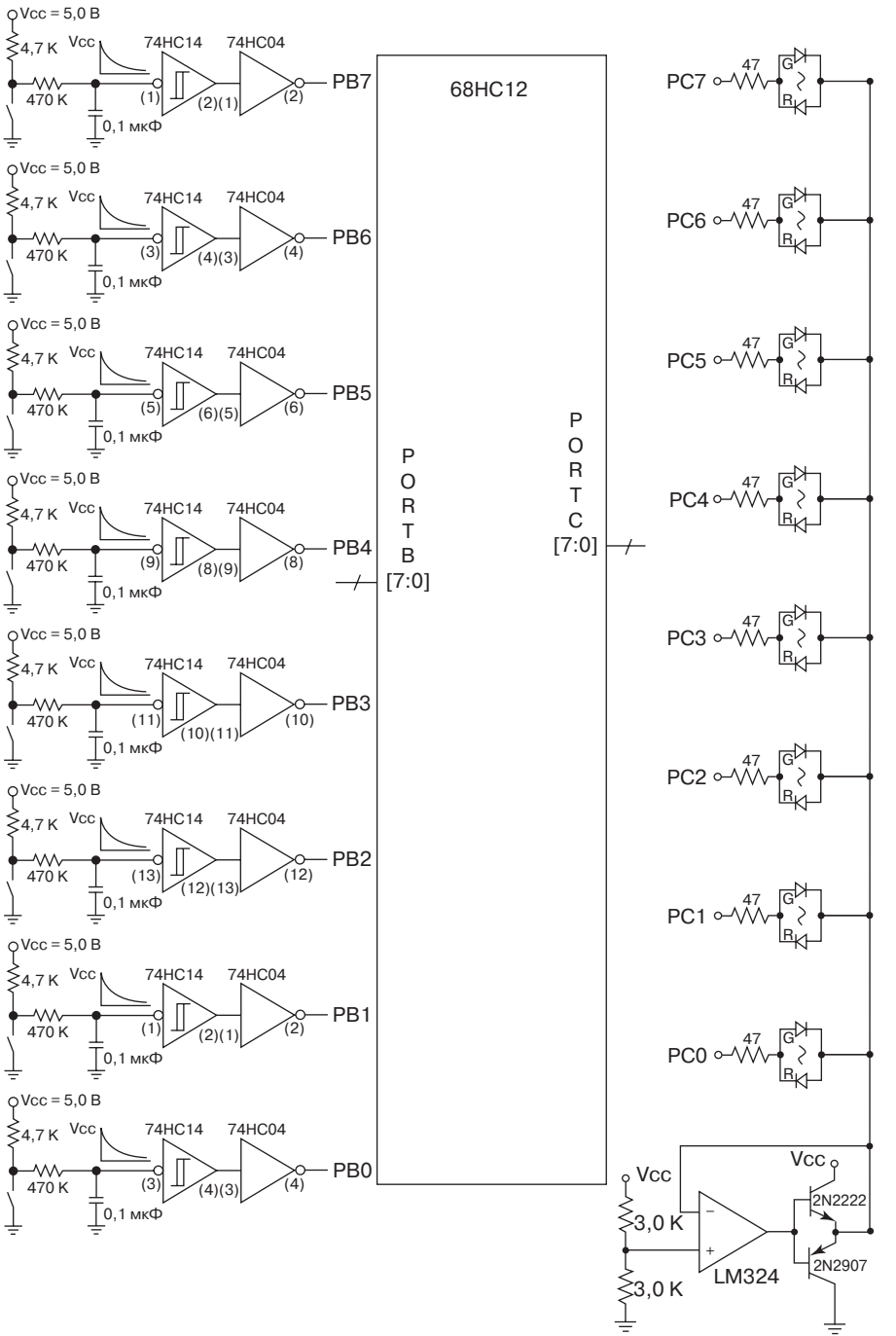


Рис. 5.11. *Функциональная схема аппаратных средств для примера 5.3.3*

```
                    case 0xFD:          // переключатель PB01
                    if(process_valid_input(new_PORTB)) // процедура
                                        // антидребезга
                    {                   //выполнять действия, связанные
                    :                   //с нажатием клавиши PB1
                    PORTC = 0x02;       //зажечь зеленый в разряде PB1
                    keep_going=1;
                    }
                    break;

                    case 0x7F:          // переключатель PB7
                    if(process_valid_input(new_PORTB)) // процедура
                                        //антидребезга
                    {                   //выполнять действия, связанные
                    :                   //с нажатием клавиши PB7
                    PORTC = 0x80;       //зажечь зеленый в разряде PB7
                    keep_going=1;
                    }
                    break;
                    default:;           //all other cases
} //окончание распознавания, какая клавиша нажата (switch(new_PORTB))
} //окончание обработки нажатой клавиши (if new_PORTB)

old_PORTB=new_PORTB;             //обновить PORTB
}      //окончание (while(keep_going))
}      //окончание (main)

/*--------------------------------------------------------------------*/
/* Функция void initialize_ports устанавливает режим работы портов  */
/*--------------------------------------------------------------------*/
void initialize_ports(void)
{
DDRC=0xff;          //порт PORTC на вывод
PORTC=0x00;         //выходы порта PORTC в 0, зажгутся красные светодиоды
DDRB=0x00;          //порт PORTB на ввод
}

/*--------------------------------------------------------------------*/
/* process_valid_input: проверка стабильности PORTB в течение периода */
/*переполнения таймера                                                */
/*--------------------------------------------------------------------*/
int process_valid_input(unsigned char portx)
{
int valid_input;           //флаг результата опроса порта
int int_value;             //время входа в процедуру

valid_input = TRUE;        //установить флаг
int_value = TCNT;          //читать текущий момент времени

while(int_value != TCNT)   //повторять пока значения времени снова не
                           //сравняются
{
if(portx= =PORTB)          //порт PORTB остается стабильным
valid_input = TRUE;
else                       //порт PORTB изменил состояние
```

```
valid_input = FALSE;
if (!valid_input)          //если порт изменил состояние, то выйти
                           //из цикла
break;
}
return valid_input;
}
/*---------------------------------------------------------------------*/
/* Функция timer_init инициализирует таймер. Частота системной шины равна */
/*8*МГц                                                                */
/*---------------------------------------------------------------------*/
void timer_init(void)
{
TMSK1 = 0x00;              //запретить прерывания
TMSK2 = 0x02;              //частота тактировании 2 МГц
TSCR = 0x80;               //разрешить работу модуля таймера
}
/*---------------------------------------------------------------------*/
```

5.6. Жидкокристаллические индикаторы

В этом разделе мы подробно рассмотрим, как подключить жидкокристаллический индикатор к МК 68HC12. Мы начнем с краткого обзора принципа действия ЖК индикаторов. Далее изучим реальный однострочный символьный индикатор на 16 знакомест со встроенным контроллером управления. Рассмотрим, как выполнить аппаратное подключение этого индикатора к МК, а затем приведем пример программы управления.

5.6.1. Краткие сведения о жидкокристаллических индикаторах

Жидкокристаллический индикатор – почти идеальное устройство отображения информации. Этот тип индикаторов использует для своей работы те же напряжения, что и микроконтроллеры, но при этом потребляет энергию на несколько порядков меньшую, чем светодиодные индикаторы. Именно поэтому жидкокристаллические (далее ЖК) индикаторы нашли чрезвычайно широкое применение в переносных устройствах с автономным питанием. В электронных часах, калькуляторах и стационарных телефонах принято использовать монохромные ЖК индикаторы, в то время как современные мобильные телефоны, фотоаппараты и видеокамеры немыслимы без малогабаритного цветного ЖК-дисплея. По способу отображения информации ЖК индикаторы также подразделяются на цифро-буквенные и графические.

Для понимания технологических особенностей создания современных ЖК индикаторов и дисплеев следует коротко остановиться на основных свойствах жидких кристаллов. Жидкие кристаллы представляют собой почти прозрачные субстанции, проявляющие одновременно свойства кристалла и жидкости. Есть две главные особенности жидких кристаллов, благодаря которым возможно создание на их основе устройств отображения информации: способность молекул жидких кристаллов переориентироваться во внешнем электрическом поле и изменять поляризацию светового потока, проходящего через их слои.

Основой ЖК индикатора являются две параллельные стеклянные пластины с на-

несенными на них поляризационными пленками. Различают верхний и нижний поляризаторы, сориентированные перпендикулярно друг другу. На стеклянные пластины в тех местах, где в дальнейшем будет формироваться изображение, наносится прозрачная металлическая окисная пленка, которая в дальнейшем служит электродами. На внутреннюю поверхность стекол и электроды наносятся полимерные выравнивающие слои, которые затем полируются, что способствует появлению на их поверхности, соприкасающейся с жидкими кристаллами, микроскопических продольных канавок. Пространство между выравнивающими слоями заполняют жидкокристаллическим веществом. В результате молекулы жидких кристаллов выстраиваются в направлении полировки полимерного слоя. Направления полировки верхнего и нижнего слоев полимера перпендикулярны (подобно ориентации поляризаторов). Это нужно для предварительного "скручивания" слоев молекул жидких кристаллов между стеклами на 90°. Когда напряжение на управляющие электроды не подано, поток света, пройдя через нижний поляризатор, двигается через слои жидких кристаллов, которые плавно меняют его поляризацию, поворачивая её на угол 90°. В результате поток света после выхода из ЖК материала беспрепятственно проходит через верхний поляризатор (сориентированный перпендикулярно нижнему) и попадает к наблюдателю. Никакого формирования изображения не происходит. При подаче напряжения на электроды между ними создается электрическое поле, что вызывает переориентацию молекул жидких кристаллов. Молекулы стремятся выстроиться вдоль силовых линий поля в направлении от одного электрода к другому. Вследствие этого пропадает эффект «скручивания» поляризованного света, под электродом возникает область тени, повторяющая его контуры. Создается изображение, формируемое светлой фоновой областью и темной областью под включенным электродом. Путем варьирования контуров площади, занимаемой электродом, можно формировать самые различные изображения: буквы, цифры, иконки и пр. Так создаются символьные ЖКИ. А при создании массива электродов (ортогональной матрицы) можно получить графический ЖКИ с разрешением, определяемым количеством задействованных электродов.

Большинство современных ЖК индикаторов и дисплеев управляются от микроконтроллеров, которые располагаются на обратной стороне индикатора. Микроконтроллер преобразует цифровые коды, содержащие информацию об отображаемом объекте, в последовательность импульсов напряжения для электродов индикатора. В постоянной памяти этого МК записаны образы символов, которые могут отображаться индикатором (функция знакогенератора). В оперативной памяти МК размещаются коды тех символов, которые высвечиваются на экране индикатора в текущий момент времени. Поэтому, когда Вы производите подключение ЖК индикатора к МК, Вы фактически устанавливаете связь между двумя микропроцессорными системами.

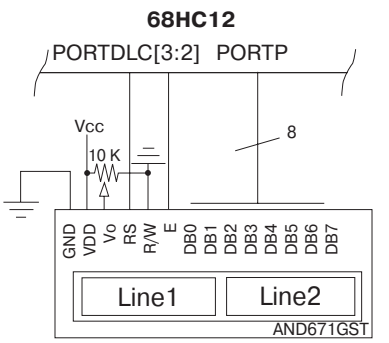
В следующем параграфе мы рассмотрим цифро-буквенный ЖК индикатор с управлением от стандартного контроллера AND671GST с параллельным интерфейсом обмена. Мы также приведем пример программного кода для управления этим индикатором.

5.6.2. Сопряжение МК с символьным ЖК индикатором

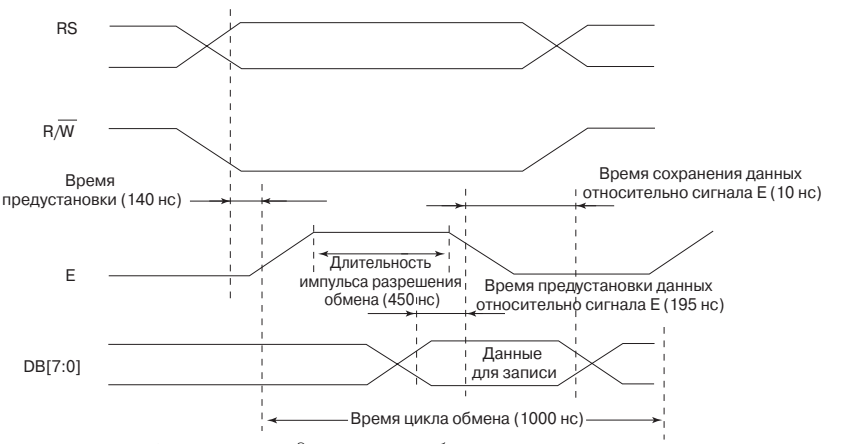
AND671GST – однострочный цифро-буквенный дисплей на 16 знакомест. Для управления такими дисплеями используются интегральные схемы контроллеров HD44100H или HD44780. Контроллер с сопутствующими элементами размещается на печатной плате, которая помещается за индикатором и образует с ним единую конструкцию. На плате

Номер вывода	Обозначение	Функция
1	GND	Общий вывод источника питания
2	VDD	Напряжение питания контроллера 5,0 В
3	V0	Напряжение питания ЖК матрицы
4	RS	Выбор регистра контроллера: 0 – регистр управления, 1 – регистр данных
5	R/W	Выбор режима обмена чтение/запись: 0 – запись, 1 – чтение
6	E	Разрешение обмена
7	DB0	Двунаправленная магистраль данных контроллера ЖК индикатора
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7	

а) Описание выводов контроллера ЖК индикатора



б) Схема подключения ЖК индикатора к МК



в) Временные диаграммы обмена в режиме записи

Рис. 5.12. *Цифро-буквенный ЖК индикатор*

контроллера смонтирован разъем на 14 контактов. На разъем выведены сигналы стандартизованного параллельного интерфейса для сопряжения символьных ЖК индикаторов с микропроцессорными системами. Назначение сигналов этого интерфейса, имена линий и соответствующие им номера контактов разъема приведены на рис. 5.12,а.

Схема подключения индикатора к МК семейства 68HC12 представлена на рис. 5.12,б. Обмен данными в параллельном однобайтовом формате производится через порт PORT P. В нашем примере мы будем только записывать данные в контроллер индикатора, поэтому порт может быть однонаправленным. В реальных задачах иногда приходится читать информацию из памяти контроллера управления дисплеем, тогда следует использовать двунаправленный порт. Для управления режимами обмена информацией с контроллером используются два дополнительных сигнала: E – разрешение обмена, и RS – выбор регистра для обмена. Эти две линии подключены к линиям другого порта вывода МК. Третий сигнал управления обменом – линия выбора направления обмена (чтение или запись) подключена к общему выводу, поскольку в примере мы будем использовать только режим записи.

Три контакта разъема индикатора предназначены для подключения двух источников напряжения: V_{DD} – напряжение питания цифровой части (схемы управления индикатора), V_O – напряжение смещения электродов. Последнее называют напряжением управления контрастностью индикатора. Вывод V_O подключен к средней точке потенциометра, поэтому напряжение V_O может изменяться от 0 до 5,0 В, обеспечивая таким образом выбор наилучшего режима отображения.

Временные диаграммы обмена. Различают два типа данных, которые может передать МК контроллеру индикатора: команды управления и данные, т.е. коды символов для отображения. В режиме чтения МК может получить от контроллера управления дисплеем слово-состояние и данные из внутреннего ОЗУ символов. В нашем примере мы не будем пользоваться режимом чтения.

Временные диаграммы одного цикла записи в контроллер дисплея приведены на рис. 5.12,в. Приведенные на рисунке длительности временных интервалов определяются конкретным типом выбранного индикатора, поэтому могут достаточно существенно различаться. Об этом следует помнить при использовании ранее написанного и отлаженного программного кода для управления индикатором в последующих разработках. Если очередной индикатор требует больших времен установления по сравнению с предыдущим, то программный код может оказаться неработоспособным.

В соответствии с приведенными временными диаграммами (рис. 5.12,в) должна сгенерировать следующую последовательность сигналов на выходах портов МК:

- Линия RS должна быть установлена в соответствии с типом передаваемых данных: 1 – запись данных, 0 – запись команды;
- С задержкой в 140 нс относительно сигнала RS должна быть установлена в 1 линия разрешения обмена E;
- Длительность импульса на линии E не должна быть менее 450 нс;
- Байт данных должен быть выставлен на линии DB0...DB7 не позднее, чем за 195 нс до спадающего фронта сигнала E. Это время называют временем предустановки данных относительно сигнала разрешения E;
- Данные на линиях DB0...DB7 должны оставаться неизменными в течение не менее 10 нс после перехода сигнала E в неактивное (логический 0) состояние. Это время называют временем сохранения данных относительно сигнала E;
- Период сигнала E не должен быть менее 1000 нс.

Основываясь на требованиях временной диаграммы обмена, следующий алго-

ритм должен быть реализован микроконтроллером для выполнения одного цикла записи в контроллер дисплея:

1. Установить линию RS в 1 или в 0 (1 – для записи кода символа, 0 – для записи кода команды управления дисплеем);
2. Установить активный уровень на линии разрешения E (логическая 1);
3. Выдать на порт PORT P байт данных (код символа или команда управления);
4. Перевести линию E в неактивное состояние (логический 0).

В процессе создания на основе этого алгоритма программы управления следует учесть все перечисленные ранее задержки между сигналами.

Программа управления ЖК индикатором. Структура программы управления символьным ЖК индикатором представлена на рис. 5.13,а. Программа включает коллекцию функций, каждая из которых выполняет законченное смысловое действие. Разбиение задачи на отдельные функции – искусство разработчика. От выбранного набора функций и способов передачи параметров между ними зависит число возможных ошибок при последующем использовании этих функций в прикладной программе и компактность файла исполняемых кодов.

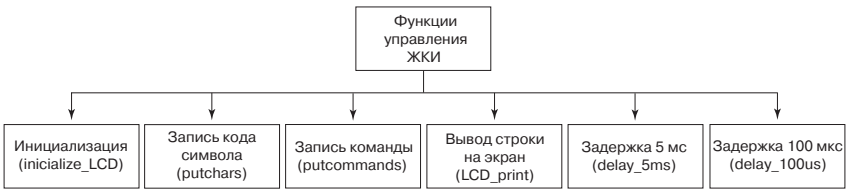
Блок-схема функции инициализации дисплея представлена на рис. 5.13,б. Она реализует алгоритм начальной установки дисплея, который разработан в соответствии с техническими рекомендациями производителя.

Ниже приведен текст программы на Си для всех функций рис. 5.13,а.

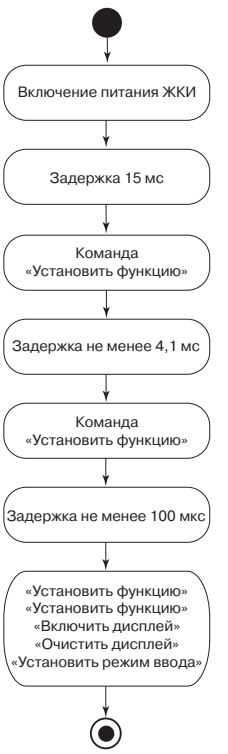
```
/*-------------------------------------------------------------------*/
/*filename: lcd.c                                                    */
/*содержит программный код для шести функций управления ЖК дисплеем */
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
/* Функция initia1ize_lcd производит начальную установку режимов дисплея */
/*-------------------------------------------------------------------*/

void initia1ize_lcd(void)
{
de1ay_5ms() ;
de1ay_5ms();
de1ay_5ms() ;              //задержка 15 мс

putcommands(0x38);         //команда установки формата интерфейса обмена
de1ay_5ms();

putcommands(0x38) ;
de1ay_l00us();

putcommands(0x38);
putcommands(0x38) ;
putcommands(0x0C) ;
putcommands(0x0l);         //очистить дисплей
putcommands(0x06);         //установить режим ввода с автоматическим
                           //увеличением адреса
                           //символа на 1
putcommands(0x0E) ;        //включить дисплей, режим курсора мигающий
putcommands(0x02) ;        //установить курсор на первое знакоместо
}
/*-------------------------------------------------------------------*/
```

```
/* Функция putchars производит запись одного кода символа в контроллер */
/*дисплея                                                              */
/*-------------------------------------------------------------------*/
void putchars(unsigned char c)
{
DDRP = 0xFF;                //установить порт PORT P на вывод
DDRDLC = DDRDLC | 0x0C;     //установить разряды 2 и 3 порта PORT DLC на
                            //вывод
PORTP = c;                  //Выдать в порт PORT P код символа c
PORTDLC = PORTDLC | 0x08;   //установить линию RS в 1
PORTDLC = PORTDLC | 0x04;   //установить линию E в 1
PORTDLC = 0;                //установить E и RS в 0
delay_5ms ()                //задержка 5 мс
}
/*-------------------------------------------------------------------*/
/*Функция putcommands производит запись одного кода команды в контроллер */
/*дисплея
/*-------------------------------------------------------------------*/
void putcommands(unsigned char d)
{
DDRP = 0xFF;                //установить порт PORT P на вывод
DDRDLC = DDRDLCIOxOC;       //установить разряды 2 и 3 порта PORT DLC на
                            //вывод
PORTDLC = PORTDLC & 0xF7;   //установить линию RS в 0
PORTP = d;                  //Выдать в порт PORT P код команды d
PORTDLC = PORTDLC | 0x04;   //установить линию E в 1
PORTDLC = 0;                //установить E и RS в 0
delay_5ms() ;               //задержка 5 мс
}
/*-------------------------------------------------------------------*/
/*Функция 1cd-print производит запись строки символов в ОЗУ дисплея   */
/*-------------------------------------------------------------------*/
void 1cd-print(char *string)
{
putcommands(0x02);          //команда установки адреса на начало строки

while(*(string) != '\0')    //вывести последовательно кода символов
       {                    //из памяти МК
       putchars(*string) ;
       string++;
       }
}
/*-------------------------------------------------------------------*/
/* Функция delay_5ms формирует задержку 5 мс                         */
/*-------------------------------------------------------------------*/
void delay_5ms(void)
{
int i;
for(i=0; i<50; i++)
       {
       delay_l00us() ;
       }
}

/*-------------------------------------------------------------------*/
```

```
/*---------------------------------------------------------------------*/
/* Функция void delay_100us формирует задержку 100 мкс                 */
/*Частота системной шины равна 8 МГц                                   */
/*---------------------------------------------------------------------*/
void delay_100us(void)
{
int i;
for(i=0; i<50; i++)
       {
       asm( "nop");         //команда nop занимает 2 цикла
       }
}
/*---------------------------------------------------------------------*/
```



а) Набор функций управления



б) Блок-схема алгоритма инициализации

Рис. 5.13. *Программное обеспечение для управления цифро-буквенным ЖК индикатором*

5.6.3 Сопряжение МК с графическим ЖК дисплеем

В этом разделе мы рассмотрим типовой графический ЖК дисплей. Следуя логике предыдущего параграфа, мы сначала обсудим информационную модель графического дисплея, затем изучим электрические характеристики и временные диаграммы обмена встроенного контроллера управления этим дисплеем, в завершении – элементы программы для передачи данных из МК в контроллер дисплея. В конце параграфа мы предложим Вам набор полезных для обслуживания графического дисплея функций. При рассмотрении мы будем использовать конкретную модель дисплея AND1391ST. Однако полученные знания Вы сможете легко применить к другим распространенным моделям дисплеев. В главе 7 мы предложим Вам подробный полностью завершенный пример с графическим дисплеем (разд. 7.).

Информационная модель. AND1391ST – ЖК дисплей с разрешающей способностью экрана 128 × 128 пикселов. Встроенный контроллер управления обеспечивает работу дисплея как в символьном, так и в графическом режиме отображения. Возможно также сочетание этих двух режимов работы при выводе одной и той же картинки. При использовании символьного режима поле экрана дисплея делится на 16 строк по 16 символов в каждой строке (рис. 5.14). Для отображения каждого символа предоставляется поле знакоместа размером 8 × 8 точек, как показано на рис. 5.15. В качестве дополнительного может быть использован режим символьного отображения, при котором поле экрана дисплея делится на 16 строк по 21 символу в стро-

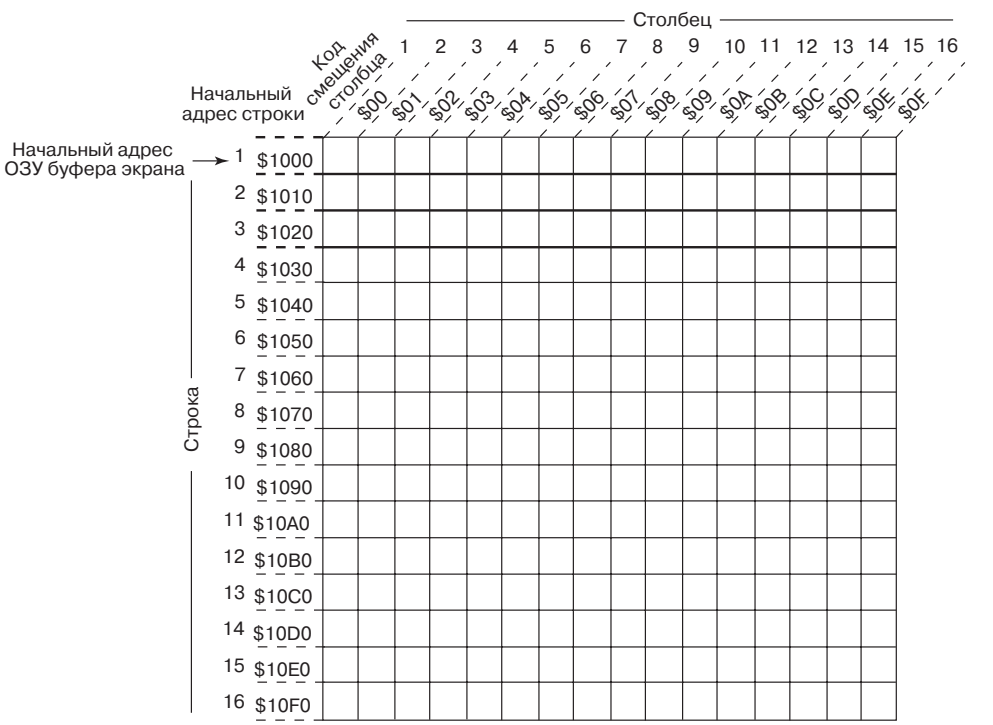


Рис. 5.14. *Информационная модель графического ЖК дисплея в символьном режиме*

младший полубайт / старший полубайт	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
1	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
2	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
3	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
4	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
5	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Рис. 5.15. *Таблица кодов символов для графического ЖК дисплея*

ке. В этом случае каждое знакоместо будет состоять из 6 × 8 точек. Размер знакоместа задается уровнем сигнала на входе FS контроллера дисплея. Набор кодов для формирования образа каждого символа (знакогенератор) хранится в постоянной памяти контроллера дисплея.

Контроллер дисплея воспроизводит на экране образы символов, коды которых хранятся в ячейках оперативной памяти контроллера. Каждому знакоместу на экране дисплея поставлена в соответствие ячейка ОЗУ с определенным адресом. В процессе инициализации контроллера дисплея МК записывает адрес начала ОЗУ экрана. На рис. 5.14 этот адрес равен S1000. Все остальные адреса для ячеек символов вычисляются посредством указанных на рис. 5.14 кодов смещения. В процессе инициализации также указывается число символов в строке. В нашем примере мы используем режим отображения с 16 символами в строке. При этом размер знакоместа для символа (8 × 8) уже выбран на аппаратном уровне: вход FS контроллера присоединен к общему выводу источника питания.

Схема подключения дисплея AND1391ST к МК семейства 68HC12 представлена на рис. 5.16, а. Наименование и краткое описание выводов контроллера управления дисплеем AND1391ST приведены в таблице рис. 5.16, б.

Проанализировав записи в табл. 5.16, б, Вы увидите, что графический дисплей требует для своей работы двуполярного источника питания. Положительное напряжение питания обеспечивает работу контроллера управления, в то время как отрицательное напряжение необходимо для регулирования контрастности получаемого на дисплее изображения.

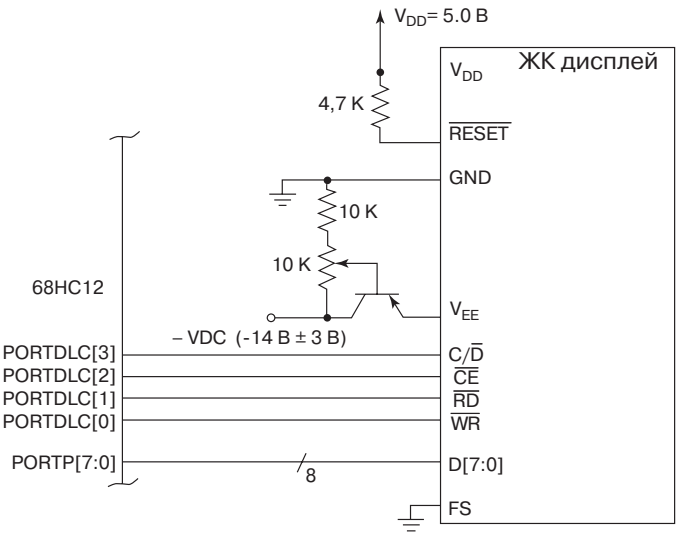
В нашем примере мы использовали линии портов PORTP и PORTDLC для обмена данными между МК и дисплеем. Двунаправленная магистраль данных контроллера дисплея D[7...0] подключена к линиям PORTP[7...0] двунаправленного порта PORTP (рис. 5.16, а). В отличие от предыдущего примера, обмен данными между МК и контроллером дисплея происходит в двух направлениях. Поэтому в процессе взаимодействия с дисплеем МК многократно перепрограммирует регистр направления передачи DDRP, изменяя режим работы линий порта PORTP (ввод или вывод).

Для управления режимами обмена информацией с контроллером используются четыре дополнительных сигнала, которые формируются МК на линиях PORTDLC[3...0] порта PORTDLC (рис. 5.16, а). Назначение и краткое описание сигналов управления дисплеем приведено в таблице рис. 5.16, б. Там же показаны комбинации управляющих сигналов, которые следует использовать при обмене с дисплеем различными типами данных.

Временные диаграммы обмена. Временные диаграммы обмена с контроллером управления графическим ЖК дисплеем приведены на рис. 5.17. Они аналогичны рассмотренным ранее диаграммам обмена с контроллером цифро-буквенного индикатора. Так же, как и в предыдущем случае, мы будем генерировать сигналы управления в соответствие с приведенной временной диаграммой, последовательно переключая линии порта PORTDLC в программе управления дисплеем

Набор функций управления графическим ЖК дисплеем. Мы предлагаем Вашему вниманию набор функций на Си, которые могут быть использованы в других программах для эффективного управления графическим ЖК дисплеем. Для каждой функции приведено описание и исходный текст программы на Си. На рис. 5.18 показана структура программного обеспечения для управления графическим дисплеем, в которой отражены полный набор функций управления и их взаимосвязь. В одном из самостоятельных заданий к этой главе мы попросим Вас разработать блок-схемы алгоритмов для реализации каждой из перечисленных функций управления.

```
//-------------------------------------------------------------------
//filename: 2D_LCD.c содержит программный код для 14 функций управления
//графическим ЖК дисплеем
//-------------------------------------------------------------------
//Схема подключения дисплея AND1391ST к МК 68HC12
//   PORTDLC[3] - C/D
//   PORTDLC[2] - CE
//   PORTDLC[1] - RD
//   PORTDLC[0] - WR
//   PORTP[7…0] - D[7…0]
//  вывод RESET дисплея AND1391ST через резистор 4,7 кОм к источнику питания
//   вывод FS дисплея AND1391ST к общему выводу источника питания
// ------------------------------------------------------------------
// Функция initia1ize_lcd производит начальную установку режимов
//графического дисплея
// ------------------------------------------------------------------
void initia1ize_lcd(void)
{
char temp = 0x00;
PORTDLC = 0xFF;     //установить 1 на всех выходах порта: запрет всех
                    //действий с дисплеем
PORTDLC = PORTDLC&0xEF;    //сброс экрана, RESET=0
delay(2000) ;              //задержка 2 мс
PORTDLC = 0x7F;     //установить вывод RESET в 1

write() ;           //установить WR=0
command(0x80) ;     //установить режим работы текстовый

data(0x00) ;        //слово управления
data (0x10) ;       //слово управления
command (0x40) ;    //установить адрес начала текста
```



а) Схема подключения ЖК дисплея к МК

Номер вывода	Обозначение	Функция
1	FGND	Вывод корпуса
2	GND	Общий вывод источника питания
3	VDD	Напряжение питания контроллера 5.0 В
4	VEE	Напряжение питания ЖК матрицы –14 В ± 3 В
5	WR	Линия управления «запись»
6	RD	Линия управления «чтение»
7	CE	Линия управления «разрешение обмена»
8	C/D	Выбор режима обмена: 0 – чтение/запись данных 1 – чтение регистра состояния/запись команды
9	NC	Не подсоединен
10	RESET	Начальная установка контроллера
11	DB0	Двунаправленная магистраль данных контроллера ЖК индикатора
12	DB1	
13	DB2	
14	DB3	
15	DB4	
16	DB5	
17	DB6	
18	DB7	
19	FS	Выбор размера символа
20	NC	Не подсоединен

б) Описание выводов контроллера графического ЖК дисплея

Рис. 5.16. *Сопряжение графического ЖК дисплея с МК*

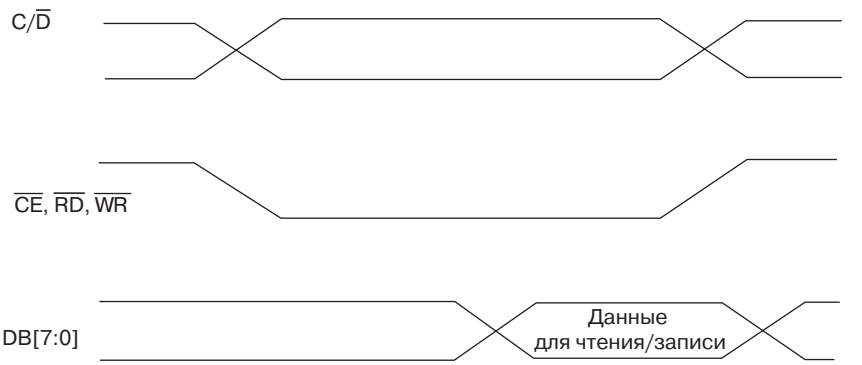


Рис. 5.17. *Временные диаграммы обмена контроллера графического ЖК дисплея*

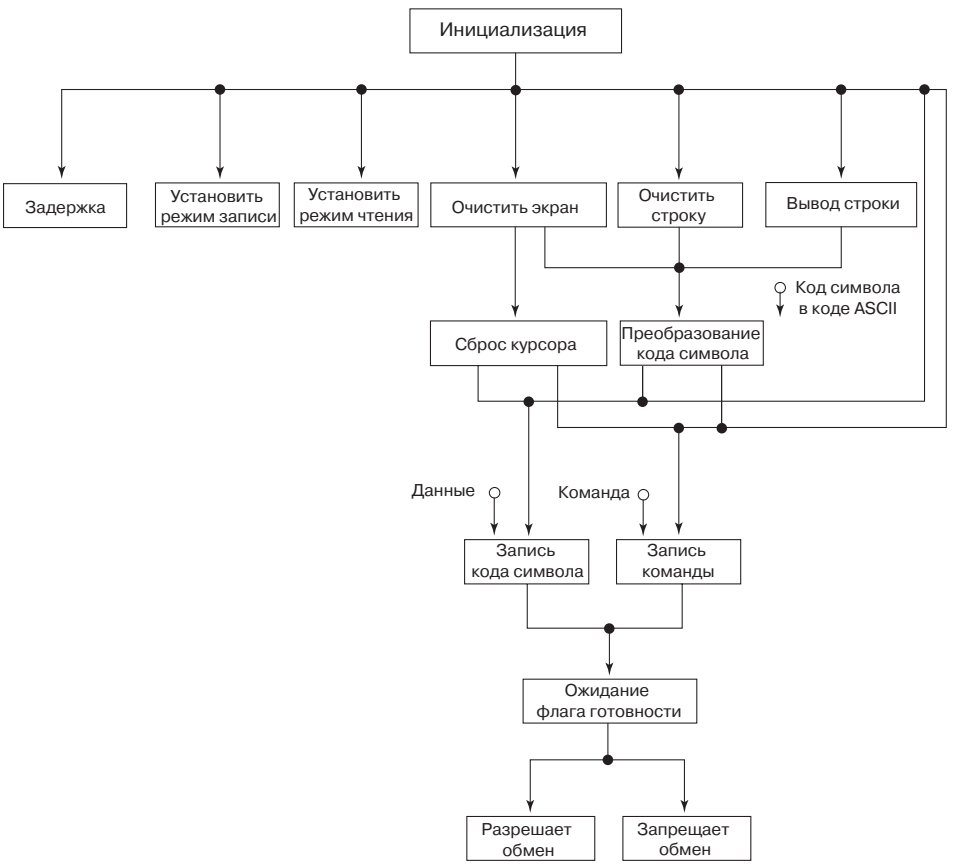


Рис. 5.18. *Структура программного обеспечения графического ЖК дисплея*

```
data (0x10) ;              // слово управления
data(0x00);                // слово управления
command (0x41) ;           // установить область текста: 16 знаков

command(0x94) ;            //выключить дисплей, курсор мигает
command (0xA7) ;           //установить курсор 8?8 точек

data(0x01);                // слово управления
data (0x01) ;              // слово управления
command (0x21) ;           //установить позицию курсора
}
// ---------------------------------------------------------------------
// Функция read конфигурирует линии порта PORTP для ввода данных от контроллера
// дисплея.
// ---------------------------------------------------------------------
void read()
{
DDRP = 0x00; //порт PORTP на ввод
}
// ---------------------------------------------------------------------
// Функция Clearscreen производит очистку экрана дисплея посредством записи
//во все ячейки памяти буфера экрана кода символа " "
// ---------------------------------------------------------------------
void Clearscreen()
{
int i,j;
Reset_cursor();

//выполнить для каждой стоки (i), для каждого знакоместа в строке (j)
for(i=0;i<16;i++)
for(j=0;j<16;j++)
LCD_char(' ');
Reset_cursor() ;
}
// ---------------------------------------------------------------------
// Функция newline производит запись во все знакоместа одной код символа " "
// ---------------------------------------------------------------------
void newline ()
{
int i;
for(i=0;i<16;i++)
       LCD_char (' ');
}
// ---------------------------------------------------------------------
// Функция LCD_output производит преобразование кодов ASCII строки символов
//в коды табл. рис. 5.15 для отображения на дисплее и передает эту строку в
//ОЗУ буфера экрана дисплея
// ---------------------------------------------------------------------
void LCD_output(char s[])
{
int n = 0;
while(s[n] 1= '\0')
       {
       LCD_char(s[n]) ;
```

```
        ++n;
        }
}
// ---------------------------------------------------------------------
// Функция delay формирует временную задержку длительностью в указанное
//число мкс
// ---------------------------------------------------------------------
void delay(int usec)
{
int i,j;
for(i=0;i<usec; i++)
        {
        for(j=0; j < 7; j++)
                { }
        }
}
// ---------------------------------------------------------------------
// Функция write конфигурирует линии порта PORTP для вывода данных на
// дисплей
// ---------------------------------------------------------------------
void write()
{
DDRP = 0xFF;         //Порт PORTP на вывод
}
// ---------------------------------------------------------------------
// Функция data производит запись одного символа в ОЗУ данных дисплея. Перед
// обменом с контроллером дисплея контролируется бит состояния, который
// свидетельствует о том, закончил контроллер выполнение предыдущей команды
// управления или нет. Затем на порт PORTP выставляется код символа и
// формируются необходимые сигналы управления
// ---------------------------------------------------------------------
void data(unsigned char n)
{
status_wait();
PORTP = n;
PORTDLC = 0xFF;
PORTDLC = PORTDLC & 0xF7; //C/D в 0
PORTDLC = PORTDLC & 0xFE; //WR в 0
PORTDLC = PORTDLC & 0xFB;
enable ();
disable();
}
// ---------------------------------------------------------------------
// Функция command производит передачу команды управления в контроллер
// дисплея
// ---------------------------------------------------------------------
void command(unsigned char n)
{
status_wait();
PORTP = n;
PORTDLC = 0xFF;
PORTDLC = PORTDLC&0xFE;
enable ();
disable();
}
```

```
// ---------------------------------------------------------------------
// Функция status_wait производит опрос байта состояния контроллера дисплея
//и ожидаетдо тех пор, пока бит состояния не станет равным 1. Тогда
//контроллер дисплея будетготов к исполнению новой команды
// ---------------------------------------------------------------------
void status_wait()
{
char temp = 0x00;
DDRP = 0x00;
PORTDLC = PORTDLC | 0x0F;
PORTDLC = PORTDLC&0xFD;
enable ();

while((temp&0x03) != 0x03)
{
temp = PORTP;
}
disable ();
DDRP = 0xFF;
}
// ---------------------------------------------------------------------
// Функция enable устанавливает линию разрешения обмена в 0
// ---------------------------------------------------------------------
void enable(void)
{
PORTDLC = PORTDLC | 0x04;
PORTDLC = PORTDLC&0xFB;
}
// ---------------------------------------------------------------------
// Функция disable устанавливает линию разрешения обмена в 1
// ---------------------------------------------------------------------
void disable(void)
{
PORTDLC = PORTDLC | 0x04;
}
// ---------------------------------------------------------------------
// Функция Reset_cursor возвращает курсор в левый верхний угол экрана
// ---------------------------------------------------------------------
void Reset_cursor()
{
data(0x00) ;
data(0x10) ;
command (0x24) ;
}
// ---------------------------------------------------------------------
// Функция LCD_char преобразует код ASCII символа в код для отображения на
//экране //дисплея в соответствии с табл. рис. 5.15.
// ---------------------------------------------------------------------
void LCD_char(unsigned char n)
{
data(n - 0x20);
command (0xC0) ;
}
// ---------------------------------------------------------------------
```

5.7. Управление электрическим двигателем

В главе 4 мы рассмотрели применение способа широтно-импульсной модуляции для регулирования напряжения, приложенного к обмоткам электрического двигателя. Изменяя коэффициент модуляции, мы изменяли длительность импульсов напряжения на двигателе, сохраняя частоту следования этих импульсов неизменной. В результате, изменялось среднее значение напряжения на двигателе, и, как следствие, скорость его вращения. При обсуждении мы отметили, что ШИМ-сигнал может быть сформирован на одном из выходов МК, однако его мощности не будет достаточно для приведения двигателя во вращение. Поэтому между выходом МК и двигателем должны быть специальные электронные цепи, которые позволяют усилить по мощности, формируемый микроконтроллером ШИМ-сигнал. Мы рассмотрим примеры таких цепей в данном параграфе.

5.7.1. Силовые полупроводниковые ключи

Для подключения к обмоткам двигателя под управлением МК источника напряжения достаточной мощности могут быть использованы различные приборы: электромагнитные реле, твердотельные реле, биполярные транзисторы и некоторые другие типы транзисторов. В нашем примере мы будем использовать для этой цели мощные полевые транзисторы. В русскоязычной литературе их называют мощными МДП-транзисторами (МДП – Металл-Диэлектрик-Полупроводник), в англоязычной литературе используют аббревиатуру MOSFET (Metal Oxide Semiconductor Field-Effect Transistor). Мы остановились на этом типе полупроводниковых приборов потому, что в современных коммутаторах для двигателей малой и средней мощности используются именно эта элементная база.

Перед тем, как исследовать электронную схему усиления мощности, необходимую для управления электрическим двигателем от микроконтроллера, рассмотрим принцип действия МДП-транзистора.

МДП-транзистор – это управляемый напряжением полупроводниковый ключ. Он обладает очень высоким сопротивлением цепи управления, что удобно для микроконтроллера, который не может формировать больших вытекающих токов. В открытом состоянии падение напряжения на транзисторе мало по сравнению с другими типами полупроводниковых ключей аналогичной мощности. Поэтому энергия, рассеиваемая МДП-транзистором в режиме проводимости, также относительно невелика, что обеспечивает высокий коэффициент полезного действия полупроводникового коммутатора.

МДП-транзистрор может работать как в режиме усиления сигнала, так и в ключевом режиме. Мы будем использовать МДП-транзистр в ключевом режиме для коммутации напряжения питания к обмотке двигателя во время длительности импульса ШИМ-сигнала.

МДП-транзистор имеет три электрода: сток (Drain), исток (Souse) и затвор (Gate). Затвор – это управляющий электрод транзистора, в то время как сток и исток – это электроды, по которым протекает коммутируемый транзистором ток. Схемное обозначение МДП-транзистора с каналом n-типа приведено на рис. 5.19,а. При отсутствии напряжения между затвором и стоком канал для протекания тока

между стоком и истоком внутри полупроводниковой структуры транзистора отсутствует. Если на затвор подать положительное напряжение относительно истока, то внутри транзистора формируется канал для протекания тока от стока к истоку. Если снять с затвора напряжение, то этот канал исчезнет, и транзистор не сможет проводить ток между истоком и стоком. Мы получили управляемый полупроводниковый ключ!

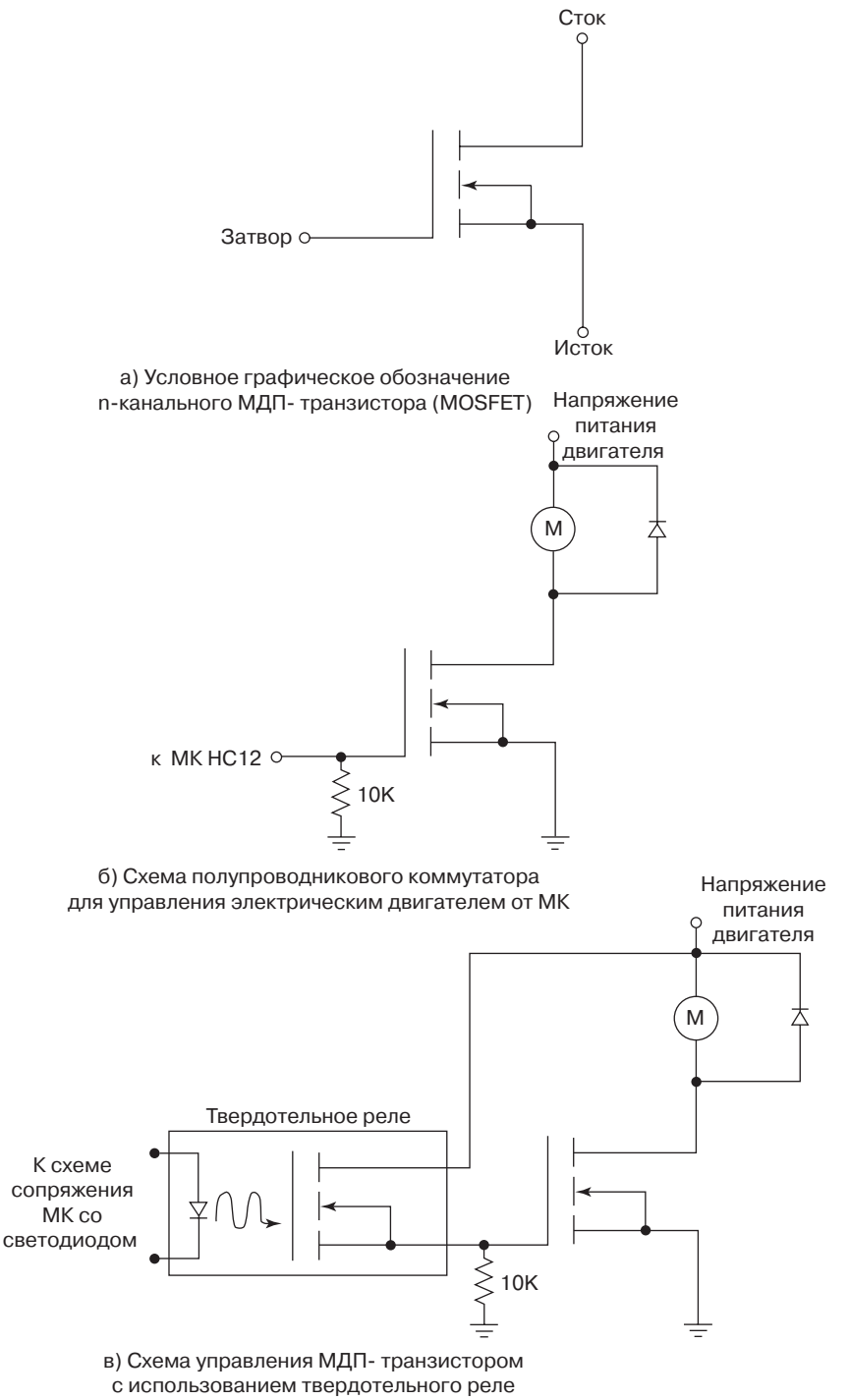
В нашем примере мы использовали МДП-транзисторы IRF530 компании International Rectifier. Максимальное значение тока стока ID для этого транзистора равно 14 А. Это означает, что мы можем коммутировать нагрузку с номинальным током 14 А под управлением маломощного сигнала с выхода микроконтроллера. Отметим, что в настоящее время подобные мощные МДП-транзисторы с каналами n- и p-типа выпускаются на токи свыше 100 А.

На рис. 5.19,б показана схема включения электрического двигателя с управлением от микроконтроллера. Резистор R = 10 кОм в цепи управления обеспечивает путь для рассасывания заряда из области управляющего электрода, когда напряжение на затворе становится равным нулю. Также в схеме присутствует защитный диод, который установлен параллельно двигателю. Мы обсудим назначение этого диода несколько позже, при изучении схемы инвертора (раздел 5.7.3). В нашем примере на затвор транзистора подается управляющее напряжение около 5 В. При этом транзистор IRF530 может проводить ток около 4 А. Для увеличения тока нагрузки до 14 А следует повысить управляющее напряжение. Однако используемый в примере электрический двигатель приводится во вращение напряжением 12 В при токе нагрузки 1 А. Поэтому параметры напряжения управления от микроконтроллера для нашего случая вполне удовлетворяют требованиям МДП-транзистора.

5.7.2. Оптоэлектронная потенциальная развязка

Электрический двигатель – печально известный для инженеров-электроников источник электромагнитных помех. В процессе регулирования скорости вращения электрического двигателя, помехи в равной мере генерируются как самим двигателем, так и импульсным источником питания. Транзисторный коммутатор формирует на обмотке двигателя импульсы напряжения требуемой длительности. В момент выключения мощного МДП-транзистора реактивные элементы схемы сбрасывают накопленную во время протекания тока энергию в питающую сеть. В результате генерируются кратковременные выбросы напряжения в питающую сеть. Последнее может неблагоприятно сказаться на работе управляющего МК. В частности, из-за помех может произойти нарушение последовательности исполнения команд программы управления, или сбой в системе тактирования МК. И случится нарушение работы всей системы в целом.

Чтобы избежать столь неблагоприятного прогноза в работе системы, следует изолировать цепи управления, т.е. плату с МК 68HC12, от силовых цепей установки. Для этого могут быть использованы оптроны (рис. 5.19,в). Управляющая цепь МК подключается к цепи излучающего светодиода оптрона. Когда МК формирует сигнал на включение МДП-транзистра, через светодиод протекает ток. Энергия излучения светодиода по оптическому каналу, расположенному внутри ИС оптрона, достигает полупроводникового прибора на вторичной стороне, и переводит его в открытое состояние. Одним из приборов, относящихся к классу опт-



а) Условное графическое обозначение n-канального МДП- транзистора (MOSFET)

б) Схема полупроводникового коммутатора для управления электрическим двигателем от МК

в) Схема управления МДП- транзистором с использованием твердотельного реле

Рис. 5.19. *Управление электрическим двигателем от МК*

ронов, является твердотельное реле (рис. 5.19,в). На вторичной стороне этого прибора установлен маломощный МДП-транзистор. Когда светодиод излучает, МДП-транзистор открывается, по цепи резистора протекает ток, и на затворе основного транзистора формируется положительное напряжение относительно истока. Основной транзистор включается, и к обмотке двигателя подключается напряжение источника питания.

Для подключения светодиода оптрона к выходу МК следует воспользоваться изученным ранее способом подключения обычного светодиода. Мы попросим Вас выполнить такое подключение в задании №8 в конце данной главы.

5.7.3. Инвертор напряжения

В предыдущем параграфе Вы научились управлять скоростью вращения электрического двигателя при помощи мощных полупроводниковых ключей. Этот способ отличается высоким коэффициентом полезного действия, поскольку потери в МДП-транзисторе во время протекания тока невелики. А как насчет изменения направления вращения двигателя? Если прикладная задача требует не только регулирования скорости вращения двигателя, но и изменения направления вращения, то предыдущая схема на одном МДП-транзисторе не может быть использована. Однако, увеличив число управляемых ключей схемы до четырех, мы получим необходимое решение.

На рис. 5.20,а представлена схема однофазного мостового инвертора напряжения, которая позволяет регулировать скорость вращения двигателя методом ШИМ при любом направлении вращения двигателя. В англоязычной литературе эту схему называют «H bridge», что в переводе означает «мост в форме буквы Н».

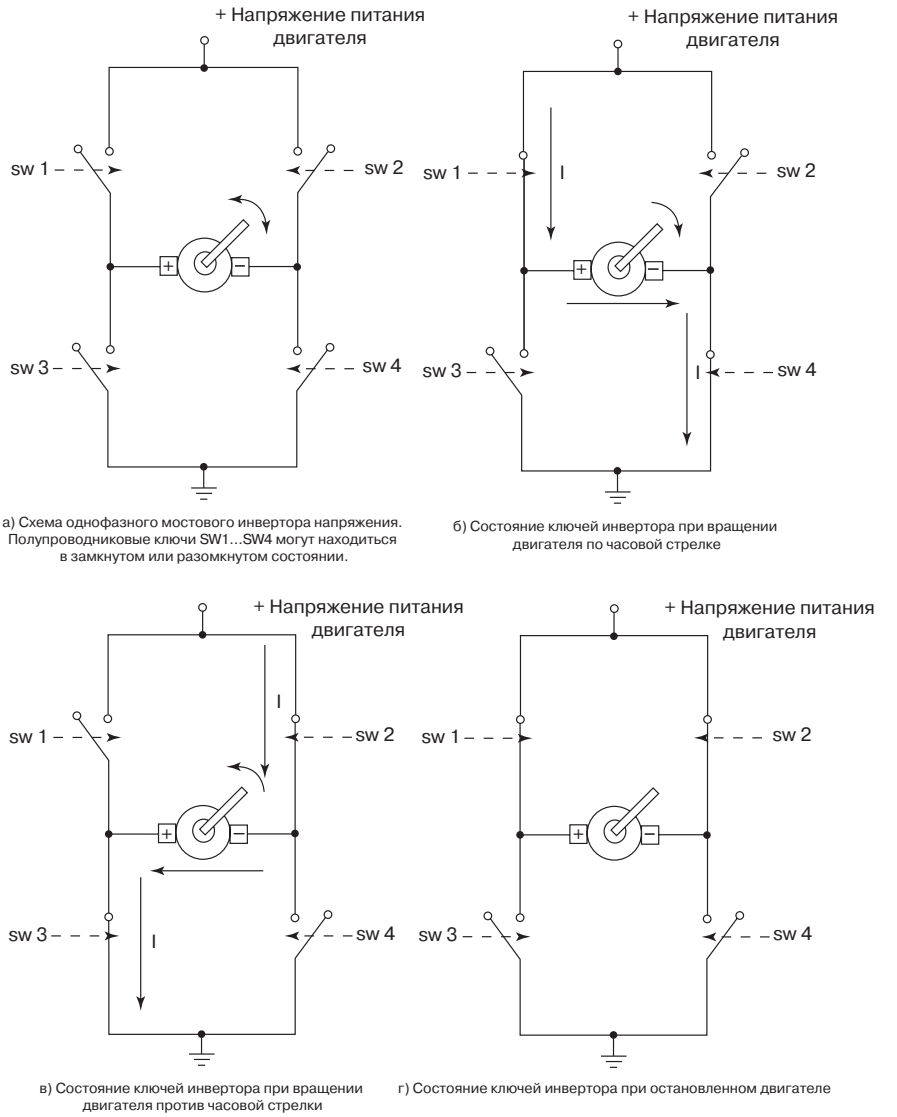
В каждом плече моста установлен управляемый полупроводниковый ключ. Обмотка двигателя подключена между двумя средними точками. При включенных транзисторах sw1 и sw4 обмотка двигателя подключается левым концом к положительной шине питания, а правым – к отрицательной. Ток протекает в указанном на рис. 5.20,б направлении, и двигатель вращается по часовой стрелке. Если транзисторы sw1 и sw4 выключить, и включить транзисторы sw2 и sw3, то ток будет протекать в противоположном направлении (рис. 5.20,в), а двигатель будет вращаться против часовой стрелки. Если одновременно включить одну из пар транзисторов sw1 и sw2 или sw3 и sw4, то обмотка двигателя будет закорочена, и он постепенно остановит свое вращение. Ни в коем случае нельзя одновременно включать пары транзисторов sw1 и sw3 или sw2 и sw4. Произойдет короткое замыкание источника питания.

Реальная схема мостового инвертора, конечно, имеет специальные средства защиты от короткого замыкания как в цепи нагрузки, так и в цепи каждого плеча инвертора. При значительных превышениях номинального тока эта схема автоматически отключит источник питания, и полупроводниковые ключи останутся работоспособными.

В заключение отметим, что малые потери в МДП-транзисторах позволяют разместить мостовую схему инвертора в корпусе DIP16! Так ИС SN754410NE компании Texas Instruments содержит в себе сразу два таких моста. Она позволяет коммутировать напряжения от 4,5 до 36 В при максимальном токе нагрузки 1 А. Упрощенная схема одного мостового инвертора в составе SN754410NE приведена на рис. 5.21.

5.8. Кодовый замок

В этом заключительном примере мы объединили ранее полученные знания по аппаратному подключению и программному обслуживанию различных устройств ввода и вывода информации. В каждом параграфе данной главы мы рассмотрели какой-то отдельный тип устройства ввода/вывода. И для каждого устройства привели фрагмент программного кода, который необходим для его обслуживания. Эти фрагменты могут быть использованы Вами в реальных разработках. На примере электронного кодового замка мы покажем Вам, как воспользоваться полученными ранее знаниями.



а) Схема однофазного мостового инвертора напряжения. Полупроводниковые ключи SW1...SW4 могут находиться в замкнутом или разомкнутом состоянии.

б) Состояние ключей инвертора при вращении двигателя по часовой стрелке

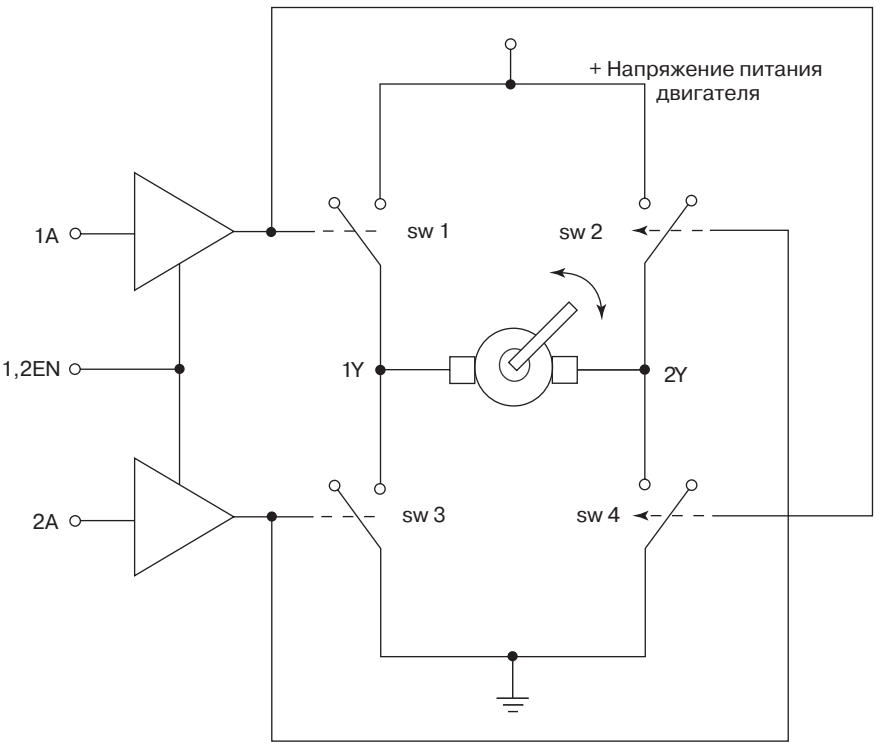
в) Состояние ключей инвертора при вращении двигателя против часовой стрелки

г) Состояние ключей инвертора при остановленном двигателе

Рис. 5.20. *Однофазный мостовой инвертор напряжения*

Рассмотренная ранее матричная клавиатура используется для ввода четырех символов кода доступа. Если последовательность этих символов совпадает с эталонной последовательностью, которая хранится в памяти программы, то доступ разрешается. Если же введенная последовательность символов ошибочная, то доступ блокируется. Для информирования пользователя о состоянии системы доступа используется символьный ЖК индикатор.

Далее мы приводим функциональную схему аппаратных средств, блок-схему алгоритма работы и полную программу управления кодовым замком.



а) Функциональная схема

Входы		Состояние двигателя
1A	2A	
0	0	Обмотка не подключена, возможно свободное вращение
0	1	Вращение против часовой стрелки
1	0	Вращение по часовой стрелке
1	1	Запрещенное состояние

б) Таблица состояний

Рис. 5.21. *Функциональная схема однофазного мостового инвертора в составе специализированной ИС SN754410NE*

5.8.1. Схема подключения периферийных устройств

Электрическая схема подключения клавиатуры на 16 клавиш и символьного ЖК индикатора к выводам портов МК семейства 68HC12 приведена на рис. 5.22. Линии порта PORT B обслуживают матричную клавиатуру. В соответствии с электрической схемой линии PORTB[0]...PORTB[3] должны работать в режиме вывода, а линии PORTB[4]...PORTB[7] – в режиме ввода. Восьмиразрядная шина данных интерфейса индикатора подключена к выводам порта PORT P. Так же, как и в примере параграфа 5.6.2, мы будем производить только операции записи в контроллер управления индикатором. Поэтому порт PORT P будет постоянно работать в режиме вывода. Линии управления индикатором подключены к выводам PORTDLC2: PORTDLC3.

Можно убедиться, что рассмотренные ранее по отдельности схемы подключения клавиатуры и ЖК индикатора полностью повторены, вплоть до конкретных выводов портов. Поэтому мы сможем воспользоваться ранее приведенными программами управления без каких-либо изменений.

5.8.2. Программа управления

На рис. 5.23 представлена блок-схема алгоритма управления кодовым замком. Ниже приведен полный текст программного кода программы управления, составленный по этой блок-схеме алгоритма.

```
/*--------------------------------------------------------------------*/
/* filename: lock.c - программа управления электронной системой доступа */
/* Число символов кода доступа равно 4. Правильная кодовая комбинация:C963*/
/* Аппаратная конфигурация:                                           */
/* PORTB - 8 линий интерфейса клавиатуры                              */
/*PORTP - 8 линий шины данных интерфейса ЖК индикатора                */
/*PORTDLC2 -RD/WR, PORTDLC3 - E.                                      */
/*--------------------------------------------------------------------*/
/*подключаемые файлы*/
#include<912b32.h>
#include <stdio.h>
#include <math.h>

/*используемые функции*/
char which_key(unsigned int keypress);  //определение, какая клавиша
                                        //нажата
void delay_100us(void);                 //задержка 100 мкс
void delay_5ms(void);                   //задержка 5 мс
void initialize_lcd(void);              //инициализация контроллера ЖК
                                        //индикатора
void initialize_key(void);              //инициализация портов для
                                        //клавиатуры
void putchars(unsigned char c);  //запись одного символа в индикатор
void putcommands(unsigned char d); //запись одной команды в индикатор
void lcd_print(char *string);    //запись в индикатор строки символов
char keypad(unsigned int keypress, int row);
```

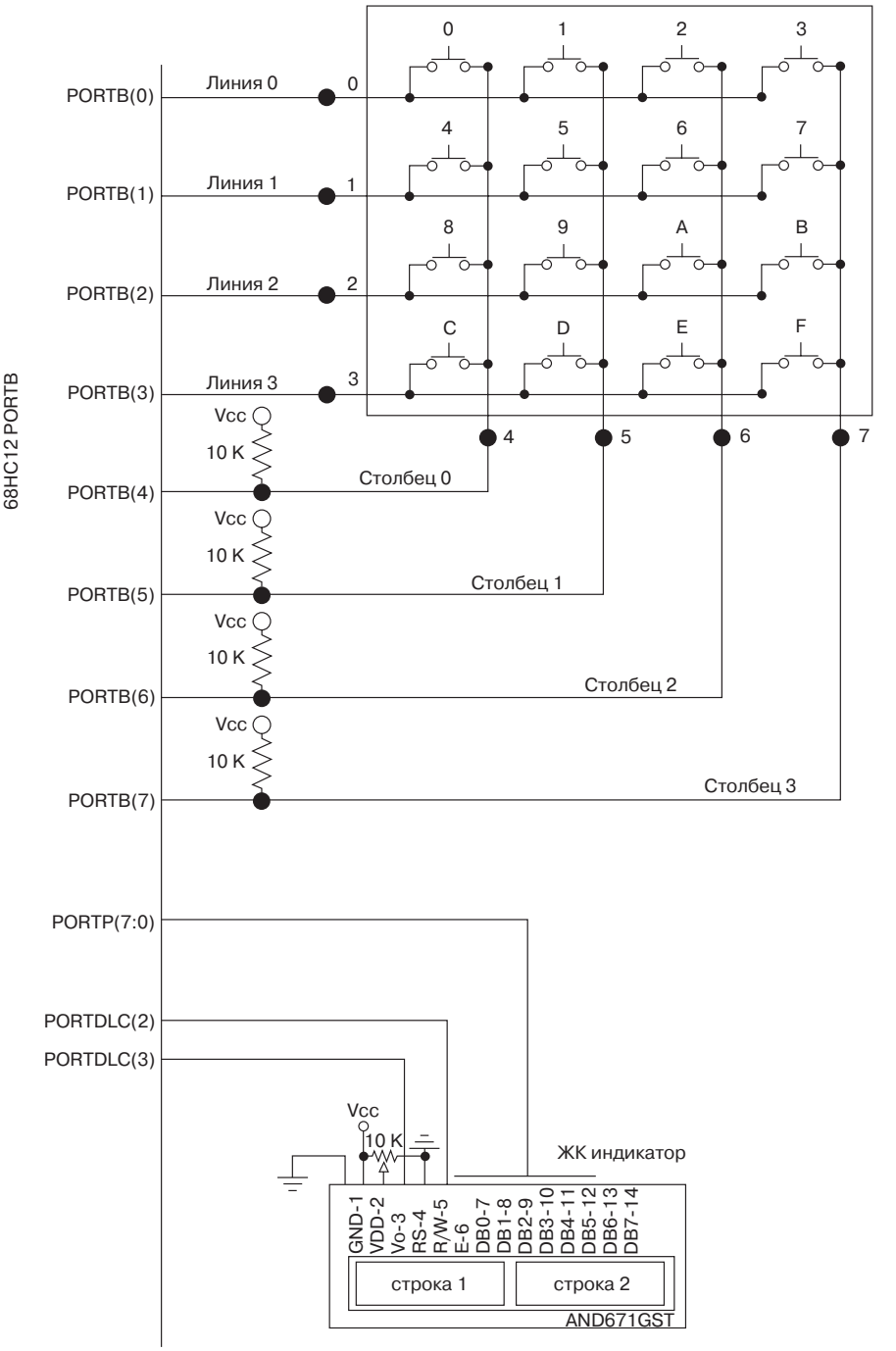


Рис. 5.22. *Функциональная схема аппаратных средств для системы кодового замка*

```
void main(void)
{
int first = 0x01;          //инициализация служебных переменных для опроса
int second = 0x02;         //клавиатуры
int third = 0x04;
int fourth = 0x08;
```

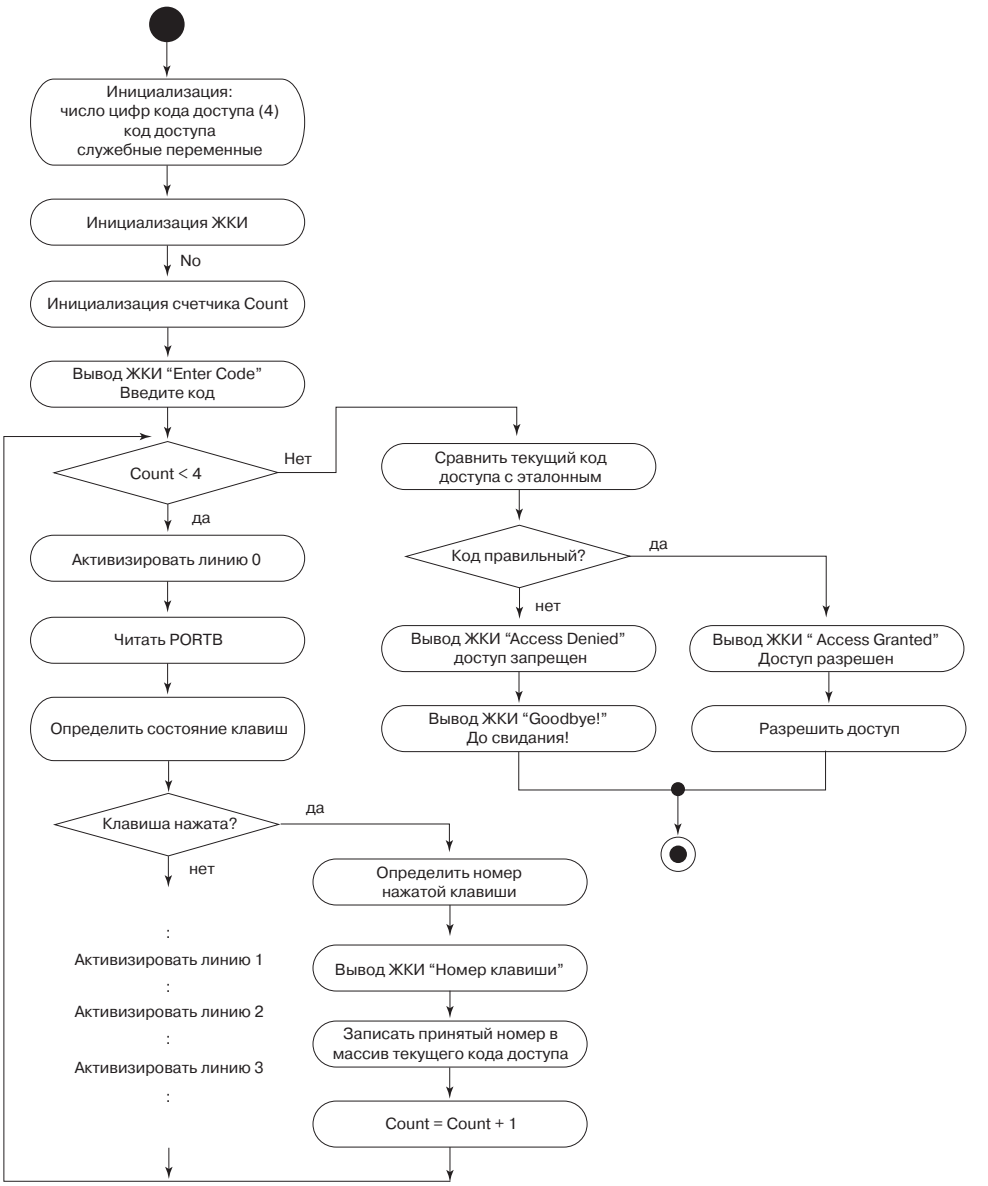


Рис. 5.23. *Блок-схема алгоритма управления кодовым замком*

```
int i,j,k, count = 0;
unsigned int keypress;
char key;
int length=4;

char pin[] = {'C', '9', '6', '3'}; //задание эталонной последовательности
                                   //символов
char code[4];                      //массив для введенной пользователем
                                   //последовательностисимволов
initialize_lcd() ;                 //инициализация ЖК индикатора
initialize_key();                  //инициализация портов для клавиатуры

for(k=0;k<4;k++)                   //разрешает 4 раза ввести комбинацию из
{                                  //4 символов
count = 0;
putcommands(0x01) ;
lcd_print ("Enter Code:");

while(count < length)              //обработка 4 нажатий клавиатуры
      {
      for(i=0; i<=50; i++)         //ожидание ввода символа
            {
            PORTB = 0x01;          //опрос первого ряда клавиатуры
            keypress = PORTB;
            key = keypad(keypress, first);

            if (key ! = 'Z')       //если клавиша нажата
            {
            printf("%c", key);
            code[count] = key;     //записать очередной символ в
                                   //массив ввода
            count++;               //перейти к сканированию следующего
                                   //символа
            }
      }

for(i=0; i<= 50; i++)              // ожидание ввода символа
      {
      PORTB = 0x02;                //опрос второго ряда клавиатуры
      keypress = PORTB;
      key = keypad(keypress, second);

if(key != 'Z')
{
      printf ( "%c", key);
      code[count] = key;           //записать очередной символ в
                                   //массив ввода
      count++;                     //перейти к сканированию следующего
                                   //символа
                  }
      }

for(i=0; i<= 50; i++)              //ожидание ввода символа
      {
      PORTB = 0x04;                //опрос третьего ряда клавиатуры
```

```
        keypress = PORTB;
        key = keypad(keypress, third);

        if (key ! = 'Z')
                {
                printf("%c", key);
                code[count] = key;  //записать очередной символ в
                                    //массив ввода
                count++;            //перейти к сканированию следующего
                                    //символа
                }
        }

for(i=0; i<= 50; i++)               //ожидание ввода символа
        {
        PORTB = 0x08;               //опрос четвертого ряда клавиатуры
        keypress = PORTB;
        key = keypad(keypress, fourth);

        if (key ! = 'Z')
                {
                printf("%c", key);
                code[count] = key;  //записать очередной символ в
                                    //массив ввода
                count++;            //перейти к сканированию следующего
                                    //символа
                }
        }
}       //очередные 4 символа введены, можно сверять с эталонными


/*Проверка соответствия кода на эталонный*/

j = 0;
for(i=0;i<3;++i)            //проверка по каждому символу эталонной
                            //последовательности
        {
        if(pin[i]= =code[i])        //очередной символ правильный
                {
                j++;
                }
        else                        //очередной символ неправильный
                {
                j--;
                }
        }

if(j= =(length - 1))        //если j=3, то все символы введены правильно
        {
        putcornrnands(0x01);
        1cd_print ( "Access Granted");
        }
else                                //символы введены неправильно
        {
        putcornrnands(0x01);
```

```
       lcd_print ( "Access Denied");
       }
       }
putcornrnands(0x01) ;
lcd_print ( " Goodbye ! " );     //конец попыток ввода
}
/*-------------------------------------------------------------------------*/
/*Перечень функций, которые использованы в этой программе                  */
/*-------------------------------------------------------------------------*/
/* Функции, программный код которых был приведен ранее по тексту главы*/
char which_key(unsigned int keypress)
void delay_5rns(void)
void delay_l00us(void)
void initialize_lcd(void)
void putchars(unsigned char c)
void putcornrnands(unsigned char d)
void lcd-print(char *string)
function body is provided char keypad(unsigned int keypress, int row)

//Функция, которая приведена ниже
void initialize_key(void)

/*-------------------------------------------------------------------------*/
/* Функция initialize_key устанавливает направление передачи линий портов */
/* для обслуживания клавиатуры                                             */
/*-------------------------------------------------------------------------*/
void initialize_key(void)
{
DDRB= 0x0F;          //линии PORTB[0]…PORTB[3] - на вывод
                     //линии PORTB[4]…PORTB[7] - на ввод
PORTB = 0x00;        //четыре младших линии порта PORTB в 0
}
/*-------------------------------------------------------------------------*/
```

5.9. Интерфейс МК с аналоговыми датчиками

Датчик – это устройство, которое преобразует некоторую физическую величину в электрический сигнал. Существуют датчики температуры, давления, ускорения, напряженности электрического поля, механического момента, и т.д. В главе 2 мы рассмотрели пример метеостанции, в которой выходы датчиков, поставляющих информацию о погоде, были подключены к МК семейства 68HC12. В том примере мы по умолчанию предположили, что датчики допускают непосредственное подключение ко входам МК. Однако на практике это условие почти всегда не выполняется. Поэтому необходимо разрабатывать дополнительную схему сопряжения, которая, в общем случае, состоит из усилителя со смещением уровня и фильтра. Подробное изучение способов преобразования аналоговых сигналов выходит за рамки данного раздела. Читатель может познакомиться с ними по книгам, приведенным в параграфе дополнительной литературы для данной главы. Мы остановимся только на общем рассмотрении масштабирования сигнала с выхода датчика.

На рис. 5.24 приведена обобщенная функциональная схема интерфейса для

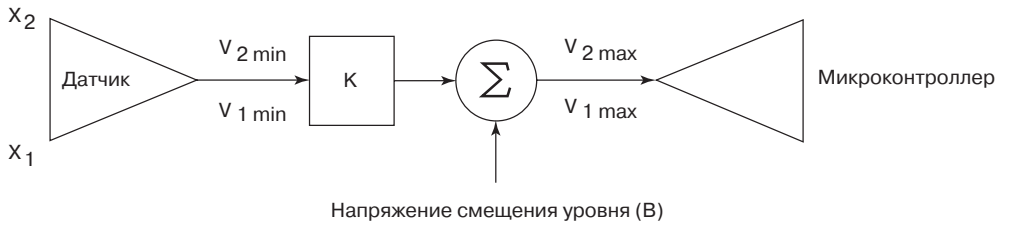


Рис. 5.24. *Обобщенная функциональная схема интерфейса для подключения аналогового датчика ко входу встроенного АЦП МК*

подключения аналогового датчика ко входу встроенного АЦП микроконтроллера. Датчик преобразует некоторую величину X в напряжение на выходе электронного преобразователя датчика. В последующих рассуждениях мы предположим, что электронный преобразователь имеет линейную передаточную характеристику. Если на входе присутствует некоторая физическая величина X_1, то на выходе электронного преобразователя формируется напряжение V_{1min}. При этом величина X_1 соответствует минимальному значению, которое может преобразовать рассматриваемый датчик. Если на входе датчика присутствует величина X_2, то на выходе электронного преобразователя формируется напряжение V_{2min}. И X_2 соответствует максимальному значению датчика. Все промежуточные значения величины X могут быть восстановлены по величине напряжения на выходе преобразователя $V_{1min} < V < V_{2min}$.

Для того чтобы значение напряжения на выходе датчика могло быть использовано в управляющей программе, его необходимо преобразовать в цифровой код при помощи модуля АЦП. Преобразование будет выполнено с максимально возможным разрешением, если V_{1min} будет равно 0 В, а V_{2min} – 5,0 В. Поэтому между выходом датчика и входом АЦП следует установить дополнительный электронные усилитель со смещением уровня (рис. 5.24). Обозначим коэффициент усиления этого усилителя K, напряжение смещения – B. Для определения численных значений K и B составим два уравнения:

$$V_{2max} = V_{2min} \times K + B$$

$$V_{1max} = V_{1min} \times K + B,$$

где V_{2max} и V_{1max} напряжения верхнего и нижнего уровня шкалы на выходе усилителя со смещением. Решив уравнения, можно получить численные значения коэффициента усиления и напряжения смещения, а затем реализовать необходимую схему на операционных усилителях.

Пример. Представьте себе, что Вам необходимо подключить барометр метеостанции к микроконтроллеру 68HC12 для автоматизированного учета показаний прибора. Диапазон допустимых напряжений сигнала АЦП микроконтроллера составляет 0...5,0 В. Минимальная величина показаний барометра равна 64 см ртутного столба, максимальная – 81 см. Барометр снабжен электронной схемой преобразования сиг-

нала, которая формирует на выходе для показания 64 см напряжение –100 мВ, а для показания 81 см – напряжение +300 мВ. Передаточная характеристика этой электронной схемы преобразования линейная. Необходимо определить параметры дополнительного электронного преобразователя для сопряжения предоставленного барометра с МК семейства 68HC12.

В соответствие с ранее введенными обозначениями, можно записать: $V_{1min} = -100$ мВ, $V_{2min} = 300$ мВ, $V_{1max} = 0$ В, $V_{2max} = 5{,}0$ В. Подставим эти численные значения в уравнения для определения коэффициента усиления и напряжения смещения дополнительного электронного преобразователя сигнала:

$$5 \text{ В} = 300 \text{ мВ} \times K + B$$

$$0 \text{ В} = -100 \text{ мВ} \times K + B$$

Решив два уравнения совместно, получим, что коэффициент усиления K = 12,5, напряжение смещения B = 1,25 В. Проверьте полученный результат, подставив полученные значения в исходные уравнения. Составьте схему на операционном усилителе, реализующую необходимые преобразования.

Применяя рассмотренный метод преобразования входного сигнала, помните, что реальная схема сопряжения может потребовать установки дополнительных фильтров с целью подавления нежелательных гармоник сигнала датчика.

5.10. Интерфейс RS-232

В главе 4 мы рассмотрели периферийные модули МК семейства 68HC12/HCS12, в том числе контроллеры последовательного обмена. Напомним, что МК 68HC12/HCS12 имеют в своем составе, как минимум, один контроллер асинхронного последовательного обмена SCI и один контроллер синхронного последовательного обмена SPI. Каждый из этих контроллеров формирует на выходе логические сигналы с напряжением около 5 В для логической 1 и около 0 В для логического 0. Однако если систему с МК необходимо соединить с другим устройством посредством интерфейса RS-232, то обмен с использованием логических уровней сигналов уже невозможен, и необходимо дополнительное согласование уровней.

Стандарт EIA-232-D устанавливает правила организации последовательного обмена данными для интерфейса RS-232 (EIA – Electronic Industries Alliance). Стандарт определяет число линий связи и их функциональное назначение, электрические характеристики сигналов в линиях, формат кадра обмена и механические соединители.

Переход от логических уровней сигналов к сигналам стандарта RS-232 может быть выполнен с использованием всего одной ИС трансивера RS-232. На рис. 5.25 приведен пример такой ИС компании MAXIM-IC. Трансивер позволяет выполнить преобразования двух сигналов с логическими уровнями TTL/CMOS к уровням сигналов интерфейса RS-232 и наоборот. Логическая 1 преобразуется этой ИС в отрицательное напряжение –10 В, логический 0 – в положительное напряжение +10 В. И при этом ИС требует всего одного источника питания для своей работы с напряжением 5,0 В ± 10%.

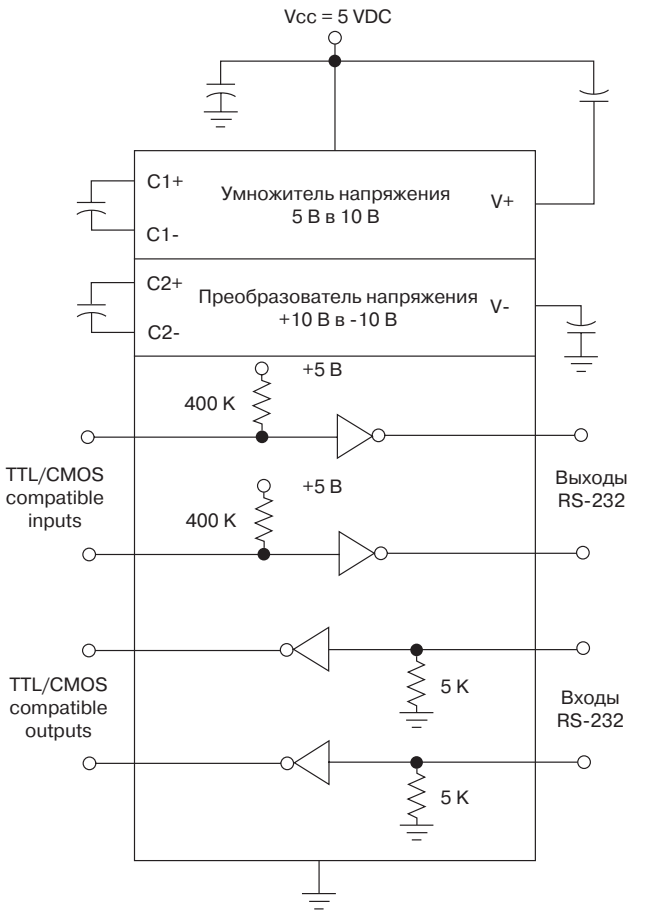


Рис. 5.25. *Функциональная схема трансивера RS-232*

5.11. Заключение по главе 5

Мы начали эту главу с анализа электрических характеристик входов и выходов микроконтроллеров и других ИС, выполненных по технологии HC CMOS. Это позволило нам понять, каким требованиям должны удовлетворять входные и выходные устройства, которые подключаются к выводам МК. Далее мы изучили конкретные типы устройств ввода и вывода, такие, как светодиоды, переключатели, клавиатуры, различные типы ЖК индикаторов. Мы рассмотрели особенности реальных устройств ввода/вывода: эффект дребезга механических переключателей, генерацию помех при работе электрического двигателя с силовым полупроводниковым коммутатором. Мы также показали, как преобразовать логические сигналы МК к уровням сигналов интерфейса RS-232. Все эти знания будут использованы нами далее в главе 6 в процессе изучения реальных встраиваемых систем на микроконтроллерах семейства 68HC12/HCS12.

5.12. Что еще почитать?

1. Horowitz, P., W. HiП. The Art o! Electronics, 2nd ed. Cambridge, England: Cambridge University Press, 1989.
2. Furlow, Bill. Circuit Design Idea Handbook. Boston: Cahners, 1975.
3. Sheingold, Daniel H., ed. Analog-Digital Conversion Handbook. Norwood, MA: Analog Devices, 1976.
4. «Transducers,» Omega Engineering Inc., 1 Omega Drive, Stamford, CT, 06907
5. Stout, David F., Milton Kaufman. Handbook of Operational Amplifier Circuit Design.
   New York: McGraw-Hi1l, 1976.
6. Hollander, M. A. Electrical signals and systems. New York: McGraw-Hi1l.

5.13. Вопросы и задания

Основные

1. Что означают буквы «HC» в названии семейства микроконтроллеров 68HC12?
2. Назовите и определите физический смысл 8 электрических параметров для входов и выходов серии логических элементов. Включите в свой ответ диаграммы входных и нагрузочных характеристик.
3. Назовите 8 электрических параметров, характеризующих входы и выходы МК семейства 68HC12. Приведите диаграммы, которые связывают эти параметры.
4. Какова максимальная нагрузка по току входов и выходов МК семейства 68HC12?
5. Что произойдет с V_{OH} и V_{OL}, если будет превышена нагрузка по току?
6. Какие два условия должны быть выполнены, чтобы светодиод светился?
7. Почему схема сопряжения со светодиодом, представленная на рис. 5.3, не будет работать? Предложите рабочий вариант схемы при условии, что прямой ток светодиода 15 мА, а прямое падение напряжения 1,7 В?
8. Опишите два способы защиты от дребезга механических контактов.
9. Что такое выход с тремя состояниями?
10. Как работает твердотельное реле? В каких случаях его следует использовать?

Более сложные

1. В тексте главы были введены две виртуальные серии логических элементов (DP1 и SB2). Для этих серий были приведены электрические параметры для входов и выходов:

DP1:

$$V_{IH} = 2{,}0\ \text{В},\ V_{IL} = 0{,}8\ \text{В}.\ I_{OH} = -0{,}4\ \text{мА},\ I_{OL} = 16\ \text{мА}$$

$$V_{OH} = 3{,}4\ \text{В},\ V_{OL} = 0{,}2\ \text{В},\ I_{IH} = 40\ \text{мкА},\ I_{IL} = -1{,}6\ \text{мА}$$

SB2:

$$V_{IH} = 2{,}0\ \text{В},\ V_{IL} = 0{,}8\ \text{В}.\ I_{OH} = -0{,}4\ \text{мА},\ I_{OL} = 8\ \text{мА}$$

$$V_{OH} = 2{,}7\ \text{В},\ V_{OL} = 0{,}4\ \text{В},\ I_{IH} = 20\ \text{мкА},\ I_{IL} = -0{,}4\ \text{мА}$$

Определите, могут ли логические элементы серии DP1 быть подключены к элементам серии SB2? Если да, то чему равен коэффициент разветвления SB2 для DP1?

2. Можно ли подключать элементы серии DP1 к выходам МК семейства 68HC12?
3. Можно ли подключать элементы серии SB2 к выходам МК семейства 68HC12?
4. На рис. 5.5 подтягивающие резисторы подключены к напряжению питания V_{CC}. Представьте, что схему изменили, и эти же резисторы подключили к общему выводу. Как должна измениться таблица рис. 5.5?
5. На рис. 5.5 была приведена схема подключения к МК клавиатуры из 16 клавиш. Разработайте аппаратную схему декодирования кода клавиш подобной клавиатуры. На выходе схемы должен формироваться ASCII код нажатой клавиши. МК не должен использоваться для решения поставленной задачи.
6. Опишите своими словами принцип действия схем противодребезговой защиты, приведенных на рис. 5.10.
7. Разработайте блок-схемы алгоритмов для каждой из функций программы управления ЖК дисплеем из разделов 5.6.2 и 5.6.3.
8. Разработайте схему сопряжения МК с твердотельным реле (рис. 5.19), считая, что прямое падение напряжения на светодиоде равно 1,7 В, прямой ток светодиода равен 20 мА.
9. Опишите, как работает МДП-транзистор.
10. Используя дополнительные источники и Интернет (www.irf.com), составьте реферат о параметрах современных мощных МДП-транзисторов.
11. Некоторый датчик формирует на выходе в нижней точке шкалы измерения напряжение 30 мВ, в верхней точке шкалы – 500 мВ. Разработайте интерфейс сопряжения этого датчика с МК семейства 68HC12.
12. Повторите задание предыдущего вопроса, но при условии, что в нижней точке шкалы измерения напряжение равно 500 мВ, а в верхней точке шкалы выходное напряжение составляет –30 мВ.

Исследовательские

1. На рис. 5.5 подтягивающие резисторы подключены к напряжению питания V_{CC}. Представьте, что схему изменили, и эти же резисторы подключили к общему выводу. Разработайте программу формирования кода нажатой клавиши для такой схемы подключения клавиатуры.
2. Разработайте алгоритм и блок-схему программы для аппаратного подключения переключателей и светодиодов рис. 5.11. Зеленый светодиод на выходе PC0 должен светиться, если нажата кнопка на входе PB0. На остальных выводах порта PC0 должны гореть красные светодиоды. Если нажата кнопка на

входе PB1, должны светиться зеленые светодиоды на выходах PC0 и PC1. И так далее, для всех кнопок, заканчивая PB7, при нажатии которой должны загореться 8 зеленых светодиодов.

3. Измените текст программы параграфа 5.8.2, разрешив доступ по комбинации из шести правильных символов.

Глава 6

ДОБРО ПОЖАЛОВАТЬ В РЕАЛЬНЫЙ МИР!

ЦЕЛИ

ПОСЛЕ ИЗУЧЕНИЯ ГЛАВЫ ВЫ СМОЖЕТЕ:

- Определить реальные ограничения проекта, которые могут не позволить микроконтроллерной системе правильно работать.
- Изложить правила обращения с устройствами на базе КМОП и разрабатать рекомендации.
- Определить источники и внутренних и внешних помех для микроконтроллерной системы.
- Перечислить основные организации, ответственные за обеспечение директив и руководств по электромагнитной совместимости (ЭМС).
- Рассказать о методах проектирования, позволяющих минимизировать чувствительность к помехам.
- Применить программные методы защиты, чтобы минимизировать чувствительность к помехам.
- Описать методы обнаружения помех.
- Применить методы управления питанием, позволяющие снизить мощность, потребляемую микроконтроллерной системой.
- Понять изменения, определяемые выбором батарейного источника питания для микропроцессорной системы.
- Определить оснвные свойства супервизоров для управления микропроцессорами.
- Определить и реализовать меры энергосбережения.

6.1. Ужасные истории об ошибках проектирования....386
6.2. Правила обращения с микросхемой 68HC12 и рекомендации по проектированию 391
6.3. Исследование помех....393
6.4. Защитное программирование 399
6.5. Методики испытаний на наличие помех....401
6.6. Управление энергопотреблением....403
6.7. Заключение по главе 6 408
6.8. Что еще прочитать? 408
6.9. Вопросы и задания....409

Что можно сказать обо всей этой главе? После того, как вы рассмотрели цели главы, вы могли бы подумать, что глава представляет собой совокупность собранных с бору по сосенке разделов, которые неудобно размещать в любой другой главе. Мы согласны с тем, что в главе рассматривается много различных тем; однако, все они связаны общей нитью. Если при разработке реальных устройств проигнорировать любую из рассматриваемых здесь проблем, система вообще не сможет функционировать, Хуже все то, это поведение схемы при этом может носить случайный, непредсказуемый, или невоспроизводимый характер. В этой главе мы имеем дело с реальными проблемами разработки, которые необходимо решать. Мы приводим всесторонний список исходных материалов для дальнейших разделов, в которых читатель может разыскать дополнительную информацию.

6.1. Ужасные истории об ошибках проектирования

Мы начнем с нескольких «ужасных историй» о проектах, которые работали неправильно. Обратите особое внимание на специфические причины этих сбоев. После этого мы покажем Вам некоторые методы, которые позволяют превратить хороший проект, созданный на бумаге, в нормально работающее практическое устройство.

6.1.1. Случай квадратичного генератора

Прежде, чем обсудить первый случай, приведем некоторую основную информацию о восстановлении сигнала. Полезная методика создания аналогового сигнала заданной формы, состоит в том, чтобы разделить сигнал на ряд аналоговых данных. Аналоговые значения в отдельных точках преобразуются затем в двоичные коды от $00 до $FF. При этом минимальный сигнал в 0 В соответствует числу $00, а максимальный сигнал, например, в 5 В соответствует числу $FF. Аналоговые данные, расположенные между этими двумя экстремальными значениями преобразуются по линейному закону в 8-разрядный двоичный код. Не правда ли эта методика что-то вам напоминает? Конечно, это процесс аналого-цифрового преобразования (АЦП), который мы уже обсуждали в главе 4.

Чтобы восстановить аналоговый сигнал, отдельные двоичные коды последовательно пересылаются на цифро-аналоговый преобразователь (ЦАП). Перед тем, как продолжить рассмотрение, проведем короткий обзор основных концепций для ЦАП.

Цифро-аналоговый преобразователь. Цифро-аналоговый преобразователь (ЦАП) преобразует входной многоразрядный двоичный код в соответствующий аналоговый выходной сигнал (см. рис. 6.1). Мы не будем обсуждать различные методы преобразования, применяемые в ЦАП. Мы просто представим ЦАП как «черный ящик» и опишем его функции с помощью блок-схемы. Аналоговый выходной сигнал создается суммированием взвешенных двоичных входных сигналов, как показано на рисунке.

Например, когда на входе ЦАП формируется код $CA, старший двоичный бит умножается на половину значения U_{MAX}. Следующий по старшинству бит умножается на четвертую часть U_{MAX} и так далее. Все взвешенные биты суммируются, давая в результате аналоговое значение.

Примем, что $F_S = V_{RH} - V_{RL}$ представляет собой максимальный размах напряжения для 8-разрядного 5-вольтового ЦАП. В этом случае высокий уровень опорными напряжениями составляет 5 В, а низкий – с 0 В. А как определить, например, аналоговое выходное напряжение соответствуюшее двоичному входному коду $CA?

Это напряжение можно определить, используя следующую взвешенную сумму для входного кода $CA:

$$U_{ВЫХ} = 5\,[(1\times1/2) + (1\times1/4) + (0\times1/8) + (0\times1/16) + (1\times1/32) + (0\times1/64) + (1\times1/128) + (0\times1/256)] = 3,945\ В$$

Аналоговый сигнал может быть затем создан путем цифровых данных на входах ЦАП, чтобы получить аналоговую реконструкцию. Методика восстановления аналогового сигнала из отдельных двоичных точек может использоваться при многих различных типах сигналов [Barrett 1979, Welch 1997].

Обсудим теперь сбой в разрабатываемом квадратичном генераторе. На рис. 6.2 показано, как эта методика может использоваться, чтобы генерировать сигналы, связанные квадратичной зависимостью (напряжения синусоидальной и косинусои-

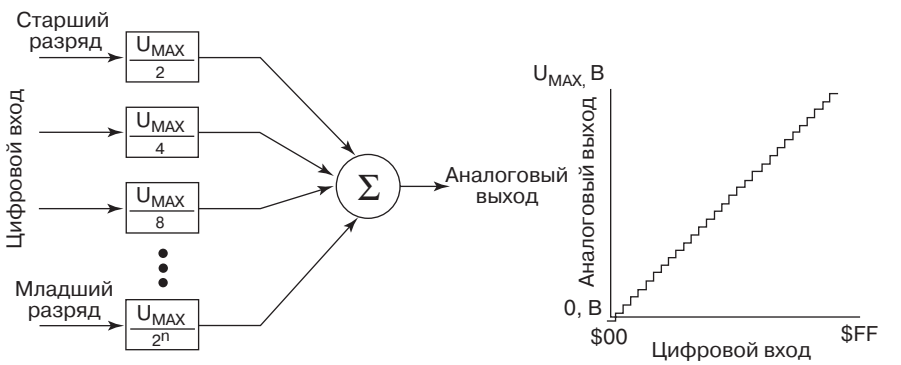


Рис.6.1. *Цифро-аналоговый преобразователь*

Цифро-аналоговый преобразователь (ЦАП) преобразует многоразрядный двоичный входной код в соответствующий аналоговый выходной сигнал. Аналоговый выходной сигнал создается суммированием взвешенных двоичных входных сигналов. Символ UMAX соответствует максимальному напряжению

дальной формы). Схема включает в себя программируемый кварцевоый CMOS генератор PXO1000 фирмы Statek. Частота генератора установливается вручную с помощью DIP переключателей. Генератор является таймером для 8-разрядного двоичного счетчика нва базе CMOS, который обеспечивает непрерывный счет от $ 00 до $FF. Выходной сишнал этого счетчика подается на две отдельных NMOS микросхемы памяти EPROM размером 2048 б, которые сохраняют данные для синусоидальной и косинусоидальной волны. Выходные сигналы с микросхем памяти подаются на отдельные 8-разрядные CMOS цифро-аналоговые преобразователи, которые опять преобразуют отдельные двоичные данные в аналоговый сигнал. Сигналы с ЦАП подаются на операционные усилители, которые осуществляют сдвиг нулевого уровня выходного сигнала ЦАП, и могут также обеспечить подстройку коэффициента усиления.

На бумаге проект работает вполне правильно. Однако, когда схема была реализована, оказалось, что она является хорошим генератором помех. В чем была проблема? В первоначальном проекте отсутствовали резисторы сопротивлением 4.7 К в выходных цепях NMOS памяти. В главе 5 мы указали, что мы можем обычно напрямую связывать входы и выходы микросхем одного семейства. Однако, при соединении микросхем из различных семейств, мы должны быть очень осторожны. Обратите внимание, что в этом проекте, мы переходим от счетчика CMOS к памяти NMOS и затем обратно на ЦАП CMOS. Мы должны тщательно спроектировать интерфейс между CMOS и NMOS и затем интерфейс NMOS к CMOS. Чтобы эта схема правильно работала, должны быть включены нагрузочные резисторы в выводы NMOS. Это позволяет входам ЦАП на базе CMOS правильно распознавать состояния выходов NMOS. Нагрузочные резисторы показаны на рис. 6.2.

Из этого примера вы можете видеть важность тщательного выбора цепей связи между всеми компонентами внутри схемы. Особую осторожность следует проявлять при соединении микросхем из различных семейств.

6.1.2. Случай таймера для лазерного излучения

Рассмотрим теперь экспериментальную установку для научно-исследовательской работы, которая касалась методов задания времени облучения для записывающего лазера. В эксперименте, лазер управлялся ТТЛ совместимым импульсом, формируемым ПК. При высоком значении импульса, лазерный затвор был открыт, при низком – закрыт. Цель экспериментов состояла в том, чтобы изучить влияние лазера на ткани глаза при глазной хирургии. В идеале, мы хотели бы привязать работу лазера к временной метке видеозаписи эксперимента. В коммерческом проекте задания времени было найдено и проверено оборудование, обеспечивающее оптимальную стоимость для малобюджетного проекта.

После рассмотрения имеющихся на рынке микросхем, были выбраны цифровые таймеры серии 74LSXX. Компоненты использовались, чтобы создать пятиразрядный счетчик, который управлял бы несколькими семисегментными дисплеями с помощью микросхем драйверов дисплея. Считывание с семисегментных дисплеев производилось оптически объективом фотокамеры, производящей запись эксперимента. Электроника счетчика / дисплея была установлена в пластмассовом корпусе, и окно было вырезано так, чтобы были видны показания дисплея, как это показано на рис. 6.3.

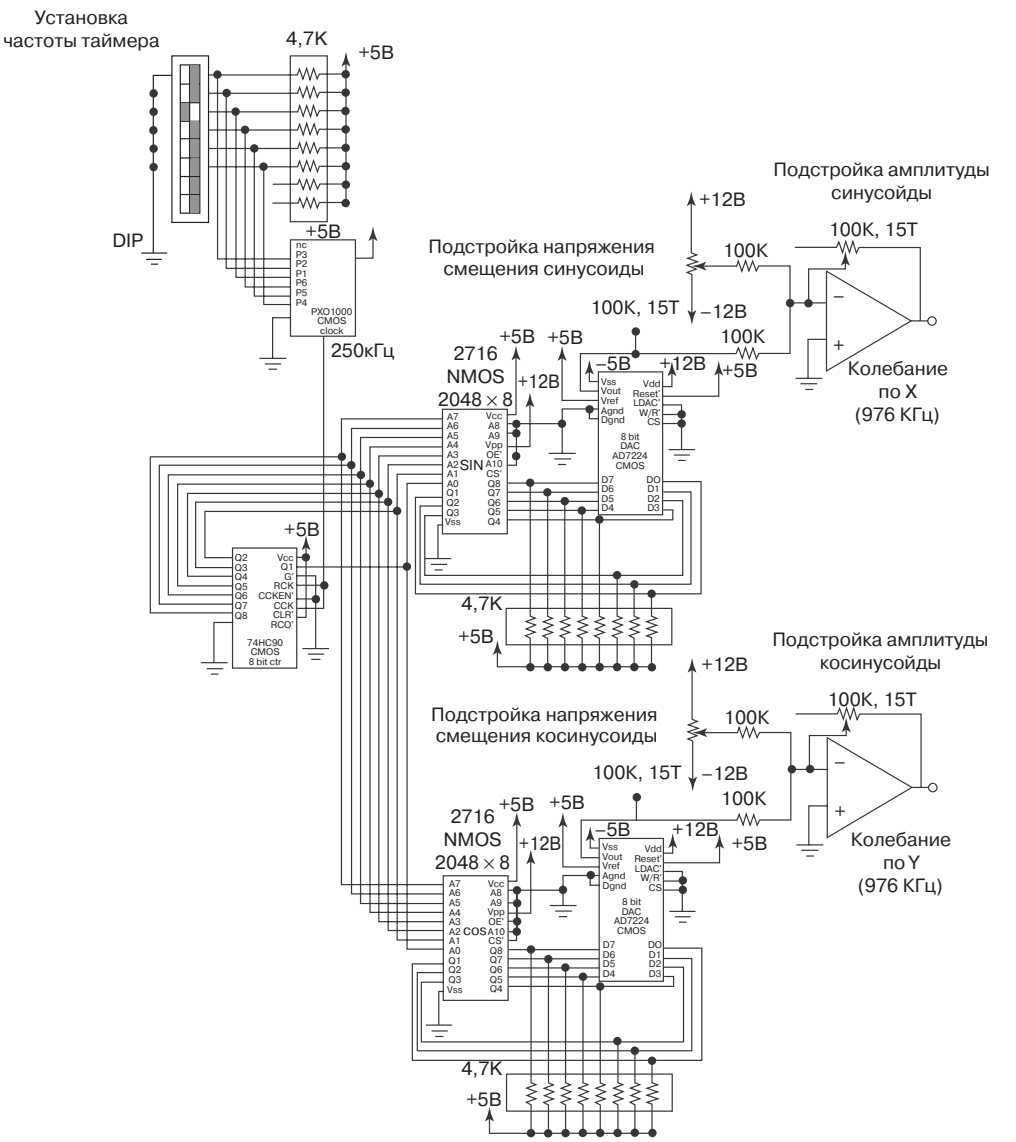


Рис. 6.2 *Квадратичный генератор*

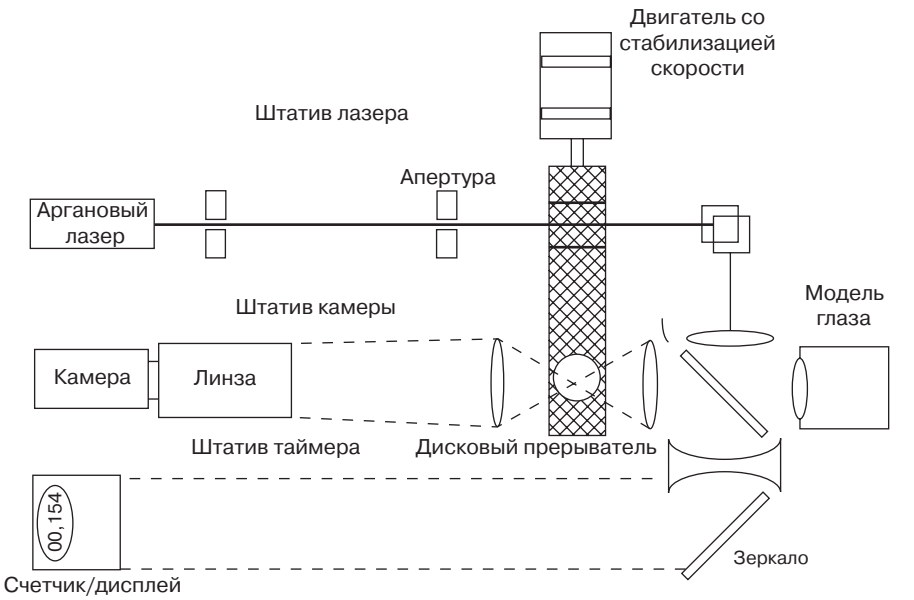


Рис. 6.3. *Дисковый прерыватель лазерного излучения поочередно фокусирует на исследуемой точке ткани сначала луч лазера, а затем объектив камеры. Поле обзора камеры, табло дисплея, и путь лазерного луча оптически съюстированы компанией*

Поле обзора камеры, табло дисплея, и путь лазерного луча были оптически съюстированы компанией. «Дисковый прерыватель» поочередно фокусирует на исследуемой точке ткани сначала луч лазера, а затем объектив камеры. Без такой синхронизации, высокая интенсивность лазера выводила бы видеокамеру в область насыщения. Окна дискового прерывателя были расположены друг против друга, чтобы чередовать пропускание лазерного луча и подсветки для камеры. Скорость двигателя, вращающего диск прерывателя, была стабилизирована таким образом, чтобы момент появления окна, открывающего объектив камеры был синхронизован со скоростью передачи кадров камеры. Это было небходимо для создания устойчивого изображения.

В теории, открытое состояние лазерного затвора, определялось не только импульсом от ПК, но также и импульсом синхронизации. Таким образом, затвор открывался при появлении обоих сигналов на входах схемы И, и счетчик определял бы время открытия затвора.

Схема счетчик/дисплей работала очень хорошо в лаборатории фирмы-производителя. Однако, когда она была перенесена в лазерную лабораторию, в работе появились сбои. Подозрение пало на помехи, и мы стали кропотливо изолировать источники помех. Мы не делали никаких попыток стандартной защиты схемы счетчик/дисплей от помех, пока не столкнулись с этой проблемой. Было достаточно трудно ввести эту защиту после того, как схема была уже создана. Как вы думаете, что за проблема могла бы здесь быть? Какие корректирующие меры должны быть приняты?

Зная, что двигатели являются известным источником помех, мы подозревали, что источником помехи был двигатель дискового прерывателя. Часто микроконтроллеры используются, чтобы управлять двигателями. В главе 5, мы показали, как оптически изолировать схему управления от двигателя. Однако, в этом примере,

схема счетчик/дисплей не имела никакого прямого подключения к электросети, питающей двигатель. Фактически, схема счетчик/дисплей и двигатель была включена от разных источников постоянного напряжения. Решение проблемы помех от этого типа двигателя была исследована более подробно. Мы использовали следующие методы, чтобы защитить схему счетчика/дисплея от помех, создаваемых двигателем:

- Сигнал управления с ПК на таймер был подан экранированным кабелем. Экран кабеля был заземлен. Кабель был проложен далеко от двигателя
- Помехоподавляющие (шунтирующие) конденсаторы были установлены на каждой микросхеме между источником и землей в схеме счетчика/дисплея. Кроме этого, помехоподавляющие конденсаторы были установлены на шинах питания платы счетчика/дисплея.
- Пластмассовый корпус счетчика/дисплея был обмотан заземленной медной лентой. Медный экран был также установлен на окне корпуса дисплея. Он был также электрически связан с медной лентой. Экран и лента обеспечили защиту корпуса схемы счетчика/дисплея.

После применения этой защиты от помех, схема стала работать в соответствии с проектом, и эксперименты были без труда завершены.

Какой урок можно извлечь из этого проекта? Не был проведен полный анализ среды. А он должен выполняться как нечто само собой разумеющееся, когда устанавливаются исходные требования для проекта. Как нечто само собой разумеющееся, должна быть также включена в любой проект, выполняемый на интегральных микросхемах, и защита от помех. Имеются много стандартных методов снижения помех, и внутренних и внешних по отношению к схеме. В заключение, оказывается, что чрезвычайно дорого вводить защиту от помех после того, как изделие было отправлено заказчику. В вышеупомянутом сценарии, проблема возникла внезапно, во время сбора экспериментальных данных.

Мы провели изучение двух случаев, чтобы иллюстрировать следующие общие проблемы в микропроцессорных систем:

- Неподходящие методы связи микросхем;
- Проблемы помех, связанные с внешними и внутренними источниками;
- Проблемы помех, связанные с аналоговой электроникой.

Теперь, расмотрев некоторые общие реальные проблемы, мы посвятим остальную часть главы описанию методов, которые действительно превращают проект на бумаге в проект, хорошо работающий на практике, позволяющий уйти от реальных ловушек.

6.2. Правила обращения с микросхемой 68HC12 и рекомендации по проектированию

Все справочные листы содержат значительный объем информации, которая зачастую игнорируется. И это, возможно, наиболее важная информация в данных, указывающая, как правильно обращаться с приборами на базе CMOS и рекомендации по проектированию устройств. Если вы разработчик, то вы не можете обычно участвовать в процессе изготовления. Но если вы студент старших курсов вуза, выпускающего разработчиков или вы связаны с изготовлением прототипов, вы должны запомнить эти рекомендации по установке.

6.2.1. Рекомендации по обращению со CMOS

Микросхемы CMOS семейства «HC» имеют чрезвычайно высокое входное сопротивление. Это происходит из-за изолированного затвора на входах устройства. Эти затворы могут быть повреждены при неверной установке микросхем. Хотя затворы CMOS имеют встроенные схемы защиты, были разработаны общие процедуры установки CMOS, чтобы минимизировать возможность повреждения. Эти процедуры установки основаны на предотвращении приложения большого статического напряжения к выводам затворов. Вот краткий обзор этих правил обращения:

- Носить заземленную полоску на запястье во время как установки устройств CMOS. Эти полоски можно легко приобрести у ряда электронных компаний;
- Хранить устройства CMOS в оригинальных контейнерах до использования. Эти контейнеры были разработаны, чтобы предотвратить повреждения статическим электричеством;
- Использовать устройства CMOS на заземленном месте для размещения тестирующих элементов;
- Исользовать при пайке только заземленные паяльники;
- Не вынимать, и не заменять устройства CMOS при включенной схеме;

Если эти правила обращения добросовестно исполняются, то схемы CMOS защищены от случайных повреждений. Кроме правил обращения, имеются также некоторые рекомендаций проектированию, котрым необходимо следовать.

6.2.2. Рекомендации по проектированию на CMOS

Рассмотрим некоторые рекомендации, позволяющие обеспечить грамотное проектирование интегральных микросхем на базе CMOS.

- Часто при проектировании встроенной системы, имеются несколько неиспользуемых вводов процессора. Вы не можете игнорировать эти вводы. Они должны быть правильно подсоединены к питающему напряжению процессора через резистор (4,7 кОм) или заземлены. Далее в этой главе, мы рассмотрим проблемы, возникающие при неправильном подсоединении этих выводов.
- Встроенная система на базе CMOS часто монтируется на печатной монтажной плате (PCB), связанноой с другими платами через соединители.

Когда внешние соединители PCB связаны непосредственно с входами или выходами устройства CMOS, должен использоваться добавочный резистор. Эти последовательно включенные резисторы минимизируют повреждения из-за статического электричества при стыковке и расстыковке соединителей PCB.

- Как мы видели в предыдущей главе, устройства CMOS должны использоваться только в границах нормированных электрических параметров. Если при разработке выйти за пределы этих спецификаций, схемы могут работать неправильно.

Вы, вероятно, чувствуете себя теперь достаточно подготовленными. Вы понимаете и можете применять методы проектирования CMOS, и знакомы с концепциями связи с помощью интерфейса из предыдущей главы. Однако вы можете следовать всем этим правильным рекомендациям, и, тем не менее, ваша система все-таки будет работать неправильно. Это приводит нас к следующему разделу, посвященному помехам системы и способам предотвращения и снижения их влияния.

6.3. Исследование помех

В этом разделе мы тщательно исследуем Немезиду проектировщика – помехи! Мы ответим на следующие вопросы: Что представляют собой помехи? Что является их источником? Какими удачными действиями при проектировании можно минимизировать чувствительность к помехам?

6.3.1. Что такое помехи

В самом простом смысле, помехой является любой нежелательный сигнал, не принадлежащий системе. Источники этих нежелательных сигналов могут быть внешними или внутренними по отношению к системе. Например, таймер на базе кристалла, необходим для системы процессора. Однако, когда сигнал таймера обнаруживается в других частях системы, где его не должно быть, он рассматривается как помеха. В любой схеме необходимо знать источник помех , потому что это знание поможет вам определить наилучший способ избавиться от них. На рис. 6.4 приведен краткий обзор источников помех.

- ***Разряд электростатического электричества (ESD)***: ESD может в основном быть определено как статическое электричество. Как уже упомянуто, в устройствах CMOS при воздействии статического электричества может быть поврежден затвор. Статическое электричество проявляется, когда два объекта с зарядами различных знаков входят в соприкосновение.
- ***Высокочастотные помехи (RFI)***: Эти помехи вызваны излучаемой энергией и могут быть созданы радио, сотовыми телефонами и т.д. Хотя они могли бы исходить от источника полезного сигнала, но если этот сигнал нежелателен для нашей системы, он все равно должен рассматриваться как помеха. Один из авторов описывает время, когда он жил в Омахе, штат Небраска, в течение ряда лет: «Мы использовали, чтобы получить некоторые довольно сильные «микрофоны грома» – ливни со значительным громом и молнией. Мы могли бы также сообщать, когда такой шторм приходил близко, потому что разряды молнии заставляли звонить наш дверной звонок. Первый раз, когда это случалось, мы были очень удивлены, «кто звонит в дверной звонок в 2 часа ночи в разгар такого большого ливня?». Однажды я выяснил, что это радиопомехи, созданные молнией, наводили напряжение в нашей схеме дверного звонка, и я использовал его как сигнал о приближении штормов. Кроме внешних помех, обратите внимание, что сам микроконтроллер может быть непосредственным источником помех. Частота таймера в процессоре 68HC12 составляет 8 МГц. Гармонический анализ тактовых импульсов показывает, что значительные гармоники существуют на десятой гармонике при частоте 80 МГц! Если печатная монтажная плата разработана неправильно, и не может минимизировать влияние радиопомех, эти частоты могут излучаться и восприниматься как помехи.
- ***Электромагнитные интерференционные (EMI)***; имеются две категории EMI: излучения и проводимости. Оба вида помех, вызываются электромеханическим оборудованием типа двигателей. В излучемых EMI, источник помех – не обязательно в контакте с системой. Фактически, этот тип EMI

часто классифицируется как радиопомехи. В EMI типа проводимости, помехи вызываются в проводнике, когда проводник проходит через электромагнитный поток, созданный источником помехи. Вспомним, что по закону Фарадея, напряжение вызывается в проводнике, когда он пересекает линии магнитного потока. Вы, вероятно, наблюдали эти явления при использовании электрической бритвы, мощной дрели или миксера. Если вы смотрите телевидение без кабеля связи, и один из этих приборов используется соседями, вы часто наблюдаете помехи на экране вашего телевизора.

- ***Просадки напряжения:*** просадки напряжения или «кратковременные провалы напряжения питания» вызваны уменьшением напряжения в сетях переменного тока. Они происходят, когда большая нагрузка подключается к сети переменного тока. Представьте себе жаркий, влажный день. Весь длинный день вы провели на работе, и ваша первая реакция, когда вы добираетесь до своего дома, включить воздушный кондиционер, чтобы охладить свое жилище. Если много людей делают это одновременно, это создает огромные перегрузки в сети, и может сопровождаться кратковременным провалом сетевого напряжения. Кратковременные провалы напряжения питания могут наносить ущерб незащищенным системам. Не забудьте, что устройства CMOS имеют очень специфические рабочие границы. Питающее напряжение определено в 5.0 В ±10 %. Кратковременный

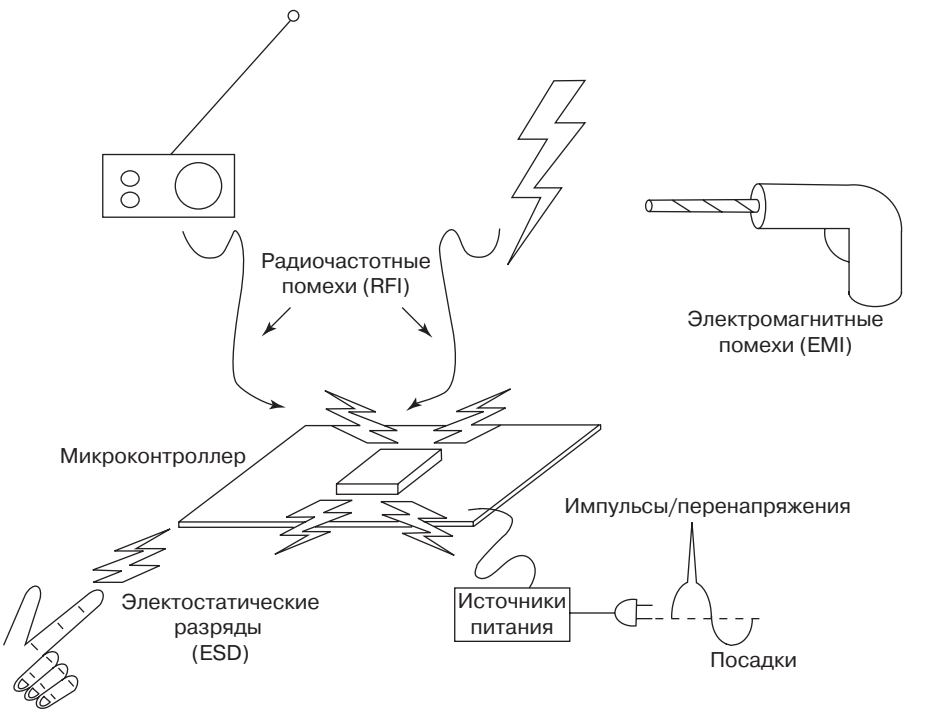


Рис. 6.4. *Источники помех*

провал напряжения питания может заставить питающее напряжение выйти за границы этого допуска. Когда это происходит, уровни логического нуля и единицы не выдерживаются. Защита от посадок напряжения обеспечивается управляющими схемами, обсужденными в разделе 6.6.5 этой главы.

- ***Импульсные перенапряжения:*** Импульсные перенапряжения подобны посадкам напряжения; однако, они – увеличивают питающее напряжение переменного тока. Волна импульсного перенапряжения может вызывать серьезное повреждение незащищенной системы. Защиту от перенапряжений можно обеспечить с помощью хорошего фильтра на удлинителе или на источнике питания.

6.3.2. Электромагнитная совместимость

Теперь, когда мы имеем хорошее определение того, что такое помеха, мы обнаруживаем, что обрели новую головную боль. Как проектировщики системы, мы должны защитить нашу встроенную микроконтроллерную систему от всех этих источников помех. Электромагнитная совместимость (ЭМС) – технический термин, относящийся к электромагнитному спектру. Если изделие испускает сигналы вне определенного спектрального диапазона, оно рассматривается как источник помех. Оно не должно генерировать сигналов, частота которых лежит вне допустимой области спектра. Кроме того, что касается излучения, мы также должны быть затронуть воросы чувствительности готового изделия к описаным выше внешним источникам помех. Мы должны рассматривать ЭМС с точки зрения излучения или чувствительности. Можно увеличивать степень ЭМС, снижая уровень излучения источника помех или чувствительность приемника.

Основываясь на изложенном, нетрудно понять, что ЭМС – это серьезная проблема при разработке изделия. Поэтому имеется ряд национальных и международных агентств, которые обеспечивают руководства и правила, касающиеся ЭМС. Мы приведем краткий обзор этих агентств и разработанных ими правил далее в этой главе.

В следующем разделе, мы начинаем исследовать, как систематически обеспечивать защиту от помех, и внешних и внутренних относительно схемы. Эта информация была собрана из ряда публикаций о применениях, изготовителей и из уроков, полученных из инженерной практики. Полные ссылки на использованную литературу приводятсятся в разделе «Что еще прочитать» в конце главы.

6.3.3. Спецификации системы помех – не будем крепки задним умом!

Для начала, мы возможно должны пересмотреть наши представления о помехах. Как мы видели в примерах, обсужденных ранее в этой главе, очень трудно обеспечить общую защиту от помех на уже законченном изделии. Мы видели, что электромагнитную совместимость необходимо рассматривать на стадии разработки технических требований для каждой встроенной системы управления. При разработке спецификаций системы, должна быть выполнена полная опись ожидаемых эксплуатационных режимов. Затем должны быть разработаны спецификации, охватывающие эти ожидаемые эксплуатационные режимы.

6.3.4 Методы снижения помех

В этом разделе мы приводим перечень методов снижения помех. Этот перечень основан на работе, проведенной Гленевинкелем (M. Glenewinkel) [1995] и дополненный информацией из ряда применений, который заслуживает внимания практикующих инженеров. Некоторые из этих методов иллюстрируются на рис. 6.5.

- ***Элементы для поверхностного монтажа:*** вообще менее восприимчивы к помехам, чем элементы с проводниками. Если вы недавно занимались созданием устройства на печатной плате, вы вероятно наблюдали, что компоненты для навесного монтажа стало труднее приобрести. Интегральные схемы, процессоры, резисторы, конденсаторы, и т.д. легче купить в корпусах, предназначенных для поверхностного монтажа.

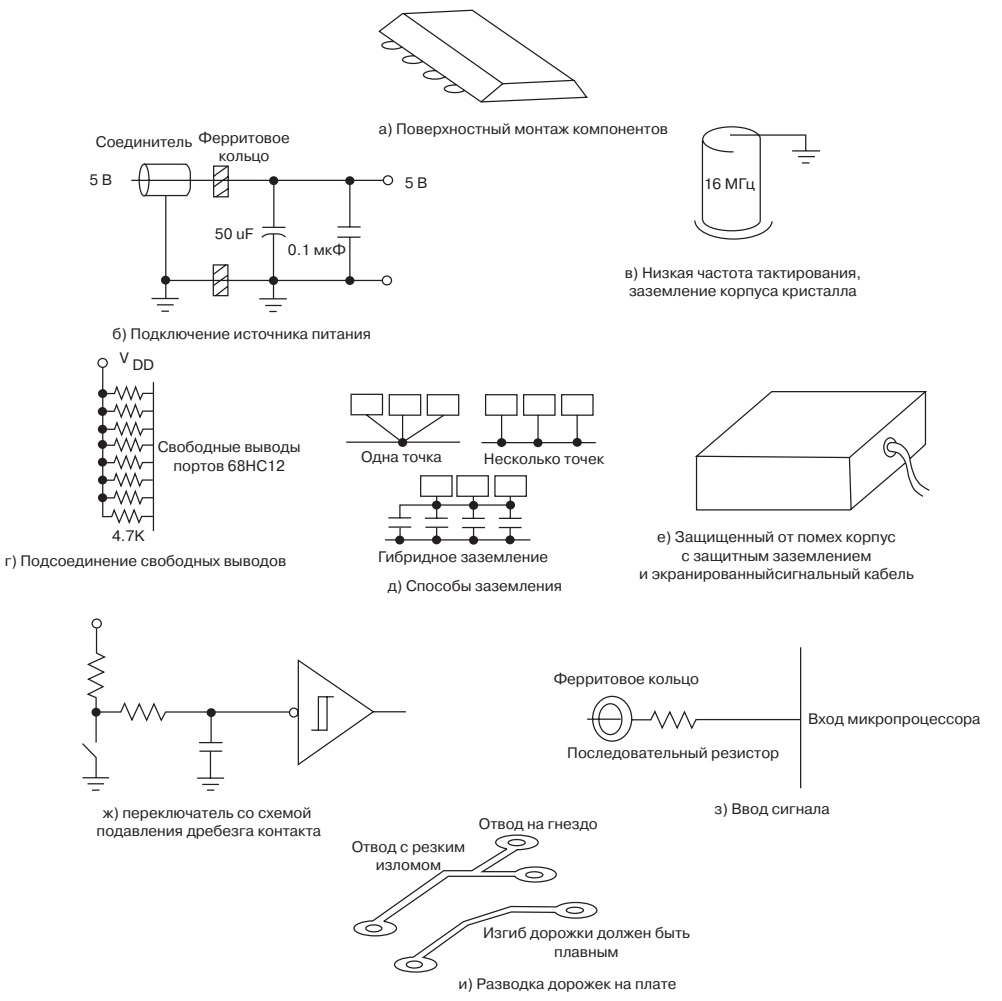


Рис. 6.5. *Методы снижения помех*

- ***Снижение помех от источников питания:*** Источники питания могут быть и источниками помех. Встроенные процессоры могут генерировать выбросы мощности из-за проходящих в них переходных процессов. Такой выброс походит на импульс, а импульс имеет значительные высокочастотные составляющие в разложении Фурье. Если переходные процессы в источнике питания неправильно развязаны с встроенной системой, могут возникать проблемы ЭМС. Как нечто само собой разумеющееся, вы должны включить развязывающие конденсаторы, чтобы минимизировать эти переходные процессы. Обычно конденсатор емкостью 0,1 мкФ используется для частот до 15МГц. Эти конденсаторы должны быть аксиальными стеклянными, многослойными керамическими. Конденсаторы в 0.01 мкФ должны использоваться для действующих частот больших, чем 15МГц. Эти конденсаторы должны быть подключены между вводами источника питания и земли для каждого корпуса интегральной схемы (ИС). Конденсаторы должны быть помещены как можно ближе к каждой ИС. В дополнение к этим конденсаторам, конденсатор емкостью от 10 до 470 мкФ должен быть включен между шинами источника питания и земли в точке входа линии питания PCB.

  Дополнительно, рекомендуется добавлять цепочку ферритовых ячеек между конденсатором и источником питания. Обратите внимание, что типичная микросхема микропроцессора может иметь несколько подводов источника питания. Например, аналого-цифровой преобразователь (АЦП) требует опорных напряжений (V_{RH} и V_{RL}). Оба эти напряжения также должны быть развязаны от переходных процессов.
- ***Синхронизация частоты:*** При проектировании встроенной системы управления должны использоваться наиболее низкочастотные таймеры, удовлетворяющие системным требованиям.

  Таймеры являются известными источниками помех, если их неправильно экранировать; они генерируют значительные гармонические частоты, намного превышающие частоту тактирующего сигнала. В микроконтроллере 68HC12 используется кристалл на 16 МГц, чтобы генерировать базовую частоту в 8 МГц. Обратите внимание, что значительные гармоники могут существовать на частотах до 160 МГц. Обычно базовая частота синхронизации для процессора обеспечивается кристаллом. Кристалл должен быть помещен в центр печатной платы схемы и приклеен к ней. В дополнение, корпус кристалла должен быть заземлен.
- ***Выводы микросхем:*** известный источник помех получается при неправильном подключении цифровых и аналоговых выводов. В цифровой схеме, неподключенный ввод может приводить к автоматическому смещению транзистора в активную область. К тому же эти неподключенные вводы также действуют как миниатюрные приемные антенны для помех. Такие вводы должны быть или подключены к напряжению источника (V_{DD}) через резистор в 4.7 кОм или к земле (V_{SS}). Резисторы можно легко приобрести в корпусах с односторонними (*SIP*) или двусторонними выводами (*DIP*), чтобы легко подключить неиспользуемые выводы порта контроллера. Кроме вводов порта, выводы аппаратного прерывания должны также быть аналогично подключены, если они не используются. Иначе, могут быть инициализированы случайные прерывания. Кроме того, любые неиспользуемые затворы в интегральной цифровой схеме также должны быть подключены.

- ***Способы заземления:*** В любой встроенной системе управления, имеются множество точек схемы, требующих заземления. Считается, что «земля» – это эквипотенциальный проводник, напряжение на котором равно нулю. Нетрудно представить себе проблемы, которые возникнут в схеме, если это не так. Следовательно, чрезвычайно важно гарантировать, что все эти различные точки схемы – действительно эквипотенциальны. Интуиция подсказывает нам, что проблему решает простое подключение всех заземляемых узлов на одну общую точку. Эта методика, которая названа заземлением в одной точке, хорошо работает при низких частотах. В качестве варианта можно предложить множество точек соединения к заземленной плоскости при высоких частотах. Реальным решением является совместное применение двух методов, называемых смешанным (гибридным) заземлением. Важно также отделить цифровые и аналоговые вводы, устройства ввода-вывода, и ключевые компоненты друг от друга на печатной плате. Комбинация методов заземления может затем использоваться для каждой подсистемы. Например, каждая подсистема может соединяться с заземлением источника питания в одной точке, а затем в подсистеме может использоваться комбинация многоточечных и смешанных заземлений.
- ***Защита от внешних помех:*** имеется ряд методов защиты встроенной системы управления от внешних помех. Как мы видели в примерах, рассмотренных ранее в настоящей главе, чтобы минимизировать чувствительность к внешним помехам может использоваться экранирование корпуса и кабелей с последующим заземлением этого экрана.

  Для кабелей, проводящих низкочастотные сигналы, экран должен быть заземлен с одного конца. Для кабелей, проводящих высокочастотные сигналы, экран должен быть заземлен с обоих концов. Должен быть заземлен также корпус прибора. Тем самым контур, вызывающий помехи будет закорочен на землю.
- ***Бездребезговые ключи:*** В главе 5 мы обсуждали бездребезговые переключатели в качестве способа подключения. Их можно также рассматривать как метод снижения помех от переходных процессов. Мы говорили, что идеальный кнопочный переключатель обычно в нормальном состоянии имеет на своих контактах сигнал высокого логического уровня, который при нажатии превращается в сигнал низкого уровня. При переключении в реальных переключателях может возникать явление дребезга. То есть из-за неидеальных механических характеристик переключателя, переключатель производит несколько замыканий и размыканий контактов при переключении. Поскольку микросхема 68HC12 работает в мегагерцовом диапазоне, она обладает достаточным быстродействием, чтобы реагировать на дребезг переключателя как на ряд включений и выключений. Чтобы предотвратить эти явления, могут использоваться бездребезговые методы включения. Переключатели могут использовать противодребезговые аппаратные средства или программные методы. При программном обеспечении отсутствия дребезга, читается первый контакт переключателя, а затем вводится программное блокирование чтения сигнала на 100 – 200 мс. В течение этой короткой задержки, дребезг не действует на микросхему. Эта методика подавления дребезга позволяет также уменьшить помехи от переходных процессов на вводах схемы. Аппаратные средства и программное обеспечение методов подавления дребезга были подробно обсуждены в разделе 5.5 главы 5.

- ***Создание условий на выводах:*** Входные и выходные сигналы платы должны быть защищены от помех. Чтобы предотвратить повреждение назгрузки статическим электричеством, необходимо включить дополнительный резистор последовательно с входным штырьком. Хорошим методом является также включение встроенного фильтра на входе. Это легко осуществить, пропустив входной провод через цепочку ферритовых ячеек, которая действует как фильтр для высокочастотных колебаний, возникающих при переключениях. Возможно вы считаете, что изготовить такие цепочки ячеек достаточно сложно. Однако на рынке имеются ферритовые «цепочки ячеек» в разнообразных конфигурациях, включая фильтры, которые могут быть закреплены на существующих ленточных кабелях или даже на одиночной линии. Имеются также ферритовые фильтры, которые могут быть установлены на печатной плате.
- ***Методы размещения монтажа на многослойной печатаной плате:*** Так как ваше изделие наиболее вероятно будет смонтировано на многослойной плате, важно понять, как расположить монтаж на плате, чтобы уменьшить помехи. Если используется многослойная плата, крайние слои, должны состоять из дорожек источника питания и земли. Дорожки, проводящие сигналы, должны быть проведены на промежуточных уровнях, расположенных между между уровнями источника питания и уровнем заземления. Линии сигнала на смежных уровнях должны быть направленный перпендикулярно друг к другу. Неиспользуемое пространство (промежутки) на печатной плате должно быть покрыты заземленной металлизацией. В дополнение к этим методам, дорожки, передающие сигналы таймеров должны быть сгруппированы вблизи друг от друга. Следует избегать резких поворотов дорожек (под углом в 90°). Вместо этого, необходимо использовать плавные изменения направления. Должны быть удалены короткие ответвления от основных дорожек платы.

Как мы упомянули ранее, все цепи цифровые, аналоговые, и т.д. должны быть отделены друг от друга. Кроме того, сигнальные дорожки должны быть расположены как можно дальше друг от друга, чтобы предотвратить возможную связь между параллельными дорожками.

6.4. Защитное программирование

В предыдущем разделе мы обсуждали метод «проб и ошибок» при проектировании, позволяющий снизитьть влияние помех. В этом разделе мы исследуем эффективные программные методы, чтобы снизить чувствительность к помехам. Эта информация адаптирована из статей по применению (Application Notes) фирмы Motorola/Freescale Semiconductor. Точная ссылка приведена в разделе «Что еще прочитать» в конце этой главы.

- ***Изменение назначения выводов порта:*** В многих применениях система микроконтроллера используется, чтобы принимать информацию с внешних вводов и затем генерировать соответствующие сигналы на выходе. При этом хорошей практикой, чтобы периодически изменять направление передачи данных в регистрах и выводах, связанных с этими портами.

- ***Опрос:*** В этой методике входной вывод опрашивается в течение некоторого периода времени, чтобы гарантировать, что зафиксирован входной сигнал, а не сигнал помехи. Ранее мы обсуждали противодребезговые методы переключения. Как было упомянуто, при одном из таких методов должен контролироваться входной сигнал, чтобы убедиться, что он не изменяется в задванном временном интервале.
- ***Эстафетная передача:*** Эта методика гарантирует, что ключевые части алгоритма выполняются в правильном порядке. Это осуществляется, путем расположение в памяти сайта эстафетной коллекции. Когда выполняется алгоритм, в сайт в порядке следования помещаются маркеры. После ввода каждого нового раздела программного обеспечения, исследуется эстафетный сайт коллекции, чтобы гарантировать, что предшествующие части программного обеспечения уже были выполнены в правильном порядке. Если какой-то маркер отсутствует, значит программное обеспечение достигло новой области неправильно. Допустим, например, что вы имеете восемь функций, вызываемых в определенной последовательности. Когда первая функция вызвана, маркер помещается на первое место в эстафетном сайте. Когда вызывается вторая функция, первый маркер проверяется, чтобы удостоверится, что он находится правильном на месте. Если это условие выполнено, помещается второй маркера, показывающий, что была инициализирована вторая функция. Эта процедура проводится для каждой последующей функции.
- ***Неиспользуемая память:*** микросхема B32 содержит 32 Кб флэш-памяти, для хранения программы. Было бы идеально записывать в эту память программы объемом точно в 32 Кб. Как эффективно использовать свободное пространство памяти? Хороший программный прием состоит в том, чтобы поместить несколько команд программного прерывания (SWI) на свободное пространство. Следовательно, если процессор неправильно закончит программу в этом пространстве, будет выдана команда программного прерывания. Это обеспечивает устранение сбоев в программе.
- ***Сторожевой таймер (COP – Computer operating properly):*** При работе встроенной микропроцессорной системы, необходимо, чтобы она продолжала функционировать правильно. В случае сбоя, процессор должен иметь возможность восстановиться. Одним из методов, который позволяет процессору выйти из режима «зависания» является введение в его состав сторожевого таймера COP. Этот таймер должен непрерывно сбрасываться при нормальном выполнении программы. Если счетчик сторожевого таймера COP переполняется, генерируется сброс COP. Чтобы сбросить таймер до переполнения на регистр сброса таймера COP (COPRST) должна быть послана последовательность команд $55 и $AA. Команды сброса могут быть посланы между командами для 68HC12 контроллера; однако, они должны посылаться через достаточно короткий интервал, чтобы предотвратить ожидание при нормальном выполнении программы. Несколько последовательностей команд $ 55 и $AA могут быть помещены в ключевые части программы. Если программа неправильно становит «зависает», COPRST не будет получать требуемую последовательность сбросов $ 55 и $AA. Тогда, контроллер получает команду сброса от таймера COP. Сброс может затем устранять дефект, который первоначально вызвал «зависание».

6.5. Методики испытаний на наличие помех

Даже если вы добросовестно применяете методы, описанные в предыдущих разделах, у вас нет никакой гарантии, что ваш прибор не будет восприимчив к помехам или не будет их излучать. До передачи прибора на полномасштабные испытания, было бы полезно проверить характеристики помех, с помощью некоторых дешевых методов тестирования. В следующих двух разделах мы обсуждаем такие методы проверки микроконтроллерной системы на создание помех и чувствительность к помехам. Эти методы созданы Gerry O. (он хочет остаться анонимным), который в течение более чем 35 лет разрабатывал электронныце приборы, а теперь является президентом и главный разработчиком фирмы, занимающейся разработкой и изготовлением международных электронных проектов.

6.5.1. Обнаружение помех

Чтобы определять если прототип, встроенная система управления выделяет помех, следующая методика может использоваться: «Обычно я (Gerry O.) настраиваю телевизор на второй канал (не подключая антенну) чтобы видеть, присутствуют ли излучаемое высокочастотные помехи, передаваемые через излучение (RFI). Я настраиваюсь также на полосу AM радио (автомобильного приемника). AM радио обычно самый лучший тест для «помех» от источника питания. Телевизор самое лучшее для высокой частотной излучения. Обратите внимание, что звук телевизора обычно не реагирует на ВЧ излучение, поскольку пердача звука идет на длинных волнах (диапазон FM) , а видеосигнал на коротких (диапазон AM). Так что, ищите помехи в изображении. Я где-то читал что, если вблизи проходит торнадо, изображение на втором канале исчезает из-за перегрузки автоматической схемы регулировки уровня (AGC). Некоторые компьютерные программы также наводят помехи на втором канале, если телевизор расположен слишком близко к компьютеру.

6.5.2. Испытание на чувствительность к помехам

Дешевый метод для испытания системы микроконтроллера на чувствительность к помехам состоит в том, чтобы использовать мощное размагничивающее устройство видеозаписи как показано на рис. 6.6. Цель испытания состояла в том, чтобы видеть, могло бы сильное переменное магнитное поле создавать сигналы в микропроцессоре или периферийных устройствах. Такие сигналы могли бы привести к сбоям в работе. Хотя таким способом невозможно смоделировать молнию или электромагнитные импульсы, это, тем не менее, хороший быстрый тест на чувствительность к помехам. Gerry указал, что несколько лет назад разряд молнии произошел вблизи от окна его офиса. Его компьютер был выключен, но все же несколько файлов в нем были уничтожены. Молния повредила также схему драйвера принтера на системной плате ПК (на канале LPT1).

Доктор Джерри Хаманн (университет штата Вайоминг) предлагает другой метод испытания на чувствительность к помехам. Он предлагает поводить рукой над испытуемым прибором. Известно, что ваше тело коварный источник статического электричества. Когда вы поднесете руку вплотную к схеме, она должна продолжать стабильно работать.

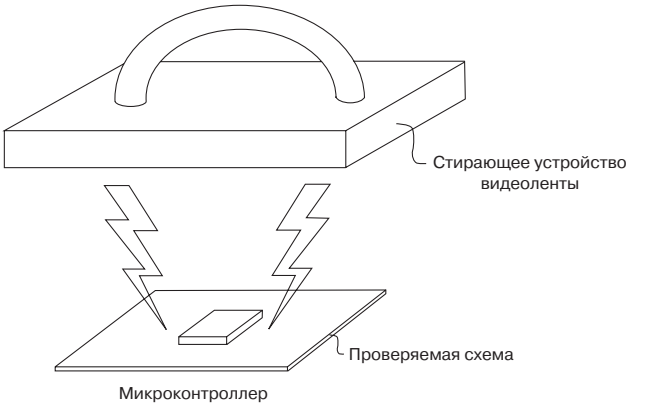


Рис. 6.6. *Дешевый метод для обнаружения чувствительности к помехам и в схеме Размагничивающее устройство видеозаписи обеспечивает дешевый источник для сильных переменных магнитных полей, чтобы проверить чувствительности микропроцессора к помехам*

Если при этом происходят изменения в работе схемы, вы должны проверить, правильно ли подключены свободные выводы микросхем.

Напомним, что эти проверки не могут заменить проверку на помехи с помощью комплекта испытательной аппаратуры. Однако, они обеспечивают быструю проверку прибора на электромагнитную совместимость.

6.5.3. Испытания на электомагнитную совместимость

До производства, разработанной встроенной системы на базе контроллера, изделие должно быть проверено на ЭМС. Правила и рекомендации по управлению этих испытаний разрабатываются Федеральной комиссией по связи (FCC) в Соединенных Штатах и в Европейском Экономическом Сообществе (ЕЭС). Кроме того, Управление продовольствия и медицинских препаратов (FDA) разрабатывает стандарты для медицинских устройств. Мы не собираемся подробно рассматривать здесь эти испытания. Правила постоянно совершенствуются (что совершенно правильно), и любая приведенная информация, быстро устарела бы. Вместо этого мы приводим краткий обзор имеющихся правил в ссылках на использованную литературу, которые приводятся в разделе «Что еще почитать» в конце главы, чтобы получить наиболее современную информацию.

Правила FCC и части 15 Правил для устройств высокой частоты формулируют правительственные правила и рекомендации для радио устройств (РФ), способных к излучению энергии в диапазоне от 9 кГц до 200 ГГц. FCC и часть 15 в настоящее время устанавливают три процедуры для проверки приборов на соответствие требованиям ЭМС:

- ***Проверка:*** изготовитель изделия регистрирует протокол испытаний на соответствие требованиям ЭМС.
- ***Сертификация:*** Комиссия FCC дает обзор применений ЭМС.
- ***Декларация соответствия:*** эти испытания выполняет лаборатория, уполномоченная Национальным Институтом Стандартов и Технологии (NIST). Например, для ряда изготовителей испытания по ЭМС выполняет лаборатория Underwriters Laboratory (UL).

Международные стандарты, касающиеся ЭМС, разработаны прежде всего Международной Электротехничекской Комиссией (МЭК, IEC). Стандарты, разработанные МЭК могут быть разбиты на следующие группы:

- Разряды электростатического электричества (публикация IEC 61000–4–2)
- Излучение электромагнитных полей высокой частоты (публикация IEC 610004–3)
- Электрические быстрые переходные процессы/пакет (публикация МЭК: IEC 61000–4–4)
- Электромагнитные импульсы (suges) (публикация МЭК: IEC 61000–4–5)
- Устойчивость к помехам, передаваемым за счет проводимости (публикация IEC 61000–4–6)
- Устойчивость к магнитным полям (публикация IEC 61000–4–9)
- Посадки напряжения, кратковременные прерывания и изменения напряжения (публикация IEC 61000–4–11)

Наиболее важный урок, следующий из нашего обсуждения, состоит в том, что требования к ЭМС должны быть частью технических требований на изделие, сформулированных до начала цикла проектирования

В следующем разделе мы исследуем, как управлять энергопотреблением во встроенных системах.

6.6. Управление энергопотреблением

Часто микроконтроллерные системы представляют собой переносные или дистанционно-управляемые модули. Обеспечение подходящего источника напряжения для системы становится главной задачей – задачей, которая может быть обоюдоострым мечом. С одной стороны, должна быть разработана подходящая система питания, чтобы обеспечить соответствующее напряжение и текущие требования для системы в течение приемлемого временного интервала. С другой стороны проектировщик должен уменьшить потребляемую мощность встроенной системы контроллера. Кроме того, система должна иметь защиту от понижения питающего напряжения. Вся эта совокупность требований рассматривается в данном разделе. Мы ограничиваем наше обсуждение системами с аккумуляторным питанием.

6.6.1. Параметры потребляемой мощности для микроконтроллера 68HC12

Чтобы разработать систему питания для встроенной микроконроллерной системы, необходимо определить несколько параметров проекта:

- Напряжения питания, необходимые для встроенного контроллера, периферийных устройств, и всех компонентов системы;
- Токи утечки для каждого компонента системы;
- Ожидаемая период работы системы без замены или перезарядки батареи;
- Температура среды.

После определения параметры, можно приступить к проектированию подходящего батарейного источника питания. Для определения этих значений необходимо тщательно исследовать технические данные для каждого компонента системы. Все вычисления должны проводиться для наихудшего случая – другими словами, при наиболее критичных значениях токах утечки и рабочей температуры.

6.6.2. Типы батарей

После определения параметров системы можно начать выбор батареи. На рис. 6.7 и 6.8 представлен краткий обзор характеристик распространенных типов батарей. В обзоре приведены только типы батарей, напряжения и емкости. Полный обзор характеристик приводится в каталоге источников питания для электрононных устройств. Такие каталоги обеспечивают хороший обзор для всего разнообразия батарей. Но сначала рассмотрим характеристики четырех основных типов батарей:

- ***Щелочные:*** Щелочные батареи относительно дешевы, имеют высокие емкости и большое число типоразмеров. Напряжение на зажимах батареи постепенно уменьшается по мере разрядки; емкость увеличивается при нагревании и значительно уменьшается в низких температурах. Большинство щелочных батарей не может перезаряжаться, но некоторые изготовители выпускают перезаряжающиеся варианты щелочных батареи.
- ***Кадмий-никелевая:*** напряжение на их зажимах при полной зарядке ниже чем напряжения щелочных элементов. Также обратите внимание, что емкость батареи этого типа также значительно меньше чем у щелочных батарей. Характеристика разрядки более пологая чем у щелочных батарей, как показано на рис.6.8.
- ***Никелевая металлогибридная:*** Никелевая металлогибридная (Ni-MH) батарея может перезаряжаться. Этот тип батареи обеспечивает умеренную емкость за умеренную стоимость.
- ***Литиевая:*** литиевая батарея имеет 3,6 В напряжение на зажимах и соответственно большую емкость. Батареи этого типа также имеет довольно пологую характеристику разяда. Однако это свойство сочетается с относительно высокой стоимостью по сравнению с батареями других типов.

6.6.3. Емкость батарей

Что точно представляет собой емкость батареи? Обратите внимание, что емкость измеряется в миллиампер-часах (мА-ч). Эта единица измерения сообщает всю историю. Если известен ток утечки батареи, может быть вычислен срок службы. Например, при напряжении 9 В, емкости 500 мА-ч и токе разряда 5 мА такая батарея должна обеспечивать питание в течение приблизительно 100 часов. Однако, емкость уменьшается при высоких токах разряда и низких рабочих температурах.

6.6.4. Стабилизация напряжения

Вспомним, что питающее напряжение (V_{DD}) для микропроцессора 68HC12 имеет довольно малый допуск, электрические характеристики для контроллера, показывают, что питающее напряжение должно поддерживаться в диапазоне 5 В ±10 %. Чтобы поддерживать это, относительно постоянное напряжение при изменении условий, используется схема стабилизатора. Типичная схема стабилизатора – супервизора микропроцессора, показанный на рис. 6.9.

Тип	Неперезаряж. щелочная		Перезаряж. Ni-Cd		Перезаряж. Ni-металлогибрдная		Неперезаряж. Li	
Размер	Напряжение	Емкость	Напряжение	Емкость	Напряжение	Емкость	Напряжение	Емкость
D	1,5	15,000	1,2	1,200	1,2	8,000	3,6	16,500
C	1,5	7,000	1,2	1,200	1,2	4,500	3,6	7,200
AA	1,5	2,250	1,2	500	1,2	2,250	3,6	2,100
AAA	1,5	1,000	1,2	180	1,2	600	–	–
N	1,5	650	1,2	150	1,2	–	–	–
9 В транзистор	9,0	550	–	–	9,0	170	–	–
6 В фонарь	6,0	11,000	–	–	–	–	–	–

Рис. 6.7. *Характеристика пропускной способности для различных типов батареи*

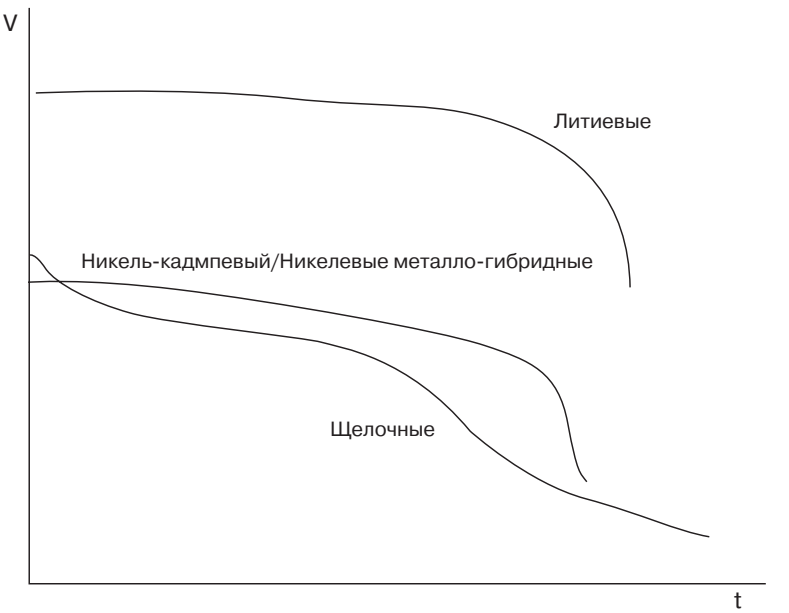


Рис. 6.8. *Характеристики разряда батареи*

Это качественные характеристики. Детальные семейства харакьеристик с учетом разрядного тока и температуры приводятся в литературе, предоставляемой изготовителями

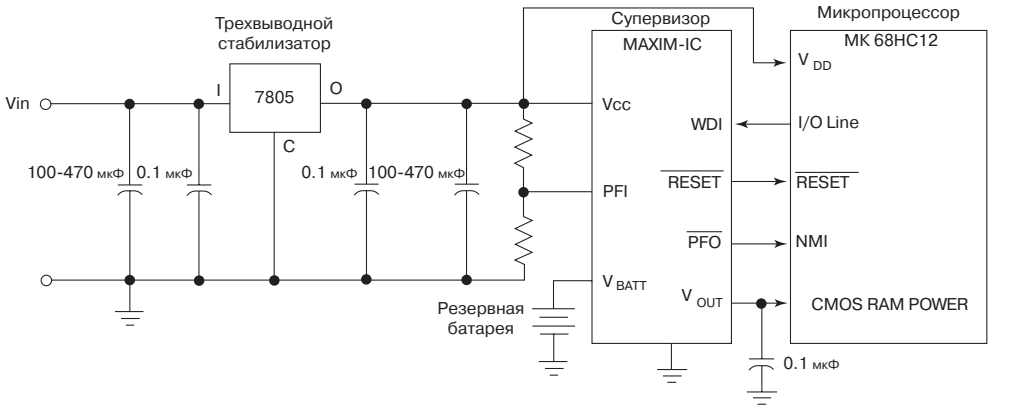


Рис. 6.9. *Цепи супервизора для микропроцессора*

Стабилизирующее устройство состоит из микросхемы стабилизатора, оборудованного помехоподавляющими конденсаторами на входах и на выводах. Микросхема стаюбилизатора обычно представляет собой устройство с тремя выводами: вход (I), выход (O) и общим выводом (C). Эти стабилизирующие устройства определяются выходным напряжением и номинальным током, Хорошим практическим решением является выбор регулятора, номинальный ток которого по крайней мере вдвое превышает максимальную токовую нагрузку. Полная линейка стабилизаторов представлена серией 78XX. Под обозначенитем XX здесь подразумевается номинальное значение выходного напряжения.

Например, на рис. 6.9 мы использовали микросхему стабилизатора 7805 (+ 5 В). В серию стабилизаторов 7805 входят приборы для широкого ряда номинальных токов. Стабилизаторы серии 79XX представляют собой ряд стабилизаторов отрицательных напряжений.

На входе и выходе используются сдвоенные конденсаторы: конденсаторы емкостью 0,1 мкФ для подавления высокочастотные помех и фильтровые конденсаторы с емкостью в диапазоне от 100 до 470 мкФ, для снижения выходных пульсаций.

Входное напряжение на схему стабилизатора можно подавать от источника постоянного напряжения или от батареи подходящего типа. При любом из этих вариантов, входное напряжение для стабилизатора должно обычно быть по крайней мере на 3 В больше, чем необходимое выходное напряжение стабилизатора.

В случае сбоя питания или разряда батареи, процессор необходимо перевести на резервное питание. Это легко сделать, используя схему управления микропроцессором.

6.6.5. Схемы супервизора для микропроцессора

Имеется много различных схем управления микропроцессором, производимых несколькими изготовителями. Приведем краткий обзор функций обеспечиваемых супервихором компании MAXIM.

Она выполнет следующие функции:

- Восстанавливает входное напряжение сброса в течение включения питания и при кратковременных провалах напряжения питания;
- Переключает на батарею резервного питания RAM CMOS, CMOS микропроцессора или другие маломощные логические схемы;
- Создает импульс сброса, если вспомогательный сторожевой таймер не переключается на определенном временном интервале;
- Использует пороговый детектор на 1,3 В для предупреждения сбоев питания, при низком напряжении батареи или подключении источника питания отличного от источника постоянного напряжения + 5 В.

Типовая схема использования супервизора показана на рис. 6.9. Супервизор постоянно сравнивает напряжение на входе V_{CC} с напряжением резервной батареи V_{BATT}, и переключает на нее питание, когда напряжение на выходе V_{OUT} становится меньше V_{BATT}. Схема сравнения имеет гистерезис, позволяющий предотвратить многократные повторные переключения при близких значениях $_{VCC}$ и V_{BATT}.

В состав супервизора входит также сторожевой таймер. Как уже упоминалось, микросхема 68HC12 имеет сторожевой таймер COP. Таймер супервизора совершенно подобен ему. Сторожевой таймер генерирует сигнал сброса, если сторожевой вход (WDI) не сбрасывается микропроцессором на интервале ожидания таймера. Подобно системе COP процессора 68HC12, пользователь должен встраивать в состав пользовательской программы команды, позволяющие регулярно создавать сигнал на входе WDI. Если программа становит «зависает», WDI не будет периодически сбрасываться, и на канале RESET супервизора появится сигнал сброса.

6.6.6. Меры энергосбережения

Проектировщик системы может использовать несколько методов, чтобы уменьшить потребляемую мощность встроенной системы управления:

- Рабочая частота: встроенный контроллер должен работать на самой низкой частоте, допустимой для специфического применения. Ключ CMOS потребляет мощность при переключении с одного логического уровня на другой. При более низких рабочих частотах количество переходов уменьшается, и, следовательно, уменьшается потребляемая мощность.
- Команды STOP и WAIT: система команд проессора 68HC12 содержит две команды STOP и WAIT, переводящие процессор в неактивное состояние, и позволяющие уменьшить потребляемую мощность. Например, когда микросхема B32 работает на частоте 8 МГц, ее выходной ток обычно составляет 45 мА. В режиме WAIT, ток, уменьшается до 5 мА а в режиме STOP – до 10 мкА. При выполнении обеих команд в стек 68HC12 помещается адрес возврата и содержание регистров ЦП. Команда STOP останавливает все таймеры системы, при выполнении же команды WAIT таймеры продолжают работать. Обе команды требуют выполнения операций прерывания или сброса для продолжиения нормальной работы системы. Обратите внимание, что во многих применениях 68HC12 работает в режиме управляемых прерываний. То есть процессор инициализируется, а затем ждет события, вызывающего прерывание.

- Активация подсистемы: Некоторые подсистемы 68HC12 имеют переключатели «вкл\выкл». Например, подсистема таймера бит разрешения работы таймера (TEN) в регистре управления системой таймера (TSCR). Также, подсистема аналого-цифрового преобрахзователя (АЦП) имеет бит подачи питания на АЦП (ADPU) бит в регистре управления АЦП 2 (ATDCTL2). Это позволяет обеспечивать подачу питания на системы только на необходимых временных интервалах необходимо и отключать их, чтобы сохранить мощность когда они не используются.

6.7. Заключение по главе 6

Мы обсудили условия применения CMOS, внешние и внутренние источники помех в схеме, методы снижения помех, методики испытаний на помехи и современные рекомендации по борьбе с помехами. Мы также обсуждали проблемыпитания для встроенной системы, включая типы батареи, ее емкость, схемы управляющие питанием и меры энергосбережения. Хотя эти проблемы относятся к самым разным источникам, все они связаны общей нитью – их решение может превратить хороший проект на бумаге устройство, правильно работающее в «реальном мире».

6. 8. Что еще прочитать?

1. Atmel, Inc. «EMC Design Considerations.» Application Note AVR040. 2004.
2. Atmel, Inc. «External Brown-out Protection.» Application Note AVR180. 2002.
3. Barrett, S. F. «Heart Arrhythmia Simulator.» Senior Design Project presented at the annual Nebraska Academy of Science, Lincoln, NE, 1979.
4. Campbell, D. «Designing for Electromagnetic Compatibility with Single-Chip Microcontrollers.» Application Note AN1263/D. Motorola, Inc., 1995.
5. Catherwood, M. «Designing for Electromagnetic Compatibility.» Application Note AN1050/D. Motorola, Inc., 2000.
6. Corp, M. Bruce. ZZAAP! Taming ESD, RFI, and EMI. Academic Press, 1990.
7. Federal Communication Commission. Rules and Regulations Part 15 Radio Frequency Devices, www.fcc.gov, 2004.
8. Glenewinkel, M. «System Design and Layout Techniques for Noise Reduction in MCU-Based Systems.» Application Note AN1259/D. Motorola, Inc., 1995.
9. Horowitz, Paul, and Winfleld Hill. Art of Electronics, 2nd ed. Cambridge, England: Cambridge University Press, 1989.
10. Internationa) Electro technical Commission. IEC 61000 Series Guidelines, www.iec.ch.
11. Johnson, Howard. High-Speed Digital Design: A Handbook of Black Magic. Upper Saddle River, NJ: Prentice Hall, 1993.
12. Kobeissi, I. «Noise Reduction Techniques for Microcontroller-Based Systems.» Application Note AN1705/D. Motorola, Inc., 1999.
13. Lun, T. C «Designing for Board Level Electromagnetic Compatibility.» Application Note AN2321/D. Motorola, Inc., 2002.

14. Maxim Integrated Products, «MAXIM Microprocessor Supervisory Circuits.» MAX 690-695. April 1995.
15. Motorola, Inc. «High-Speed CMOS Logic Data.» 1989.
16. Welch, T. B. «Teaching Three Phase Power-A Low Voltage Approach.» Paper presented at the ASEE Annual Conference, Milwaukee, WI, June 1997. Welch, T. B., and J. N. Berry. «Teaching Three-Phase Electrical Power Using a Low-Vollage Power Supply.» Paper presented at the ASEE Annual Conference, Seattle, WA, 1998.

6.9. Вопросы и задания

Основные

1. Опишите методы правильного обращения с устройствами CMOS.
2. Что может случиться, если обращаться с устройствами CMOS неправильно?
3. Что такое «дребезг» переключателя? Как исклчить влияние «дребезга» переключателя?
4. Опишите методы снижения потребляемой мощности во встроенных микроконтроллерных системах.
5. Создайте карту выбора батареи из обычно доступных типов. Включить такие пункты, как тип батареи, конструктивные размеры, емкость и стоимостные показатели.
6. Что такое стабилизация напряжения? Почему важно использовать методы стабилизации напряжения во встроенных микроконтроллерных системах?

Более сложные

1. Опишите различие между ESD, RFI, ЭМП помехами, посадками, и импульсами напряжения. Приведите пример для каждого типа помех.
2. Почему таймеры для встроенных микроконтроллерных систем представляют собой традиционный источник помех? Какие методы могут использоваться, чтобы снизить помехи от таймеров?
3. Почему важно подсоединять свободные вводы портов во встроенных микроконтроллерных системах? Как эти вводы должны быть правильно подключены?
4. Почему важно подсоединять свободные входы аппаратного прерывания во встроенной системе контроллера? Как эти вводы должны быть правильно подключены?
5. Расскажите о методах программной защиты.

Исследовательские

1. Разработайте систему источника питания с напряжением 5 В для 68HC12. Система должна иметь резервную литиевую батарею и содержать супервизор компании MAXIM. Разработайтете блок-схему системы и детально опишите ее работу.

2. Опишите на двух страницах методы проектирования, позволяющие минимизировать чувствительность к помехам.
3. Изучите рекомендации от IEC 6100–4–2 до 6100–4–9. Кратко опишите каждое испытание.

Глава 7

ПРИМЕРЫ ВСТРОЕННЫХ СИСТЕМ УПРАВЛЕНИЯ

ПОСЛЕ ИЗУЧЕНИЯ ГЛАВЫ ВЫ СМОЖЕТЕ:

ЦЕЛИ

- Применить структурный системный подход к проектированию встроенных микроконтроллерных систем;
- Создать детальное описание проекта;
- Выбрать модули в составе 68HC12, необходимые для конкретного применения;
- Описать структуру программного обеспечения встроенной микроконтроллерной системы, используя структуру программы и блок-схему алгоритма;
- Написать программу на языке Си для конкретного приложения.

7.1. Система привода робота, движущегося вдоль стенок лабиринта 412
7.2. Лазерный проектор 426
7.3. Цифровой вольтметр 438
7.4. Стабилизация скорости вращения двигателя с использованием оптического тахомера 445
7.5. Парящий робот 463
7.6. Система защиты компьютера, основанная на нечеткой логике 472
7.7. Электронная версия игры в «15» 490
7.8. Приложение: программирования флеш-памяти B32 EVB 512
7.9. Заключение по главе 7 513
7.10. Что еще прочитать? 514
7.11. Вопросы и задания 515

Представим себе красивый, старинный дубовый сундучок для инструментов, подобный показанному на рис.7.1. Каждая его секция разбита на отделения, для хранения ценных инструментов в каждом из лотков. На каждом из войлочных лотков аккуратно написано название инструмента. В том же лотке хранится также и инструкция по применению инструмента. В предыдущих главах мы описывали аппаратные средства, программное обеспечение, и средства разработки систем. Именно они и лежат в этом «сундучке» инструментов, позволяющих разрабатывать, производить и внедрять встроенные системы управления. В этой главе мы покажем на различных примерах, как разрабатываются встраиваемые микроконтроллерные системы. Мы тщательно выбрали проекты, чтобы показать, как используются подсистемы МК 68HCI2 и HCS12 для выполнения разнообразных задач. Методы совместного использования этих периферийных модулей мы покажем на ряде примеров. Для каждого такого примера, мы приведем детальное описание проекта, список используемых в нем подсистем 68HC12, краткие основы теории, если это необходимо, детальную структуру программы, сопровождаемую блок-схемой алгоритма и хорошо документированным программным кодом. Будут рассматрены следующие примеры применения:

- Система привода для робота, движущегося вдоль стенок лабиринта;
- Лазерный проектор;
- Цифровой вольтметр;
- Система стабилизации скорости вращения двигателя с оптическим тахометром;
- Парящий робот;
- Система защиты от несанкционированного внедрения на базе нечеткой логики;
- Электронная версия популярной игры в «15».

В конце главы мы рассмотрим также процедуру программирования Flash-памяти с помощью отладочной платы «B32».

7.1. Сиситема привода робота, движущегося вдоль стенок лабиринта

7.1.1. Описание проекта

В техническом задании на этот проект, предлагается разработать автономный робот, который мог бы проходить через неизвестный лабиринт. Робот должен двигаться через лабиринт, находя стенки лабиринта с помощью инфракрасных (ИК) локаторов (пар излучатель-приемник), и принимая решения, продвигаться вперед или назад,

чтобы продолжить путь через лабиринт. При своем движении робот должен также избегать «мин» (магнитов), скрытых в полу лабиринта. Робот обнаруживает «мины» с помощью датчика Холла. Если робот обнаруживает «мину», то он останавливается, дает задний ход и объезжает ее.

Конструкция робота приведена на рис. 7.2. Корпус его состоит из двух легких связанных вместе алюминиевых платформ. На нижней платформе расположены два двигателя постоянного тока, приводящие в движение два больших колеса, установленных с обеих сторон корпуса. При использовании этой конструкции с двумя колесами, робот может управляться подобно танку. То есть, чтобы выполнить поворот по часовой или против стрелки на двигатели подаются сигналы. Два маленьких ролика с обеих сторон от робота используются, чтобы обеспечить его равновесие и устойчивость. Находясь в покое, робот опирается на три точки. На верхней платформе установлены пять ИК пар излучатель-приемник, которые позволяют роботу обнаружить стенки прямо перед собой и с любой стороны от корпуса. Верхная платформа содержит также отладочную плату с микроконтроллером 68HC12, которая используется, чтобы принимать входные сигналы, вырабатывать решения, основанные на этих сигналах, и формировать сигналы управения двигателями для реализации этих решений. На нижней пластине робота установлен датчик Холла, позволяющий обнаруживать магнитные «мины».

Чтобы начать движение через неизвестный лабиринт, робот помещается вблизи от его входа. Предназначение робота состоит в том, чтобы проходить через лабиринты, избегая столкновения со стенками и контакта с «минами», скрытыми в полу. Робот едет вперед, когда на оба двигателя поступает одинаковое постоянное напряжение. Когда робот движется вперед с помощью двигателей постоянного тока, он непрерывно проверяет свое положение относительно стенок и магнитов с помощью пяти ИК локаторов и датчика Холла.

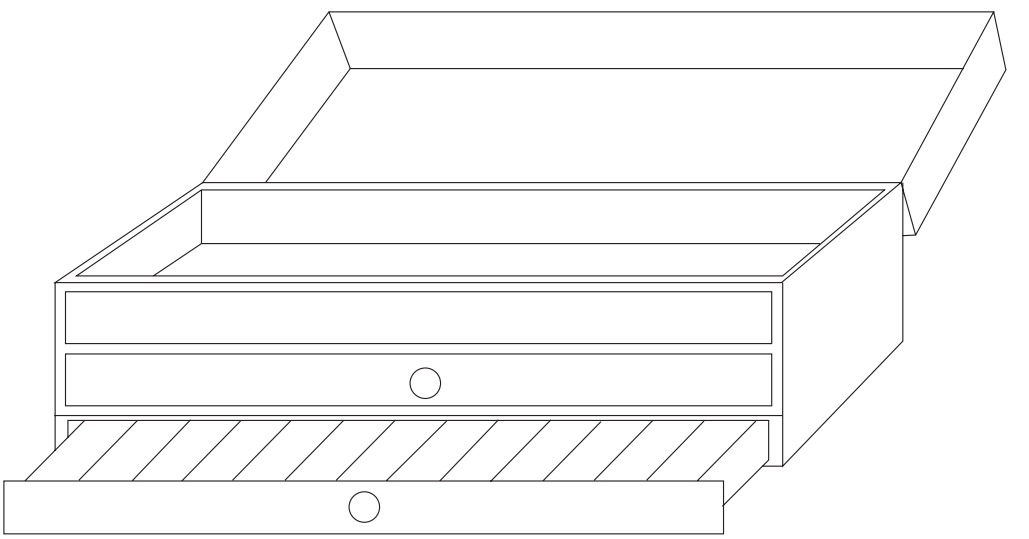


Рис. 7.1. *Старинный сундук, с несколькими ящиками. В лотки ящиков мы уложили инструментальные средства, рассмотренные в предшествующих главах*

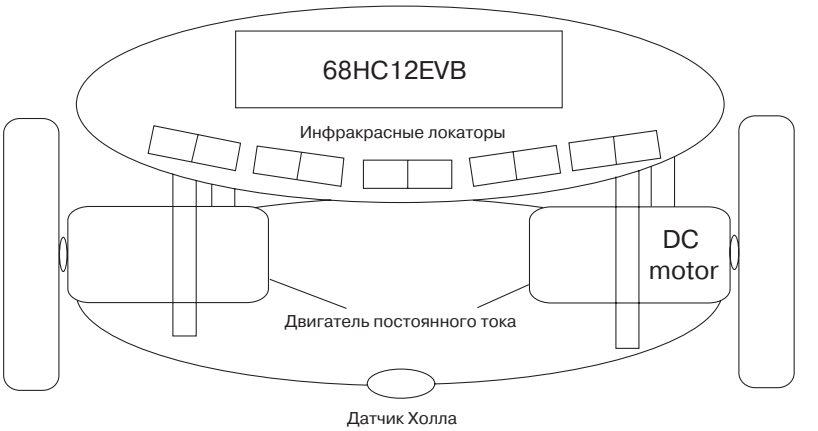


Рис. 7.2. *Робот, движущийся вдоль стенок лабиринта*
Робот оборудован двумя основными колесами, с приводом от двигателей постоянного тока. Когда робот движется вперед, он постоянно следит за положением стенок, используя инфракрасные локаторы (пары излучатель-приемник), и за «минами», скрытыми в полу, с помощью датчика Холла.

Стенки лабиринта окрашены белой краской с высоким коэффициентом отражения, чтобы ИК-сигналы, поступающие от источников, отражались от стенок обратно на приемники. Если робот приближается к стенке, то ее присутствие обнаруживается соответствующим ИК локатором. Например, если робот приближается к углу, расположенному с правой стороны (см. рис. 7.3), то передняя стенка обнаруживается передним локатором, а правая – правым локатором. Робот затем отвечает на принятые им входные сигналы, поворачивая влево, чтобы избежать столкновения со стенками.

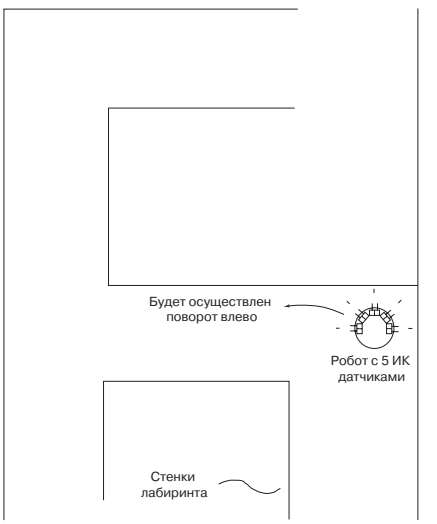


Рис. 7.3. *Робот, разработанный, чтобы обнаруживать близлежащие стенки*
При использовании пяти инфракрасных датчиков, робот готовится сделать левый поворот, избегая стенок, расположенных спереди и справа

Естественная реакция разработчика, получившего задание на столь сложный проект – это паника. Однако если мы используем нисходящее проектирование, мы сможем разбить общее техническое задание на частные требования, выполняемые отдельными подсистемами. Начнем с создания списка функций, требуемых операционной системе робота для выполнения всех его задач. Эти задачи сводятся к следующим:

- Аналого-цифровое преобразование выходных сигналов ИК-датчиков;
- Сравнение сигнала датчика ИК-излучения с пороговым уровнем, соответствующим обнаружению стенки;
- Создание алгоритма поворота робота, позволяющего определить, в каком направлении он должен повернуть в ответ на сигналы ИК-датчиков;
- Разработка функций управления приводом, осуществляющих движение робота вперед и назад и повороты влево или вправо;
- Создание механизма для обработки выходных сигналов датчика Холла;
- Обеспечение функции, позволяющей останавливаться, давать задний ход и обходить обнаруженную «мину»;
- Отображение выполняемых функций на символьном ЖК индикаторе.

7.1.2. Подсистемы 68HC12, используемые в проекте

По списку требуемых функций мы можем определить, какие периферийные модули МК 68HC12 необходимо использовать и какие другие внешние устройства будут необходимы для решения нашей задачи. К используемым устройствам и системам относятся:

- Датчики ИК-излучения и датчики Холла;
- Модуль аналого-цифрового преобразования ATD в составе МК 68HC12 для оцифровки сигналов инфракрасных датчиков и датчика Холла;
- Модуль ШИМ МК 68HC12 для модуляции ширины импульса;
- Интерфейс сопряжения МК с ЖК дисплеем;
- Интерфейс сопряжения МК с ИК-датчиками;
- Интерфейс сопряжения МК с датчиком Холла;
- Интерфейс драйвера двигателя;
- Аккумуляторные батареи, для питания двигателей, датчиков, и отладочной платы MC68HC912B32EVB.

7.1.3. Компоненты системы

Убедимся сначала, что на рынке имеются инструментальные средства, позволяющие выполнить весь список требуемых функций. В предыдущих главах (уложенных в наш сундучок) мы рассматривали следующие компоненты:

- Модуль АЦП в составе МК семейства 68HC12;
- Модуль ШИМ в составе МК семейства 68HC12;
- Интерфейс и программное обеспечение ЖК дисплея;
- Интерфейс драйвера двигателя;
- Батарею для питания системы.

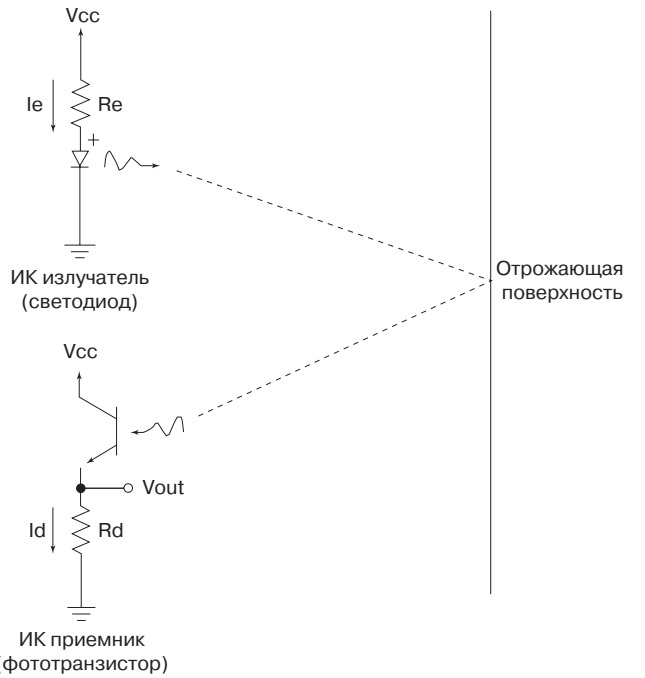


Рис. 7.4. *Пара излучатель-приемник – ИК локатор*
Резистор (Re) ограничивает ток ИК излучателя на уровне номинального значения (Ie). Ток приемника формирует на резисторе Rd выходное напряжение приемника (Vout)

Если вы еще не познакомились с этими устройствами и подсистемами МК, вернитесь к соответствующим главам. Мы же остановимся на двух темах, которые не рассматривали ранее: на паре излучатель-приемник, образующей ИК локатор, и на датчике Холла.

Пара ИК излучатель-приемник. В паре ИК излучатель-приемник объединены источник и приемник инфракрасного (ИК) излучения. Источником является светоизлучающий диод с соответствующей схемой, а приемником – фототранзистор, чувствительный к ИК-диапазону излучения с собственной схемой, показанной на рис. 7.4. Для питания ИК диода используются электрические цепи, описанные ранее в разделе, посвященном светодиодам. Фототранзистор имеет светочувствительный переход база-эммитер. Когда свет соответствующей длины волны падает на переход, в нем возникает базовый ток. В цепь эммитера включен резистор нагрузки, сопротивление которого позволяет обеспечить необходимую величину выходного напряжения. Часто вместо резистора с фиксированным сопротивлением используется 10-оборотный измерительный потенциометр, позволяющий индивидуально подстраивать чувствительность каждого приемника. График зависимости выходного напряжения от расстояния до стенки лабиринта может быть получен экспериментально. Выходной сигнал каждого приемника подается на канал АЦП микроконтроллра 68HC12.

Датчики Холла, как и показывает их название, используют эффект Холла, чтобы генерировать напряжение, пропорциональное напряженности обнаруженного магнитного поля. На рынке имеются датчики Холла двух типов: (1) переключатели и

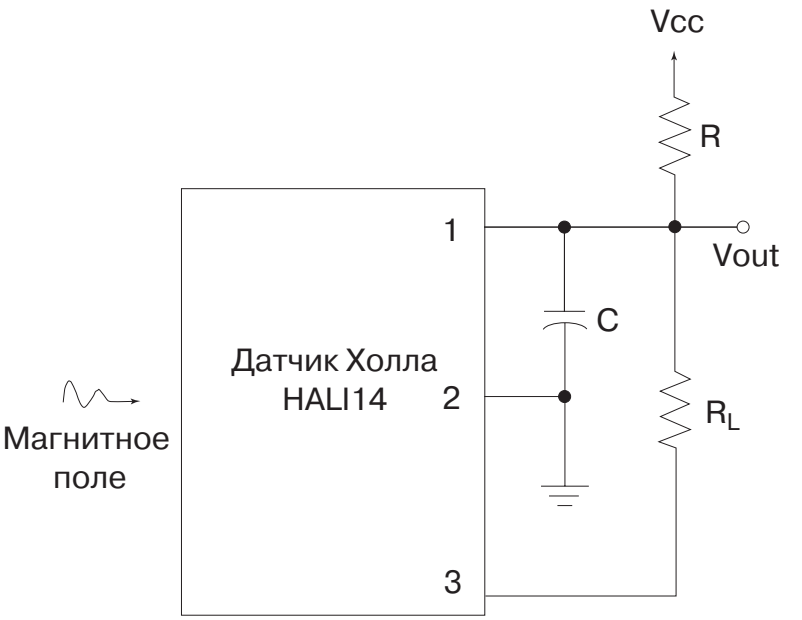


Рис. 7.5. *Датчик Холла HAL114 компании Micronas*

(2) линейные датчики. Датчик переключающегося типа, обнаружив присутствие магнитного поля, фиксируется во включенном состоянии. Он остается в этом состоянии, даже если магнитное поле исчезает.

Линейный датчик формирует аналоговое выходное напряжение, пропорциональное измеряемому магнитному потоку. И именно такой прибор мы будем использовать для обнаружения «мин».

Датчики Холла поставляются несколькими изготовителями. Мы выбрали простой датчик с тремя выводами HAL114 фирмы Micronas, схема включения которого содержит два резистора R, RL и конденсатор C, как показано на рис. 7.5. Как и ранее, график зависимости выходного напряжения от расстояния до стенки лабиринта может быть получен экспериментально. Выходной сигнал с датчика подается на канал АЦП микроконтроллера 68HC12. Полная схема интерфейса связи МК 68HC12 с аппаратными средствами робота показана на рис. 7.6. Закончив на этом краткий обзор аппаратных средств, мы перейдем к обзору программного обеспечения робота.

7.1.4. Структура программы и блок-схема алгоритма

Рабочая программа для этого проекта разработана Томом Шеи, бывшим студентом университета штата Вайоминг, с помощью компилятора ImageCraft ICC12. Прежде чем представить полный текст программы, рассмотрим структуру основной программы и блок-схему ее алгоритма, представленные на рис. 7.7. Мы просим читателя, самостоятельно разработать блок-схему алгоритма для каждой из функций в качестве домашней работы (см. задание 12).

7.1.5. Программный код

```
/*********************************************************************/
/*имя файла: robot.c */
/* Система управления роботом, движущимся в лабиринте: это система  */
/* слежения за стенками лабиринта. Робот использует пять ИК-датчиков, */
/* состоящих из излучателя и приемника, чтобы определять свое положение */
/* относительно стенок лабиринта.                                    */
/* Робот определяет расположение стенки, основываясь на информации, по- */
/* лучаемой от датчиков. Если сигнал от ИК-приемника превыщает порог */
/* опорного напряжения, то стенка находится в непосредственной близости */
/* от робота.Основываясь на информации, получаемой от пяти датчиков,*/
/* робот может определять, какое направление дальнейшего движения   */
/* избрать, чтобы избежать столкновения со стенками лабиринта.       */
/* Датчик Холла позволяет роботу обнаружить магниты или "скрытые мины", */
/* установленные под полом лабиринта. Робот имеет также ЖК дисплей  */
/* для сообщения информации пользователю. Программа использует метод*/
/* полинга для считывания результатов АЦП.Сигнал модуля ШИМ          */
/* управляет драйвером двигателей колес робота.                      */
/*Автор: Томас Шеи. Дата создания: 18 октября 2002                   */
/*Последняя редакция: 4 декабря 2002                                 */
```

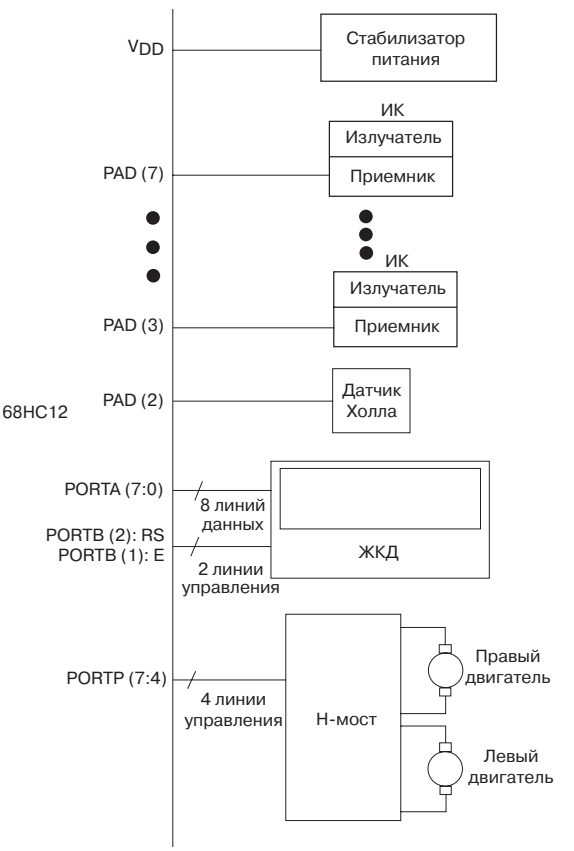
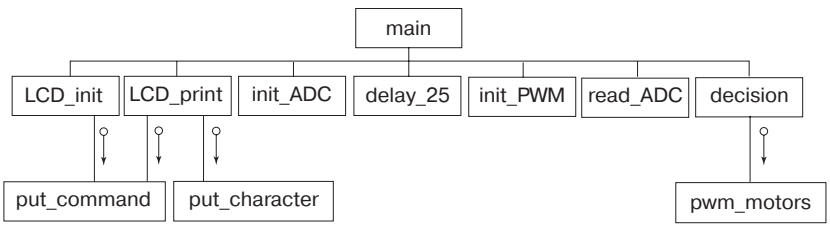
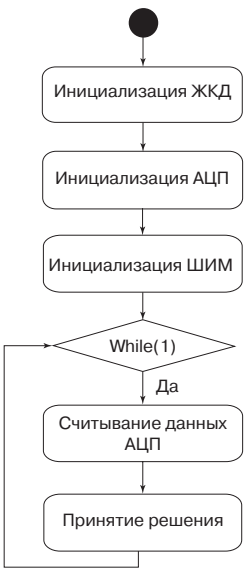


Рис. 7.6. *Интерфейс между аппаратными средствами робота и 68HC12*

```
/*********************************************************************/
/* Включенные файлы*/
#include <912b32.h>
#include <stdio.h>

/*Пороги датчиков были определены экспериментально*/
#define opto_threshold 0x50 /* порог оптического датчика           */
#define hes_threshold 0x80  /* порог датчика Холла                 */
#define forward 0
#define half_left 1
#define half_right 2
#define left_turn 3
#define right_turn 4
#define back_up 5

/*глобальные переменные*/
unsigned int i=0,j=0; /*переменные для программной задержки        */
unsigned char sens[6]={0, 0, 0, 0, 0, 0};/*массив результатов АЦП  */

/*прототипы функций*/
```



а) Структура программы



б) Блок-схема алгоритма UML

Рис. 7.7. *К программе управления роботом, движущимся вдоль стенок лабиринта*

```
void init_adc(void); /*инициализация АЦП                                  */
void read_adc(void); /*считывание значений АЦП                            */
void decision(void); /*передача решения о повороте, основанного на       */
                          /*данных АЦП*                                    /
void init_pwm(void);         /*инициализация ШИМ                          */
void pwm_motors(const char a); */активация ШИМ для пересылки             */
void lcd_init(void);         /* инициализация дисплея                     */
int putchar(char c);         /*вывод символа на дисплей                   */
int putcommand(char c);      /*вывод комагнды на дисплей                  */
void delay_25(void);         /*подпрограмма задержки на 2,5 с             */
void lcd_print(char *string); /*вывод строки на ЖК дисплей                */
void main()
(

asm(".area vectors(abs)\nr   /*инициализация вектора сброса МК            */
    " org 0xFFF8\n"
    " .word 0x8000, 0x8000, 0x8000, 0x8000\n"
    ".text");

lcd_init();  /*инициализация ЖК дисплея                                   */
lcd_print ("LCD initialized");
void delay_25(void);         /* задержки на 2,5 с                         */
init_adc();  /*инициализация АЦП                                         */
lcd_print("ADC initialized");
void delay_25(void); /* задержки на 2,5 с                                */
init_pwm (); /*инициализация ШИМ                                         */
lcd_print("PWM initialized");
void delay_25(void); /* задержки на 2,5 с                                */

while(1)             / *непрерывный цикл                                  */
   {
   read_adc();  /* считать текущее значение из АЦП                       */
   decision();  /* принять решение о направлении движения                */
   }
} /*конец программы main*/
*********************************************************************/
/*initialize_adc: инициализация АЦП контроллера 68HC12                  */
/*********************************************************************/

void init_adc()
{
ATDCTL2 = 0x80          /*Установить бит ADPU для подачи питания на АЦП */
ATDCTL3 = 0x00
ATDCTL4 = 0x7F          /* частотцу P_CLK установить на 125 кГц          */
                        /* время преобразования: 32 ATD CLK,             */

                        /*1 считывание каждые 256 мкс                    /*

for(i= i<67; i++)       /*задержка 100 мкс при 8 МГц E_CLK              */
   {
   ;
   }
}

/*********************************************************************/
```

```
/*********************************************************************/
/*read_adc: считывание результатов из АЦП                            */
/*********************************************************************/

void read_adc()
{
ATDCTL5 = 0x50;      /*Установить АЦП на режим многоканального,*/
     /* преобразования 8 каналов                                     */
while((ATDSTAT & 0x8000)= =0)/* проверка бита SCF для окончания     */
     /*преобразования        */
   {
   ;
   }

                          /* сохранения результата в глобальном массиве     */
sens[0] = ADR7H;     /*дальний левый датчик                          */
sens[1] = ADR6H;     /*средний прапвый датчик                        */
sens[2] = ADR5H;     /*центральный датчик                            */
sens[3] = ADR4H;     /* средний правый датчик                        */
sens[4] = ADR3H;     /* дальний правый датчик                        */
sens[5] = ADR2H;     /*датчик Холла*/
}
/*********************************************************************/
/*decision(): решение о повороте основано на информации, полученной от*/
/* пяти датчиков. Порог датчика Холла (hes_threshold) и порог       */
/* оптического датчика (opto_threshold) определяются экспериментально. */
/*********************************************************************/

void decision()
{
if(sens[5] < hes_threshold){ /*  датчик Холла нашел "мину",         /*
   pwm_motors(back_up);      /* робот движется назад и определяются /*
     /* дальнейшие действия/*
   if(sens[0] > opto_threshold)
       pwm_motors(right_turn);
   else
       pwm_motors(left_turn);

   for(i=0; i<0xFFFF; i++){  /*задержка вращения двигателя          */
   for(j=0 J<15; J++)
      {
      ;
      }
   }
}

          /*если обнаруживает три стенки (тупик), то движется назад */
else if((sens[2]>opto_threshold)&&(sens[0]>opto_threshold)
        &&(sens[4]>opto_threshold))
      {
      pwm_motors(back_up);
      }
          /*если стенки спереди и слева, поворачивает направо      */
else if((sens[0]>opto_threshold)&&(sens[2]>opto_threshold))
      {
```

```
    pwm_motors(right_turn);
    }
            /*если стенки спереди и справа, поворачивает налево      */
else if((sens[2]>opto_threshold)&&(sens[4]>opto_threshold))
    {
    pwm_motors(left_turn);
    }
            /*если стенка спереди  справа, делает полуповорот направо*/

else if(sens[1]>opto_threshold)
    {
    pwm_motors(half_right);
    }

            /*если стенка спереди  слева, делает полуповорот налево  */

else if(sens[3] > opto_threshold)
    {
    pwm_motors (half_left) ;
    }

/*если стенки вблизи нет, продолжает движение вперед                */

else
    {
    pwm_motors(forward);}
    }
}
/*********************************************************************/
/*init_pwm(): инициализация ШИМ контроллера 68HC12                   */
/*********************************************************************/

void init_pwm()
{
PWTST= 0x00;
PWCTL= 0x00;          /*Режим фронтовой ШИМ                          */
WCLK=  0x3F;          /*Каналы без каскадного соединения, E_CLK/128  */
PWPOL= OxOF;          /*set pins high then low transition            */
DDRP = OxFF;          /*Порт PORT T на вывод                         */
PWEN = OxOF;          /*Активизировать выход ШИМ                     */
PWPERO = 250;         /*Установить частоту ШИМ 250 Гц                */
PWPER1 = 250;
PWPER2 = 250;
PWPER3 = 250;
PWDTY0 = 0;           /*начальная установка ШИМ на отсутствие движения*/
PWDTYl = 0;
PWDTY2 = 0;
PWDTY3 = 0;
}

/*********************************************************************/
/*pwm_motors: /*выполнение определенного поворота                    */
/*********************************************************************/
```

```
void pwm_motors(const char a)
{
for (i = 0;i<2000;i + +) /*задержка на 3 мс чтобы позволить двигателю*/
     {                                                    /* отреагировать*/
     }
switch(a){           /*определение вида поворота                    */

     case 0:                 /* движение вперед                     */
          PWDTY0 = 200;      /*регистры коэффициента заполнения ШИМ  */
          PWDTY1 = 250;
          PWDTY2 = 250;
          PWDTY3 = 200;
          lcd_print("Forward\n");
          break;

     case 1:                 /*полуповорот налево                   */
          PWDTY0 = 0;        /*регистры коэффициента заполнения ШИМ */
          PWDTY1 = 250;
          PWDTY2 = 250;
          PWDTY3 = 125;
          lcd_print  ("Half Left\n");
          break;

     case 2:                 /*полуповорот направо*/
          PWDTYO = 125;      /*регистры коэффициента заполнения ШИМ  */
          PWDTY1 = 250;
          PWDTY2 = 250;
          PWDTY3 = 0;
          lcd_print("Half Right\n");
          break;

     case 3:                 /*поворот налево*/
          PWDTYO = 125;      /*регистры коэффициента заполнения ШИМ  */
          PWDTY1 = 250;
          PWDTY2 = 0;
          PWDTY3 = 12 5;
          lcd_print("Left Turn\n");
          break;

     case 4:                 /*поворот направо*/
          PWDTYO = 125;      /*регистры коэффициента заполнения ШИМ  */
          PWDTY1 = 0;
          PWDTY2 = 250;
          PWDTY3 = 12 5;
          lcd_print("Right Turn\n");
          break;

     case 5:                 /*задний ход*/
          PWDTYO = 125;      /*регистры коэффициента заполнения ШИМ  */
          PWDTY1 = 0;
          PWDTY2 = 0 ;
          PWDTY3 = 125;
```

```
          for(i=0; i<0xFFFF; i++)
          {                    /* Задержка в 1,25 с перед движением назад*/
          for(j=0; j<15; J++)
               {
               ;
               }
          lcd_print ( "Back Up\n" ) ;
          break;

     default:        /*по умолчанию движение вперед, малая скорость */

          PWDTYO = 63;       /*регистры коэффициента заполнения ШИМ  */
          PWDTY1 = 250;
          PWDTY2 = 250;
          PWDTY3 = 63;
          lcd_print("Error\n");
          break;
   }
}
/*********************************************************************/
/*lcd_init(): инициализация режима работы ЖК дисплея                 */
/*Последовательность команд инициализации определяется изготовителем */
/*PORTA: магистраль данных, PORTB[2:1]: линия  R/S, линия разрешения E*/
/*********************************************************************/

void lcd_init()
DDRA=0xff; /*порт PORTA на вывод                                     */
DDRB=0xO6; /* порт PORTB [2:1] на вывод                             */
          /*последовательности команд для инициализации ЖК дисплея */

putcommand(0x38)
putcommand(0x38)
putcommand(0x38)
putcommand(0x38)
putcommand(0x0f)
putcommand(0x01)
putcommand(0x06)
putcommand(0x00)
                          /*очистка дисплея, возврат курсора      */
putcommand(0x00)
}
/*********************************************************************/
/*putchar(char c): вывод символа на дисплей                          */
/*********************************************************************/

int putchar(char c)
(
P0RTA=C;
PORTB= PORTB |0x04;
PORTB= PORTB |0x02;
PORTB= PORTB&Oxfd;
for (i=0; i<100; i++) ;      */задержка на *150 мкс до того, как ЖКД */
                          */ сможет принять информацию            */
return c;
}
/*********************************************************************/
```

```
/*********************************************************************/
/*putcommand(char c): выдача команды управления для ЖК дисплея      */
/*********************************************************************/

int putcommand(char c)
(

PORTA= c;
PORTB= PORTB&Oxfb;
PORTB= PORTB|0x02;
PORTB= PORTB&Oxfd;
for (i=0; i<100; i++)   */ задержка на *150 мкс до того, как ЖКД сможет*/
                        */принять информацию                          */
 {
 ;
 }

return c;
)
/*********************************************************************/
/*delay_25(): задержка на 2.5 с
/*********************************************************************/

void delay_25()
(
for(i=0; i<0xFFFF; i++)
    (
    for (j=0; j<30; j++)
    {
    ;
    }
   }
}

/*********************************************************************/
/*lcd_print(): вывод строки символов на дисплей.                     */
/*********************************************************************/

void lcd_print(char *string)
{
putcommand{0x02);    /*возврат курсора ЖКД    */

while (* {string) != '\0') {
    {
    putchar(*string);
    string++;
    }
}
/*********************************************************************/
/*********************************************************************/
```

7.2. Лазерный проектор

В этом разделе, мы описываем разработку и построение встроенной системы управления для лазерного проектора. Вы вероятно видели такую систему, на концерте или в планетарии. Аналогичный подход может быть использован при разработке точного лазера для медицинских целей или для технологических установок, например, чтобы управлять лазером для точной гравировки.

7.2.1. Описание проекта

Система имеет семь образцов изображений, которые проектируются лазером на стену или киноэкран. Мы выбираем один из них для проекции, нажав соответствующую кнопку. Как только изображение выбрано, подсвечивается светодиод, соответствующий выбранному варианту. Затем система управления открывает лазерный затвор, позволяя лазерному лучу пройти на пару гальванометрических зеркал. Микроконтроллер 68HC12 генерирует сигналы управления, позволяющие изменять угол поворота зеркал, чтобы создать предварительно записанное в памяти изображение с помощью внешних по отношению к МК цифро-аналоговых преобразователей (ЦАП). Выбранное изображение выводится однократно при каждом нажатии кнопки. Конструкция системы приведена на рис. 7.8.

7.2.2. Подсистемы 68HC12 используемые в проекте

Основываясь на кратком описании проекта, мы можем определить периферийные модули МК 68HC12 необходимо использовать и внешние устройства которые которые будут использоваться для решения нашей задачи:

- восемь двухпозиционных переключателей с аппаратной противодребезговой защитой, подключены к порту ввода МК;
- Восемь светодиодных индикаторов для логических выходов с тремя состояниями, подключены к порту вывода МК;
- Двухканальный ЦАП, связанный с МК по последовательному интерфейсу SPI, или два порта МК 68HC12, конфигурированных как выходные;
- Лазерный источник;
- Затвор и драйвер затвора;
- Два гальванометрических зеркала.

7.2.3. Описание некоторых компонентов системы

Как и прежде, рассмотрим доступные инструментальные средства из нашего инструментального ящика, позволяющие выполнить все требуемые функции.

В нашем сундучке инструментов, уже имеются следующие компоненты:

- Противодребезговые переключатели;
- Восемь светодиодных индикаторов с тремя состояниями;
- ЦАП;

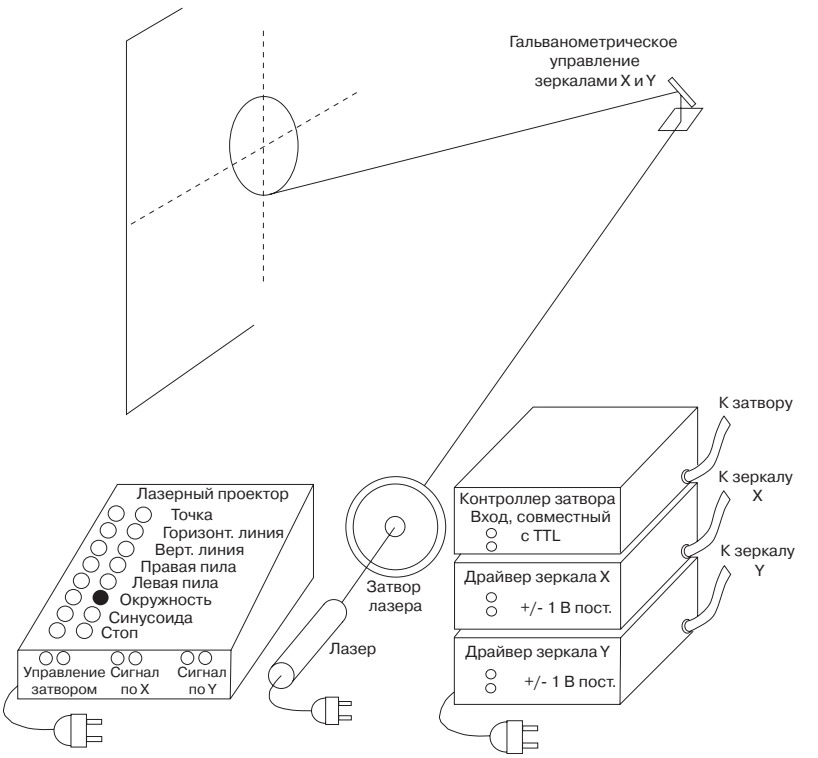


Рис.7.8. *Встроенная система управления лазенным проектором*

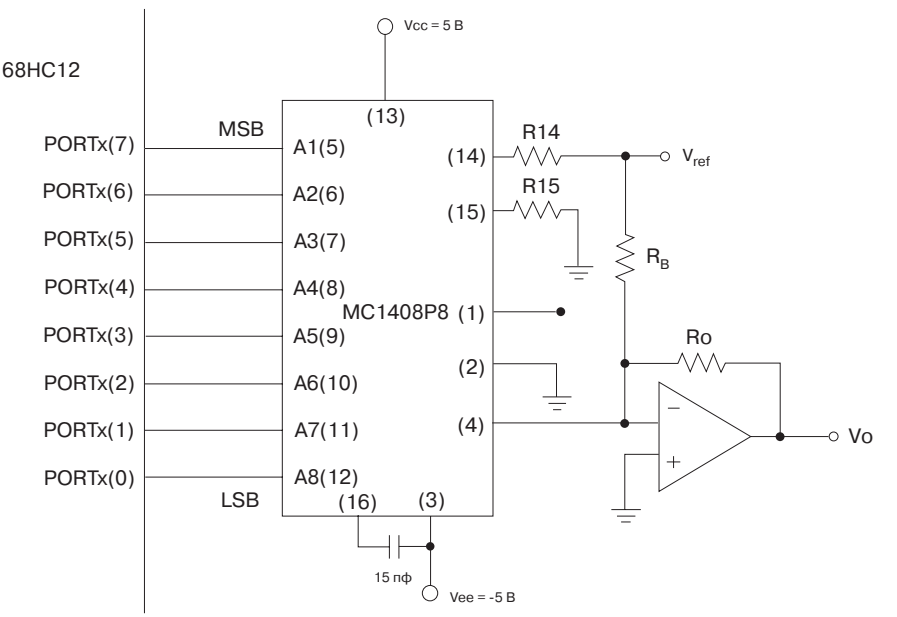
Однако мы еще не рассматривали лазеры, лазерные зеркала, лазерные затворы и гальванометрические зеркала. Рассматривают эти устройства. Мы рассмотрим также более подробно технологию ЦАП.

Цифро-аналоговые преобразователи (ЦАП). В гаве 6 мы обсуждали основы цифро-аналоговых преобразователей (ЦАП). Поскольку МК 68HC12 не имеют в своем составе модуля ЦАП, необходимо воспользоваться внешними ИС ЦАП. Для данного проекта нам нужны фактически два отдельных канала ЦАП, чтобы управлять X и Y каналами гальванометров. Существует много различныхазие ИС ЦАП, совместимых с 68HC12 и удовлетворяющих требованиям данного проекта. Их можно разделить на две категориии: ЦАП с последовательными или с параллельными входами. ЦАП с последовательными входами обычно подключается к МК с помощью интерфейса SPI. Читатель, интересующийся более подробным описанием этого типа интерфейса может обратиться книге Pack and Barrett [2002, гл. 10]. В этом примере, мы используем два 8-разрядных ЦАП с параллельными входами. Имеется широкое разнообразие таких ЦАП. В этом проекте мы используем ИС MC1408P8 фирмы Motorola. Типовая схема подключения ЦАП MC1408P8 к порту вывода МК показана на рис. 7.9. Выходное напряжение ЦАП определяется величиной опорного напряжения Vref, коэффициентом обратной связи операционного усилителя (определяется R14 и Ro) и цифровым кодом на входах A8...A1. Зависимость напряжения на выходе ЦАП в функции перечисленных параметров приведена на рис. 7.9. Величина опорного напряжения и номиналы резисторов оп-

ределяются схемой подключения и не могут быть изменены в процессе эксплуатации. А вот кодовая комбинация на входах A8...A1 постоянно изменяется в процессе управления. И в соответствии с передаточной характеристикой ЦАП изменяется напряжение на выходе V_0. Электрические характеристики цифровых входов ЦАП (A8...A1) позволяют выполнить их прямое подключение к выводам порта МК 68HC12. В соответствие с техническими условиями необходимо, чтобы напряжение V_0 изменялось в диапазоне ± 1 В при изменении кода на входах A8...A1 от $FF. Данное требование может быть выполнено при следующих номиналах резисторов схемы и опорного напряжения ЦАП:

- V_{ref} = 5,0 В
- R14=R15= 1 кОм
- Ro = (2/5) R14 = 400 Ом
- R_B = 2 (R14) = 2 кОм

Эти значения получены из решения уравнения для выходного напряжения, приведенного на рис. 7.9 для двух различных случаев: (1), когда выходное напряжение равно + 1 В, и (2), когда выходное напряжение составляет 0 В; опорное напряжение Vref удобно выбрать равным 5 В. Эти значения составляющих обеспечивают выходное напряжение –1,0 В для двоичного кода $00 и выходное напряжения 0,992 В для двоичного кода $FF. Двоичные коды, заключенные между этими крайними значениями кода, обеспечивают 256 значений аналогового выходного напряжения линейно изменяющегося от –1,0 до 0.992 В.



$$V_0 = \frac{V_{ref} \cdot (R_0)}{R_{14}}\left[\frac{A1}{2} + \frac{A2}{4} + \frac{A3}{8} + \frac{A4}{16} + \frac{A5}{32} + \frac{A6}{64} + \frac{A7}{128} + \frac{A8}{256}\right] - \frac{V_{ref} \cdot (R_0)}{R_в}$$

Рис.7.9. *Типичная схема подключения ЦАП MC1408P8 фирмы Motorola*

Лазеры. Лазер (слово получено из сокращения light amplification by stimulated emission of radiation – усилитель света на базе вынужденного излучения) представляет собой источник света с рядом специфических свойств и характеристик. Он, как считают, является монохроматическим (одна длина волны или очень узкий диапазон длин волн), когерентным (фронты волн находятся в фазе друг с другом), и нерасходящимся. Что же это означает? В основном, лазер обеспечивает одноцветный источник света с узким, подобным карандашу, лучом. Лазер с самого начала нашел применение фактически во всех областях промышленности и медицины [12]. Читателя, интересующегося более подробным знакомством с этой увлекательной темой, мы отсылаем к разделу «Что еще прочитать» в конце данной главы. Для рассматриваемого устройства, мы используем маломощный (менее 3 мВт) лазер в видимом диапазоне излучения. Лазеры этого типа доступны в нескольких различных исполнениях. Имеется ряд гелий - неоновых (HeNe) лазеров различных цветов. Однако газоразрядные трубки таких лазеров обычно имеют длину 25 см и диаметр 5 см. Более новая технология – твердотельный лазер с диодной накачкой (DPSS), маломощный лазер, для нескольких частот в видимой области. Его длина составляет приблизительно 5 см и диаметр 1,5 см, он питается от маленького выпрямителя [3]. Мы используем этот тип лазера для данного проекта. Эти лазеры достаточно просты в обращении. Вы подключаете их к сети, и сразу появляется луч.

При работе с лазерами применяется стандарт безопасности ANSI Z136. 1 «Безопасность при применении лазеров» [1], касающийся безопасной работы с приборами на базе лазеров. Если вы планируете реализовать подобный проект, мы советуем вам получить копию этого документа и подробно ознакомиться с ним. Стандарт делит лазеры на различные категории (классы от 1 до 4), основываясь на степени их опасности для пользователя. Чем выше номер класса, тем больше опасность. Лазер, который мы используем в этом приложении, принадлежит к классу 3а, обеспечивая мощность излучения 1...5 мВт в видимом диапазоне. Это тот же класс лазеров, к которому принадлежит лазерная указка. Хотя эта мощность относительно мала, обращаться с лазерами необходимо с особой осторожностью. Необходимо также предпринимать особые меры предосторожности при юстировке оптики, связанной с лазерной системой, чтобы предотвратить поражение глаз лазерным излучением. Ни в коем случае нельзя смотреть непосредственно в лазерный источник. Стандарт ANSI касается также других требований безопасности при использовании этого класса лазеров, включая предупредительные знаки, безопасное размещение лазера, обучение безопасному обращению с лазерами и ограничение доступа к ним.

Зеркала. Имеются оптические зеркала самых различных форм, размеров, и толщины, рассчитанные на различный диапазон частот излучения. Для этого применения, мы используем зеркала с нанесением покрытия на переднюю поверхность. Это означает, что зеркало имеет отражающее покрытие на наружной поверхности стекла. Такая технология предотвращает появление многократных отражений между передней и задней поверхностями зеркала. Кроме того, находящееся на передней поверхности отражающее покрытие должно быть рассчитано на соответствующую длину волны. То есть оно должно правильно отражать свет в интересующем нас диапазоне частот. Для этого приложения, мы используем лазер в видимом диапазоне (с длиной волны от 400 до 700 нм). Различные изготовители и поставщики обеспечивают широкий диапазон зеркал с различными покрытиями [3]. Оптические зеркала устанавливаются на вращающихся гальванометрических подвесках при помощи легких держателей.

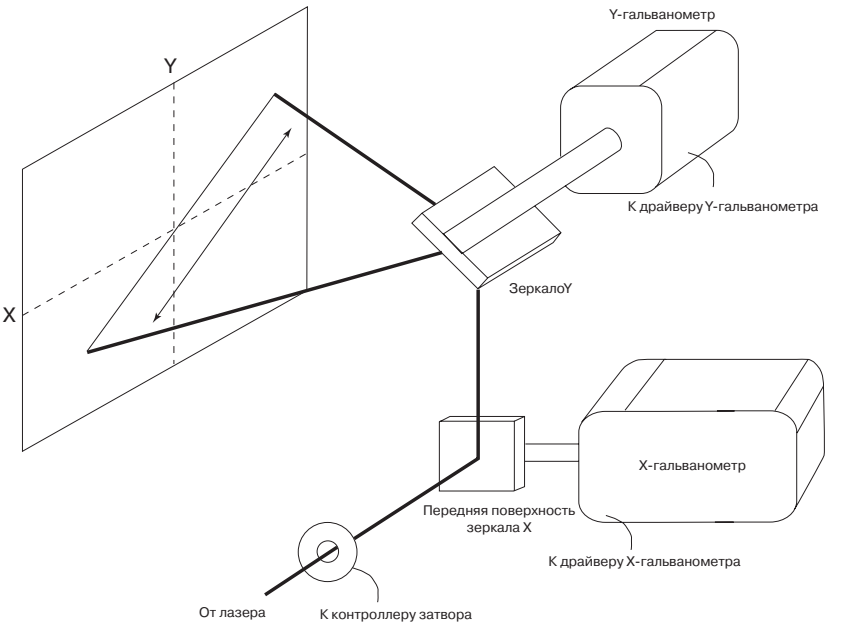


Рис. 7.10. *Сканирующая X-Y система*

Лазерные затворы. Лазерный затвор – это просто апертура для лазера, перекрывающая лазерный луч. В закрытом состоянии затвор обычно перекрывается створками. На створки наносится теплостойкое покрытие, способное выдерживать высокую плотность энергии лазера. Створки управляются драйвером лазерного затвора, имеющим логический вход, совместимый с транзисторно-транзисторной логикой (ТТЛ) и генерирующим выходной сигнал, согласованный с характеристиками затворов ряда изготовителей. Они имеют широкий диапазон диаметров апертуры: от 2 до 45 мм [13].

Гальванометры. В гальванометрах, называемых также оптическими сканерами, применяется эффективный метод перемещения лазерного луча. Гальванометры обеспечивают угол поворота зеркала точно соответствующий заданному значению входного напряжения. В идеале, должна быть обеспечена линейная связь между входным напряжением и углом поворота. Кроме того, гальванометры характеризуются максимальным и минимальным углами поворота. Легко приобрести гальванометры для углов до ±30 [4]. Гальванометры управляются внешними усилителями. Обычно это твердотельные усилители с переменным выходным сопротивлением. На их вход подается напряжение ± 1,0 В, при этом ток управляющий поворотом зеркала гальванометра пропорционален входному напряжению. Как правило, обеспечивается регулировка нулевого смещения и коэффициента усиления драйверов гальванометров.

На рынке имеются сканирующие X–Y системы, которые используют для оптического сканирования в плоскости X–Y два зеркала, и два отдельных гальванометра.

Зеркала помещены перпендикулярно друг к другу. Лазерный луч следует по пути, показанному на рис. 7.10. Управляющие сигналы для X- и Y-гальванометров формируются отдельными драйверами, и позволяют проецировать лазерное излучение в любую точку плоскости X–Y. Путем последовательного вывода ряда точек могут быть созданы различные образы.

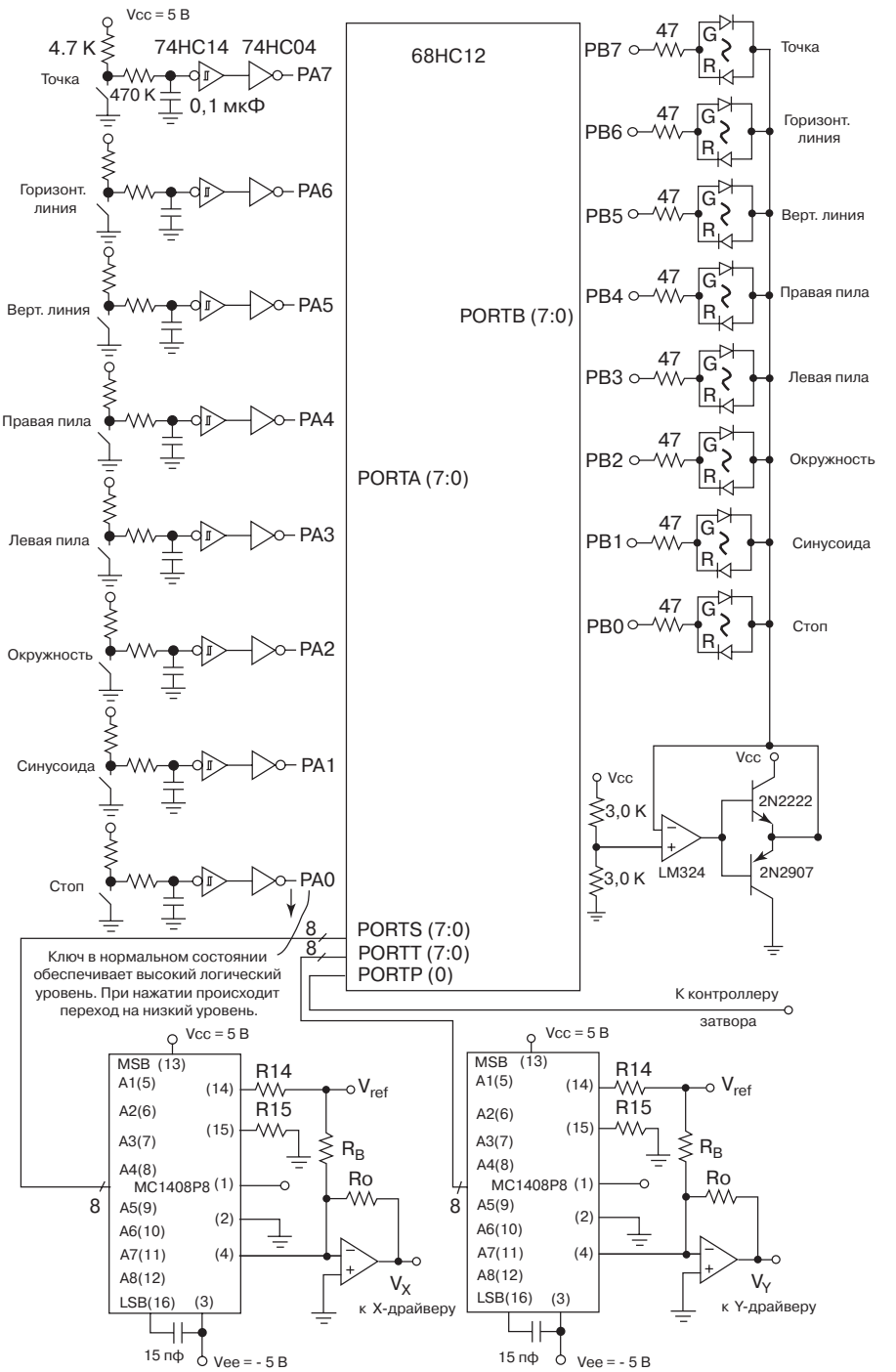


Рис.7.11. *Внешний аппаратный интерфейс для 68HC12*

7.2.4. Аппаратные средства

Функциональная схема управление лазерным проектором приведена на рис. 7.11. На первый взгляд эта схема может показаться сложной. Однако, Вы уже знакомы с каждой из ее подсистем, что облегчает понимание общего устройства системы. Линейка из восьми противодребезговых переключателей, подключенных к порту PORTA микроконтроллера 68HC12, используется для выбора образа, которое должно быть воспроизведено лазером. Когда образ выбран, подсвечивается соответствующий ему светодиод на PORTB. Например, если мы нажимаем переключатель, связанный с выводом PORTA[4], должно выводиться изображение правого пилообразного сигнала. При этом начинает светиться светодиод, связанный с PORTB[4].

Чтобы показать, какое из изображений было выбрано, используется линейка светодиодных индикаторов. Когда изображение выбрано, соответствующие двоичные значения появляются на выходе порта PORTS[7:0] для X-гальванометра и порта PORTT[7:0] для Y-гальванометра. Они подаются на ЦАП для X-канала и ЦАП для Y-канала, соответственно. ЦАП преобразует каждое из двоичных значений в аналоговый сигнал, способный управлять усилителями гальванометров.

Сигнал управления, открывающий и закрывающий створки затвора, выводится на порт PORTP[0]. Логическая «1» открывает створки, а логический «0» закрывает их. Как и для гальванометров, для створок необходимо преобразовать логический сигнал контроллера 68HC12 в аналоговый сигнал, достаточный для того, чтобы открыть или закрыть створки. Контроллер принимает ТТЛ совместимый входной сигнал, и преобразует его в сигнал, способный управлять приводом створок.

Завершив на этом обсуждение аппаратных средств, мы подробно рассмотрим в следующем разделе программное обеспечение для управления данной системой.

7.2.5. Структура программы и блок-схема алгоритма

Структура программы и блок-схема алгоритма для системы управления лазерным проектором представлены на рис. 7.12. Обе диаграммы довольно очевидны. Диаграмма интерфейса приводится, чтобы показать преобразование двоичного кода, поступающего от микроконтроллера 68HC12, в аналоговый сигнал ± 1 В, необходимый для усилителей гальванометра.

Система сначала инициализируется, устанавливая порты в необходимый режим (вход или выход) и закрывая лазерный затвор. Затем программа считывает информацию из порта PORTA, чтобы определить, нажат ли переключатель. Если он нажат, используется команда переключателя, позволяющая определить выбранное изображение. Код, необходимый для создания данного изображения обеспечивается выбором переключателя. Чтобы создать изображение, лазер сначала перемещается в требуемую исходную позицию для заданной траектории. Затем открывается лазерный затвор, и соответствующая траектория обеспечивается, последовательностью положений лазерного луча в различные моменты времени. Мы приводим несколько примеров, и затем предлагаем вам самостоятельно, сформировать оставшиеся варианты в качестве домашней работы (задание 14).

7.2.6. Программный код

```
//*****************************************************************
//имя файла: laser.c
//функции: программа для управления лазерным проектором
//контроллер: отладочная плата 68HC12B32 фирмы Motorola
//Выводы отладочной платы микроконтроллера 68HC12B32 фирмы Motorola:
//Port A: Конфигурирован как входной порт, активируемый нажатием
//        бесдребезговых переключателей на каждом входе
//Port B: Конфигурирован как выходной порт для управления светодиодным
//        индикатором
//Port S: Конфигурирован как выходной порт, создающий двоичный код
//        на канале X ЦАП
//Port T: Конфигурирован как выходной порт, создающий двоичный код
//        на канале Y ЦАП
//Port P[0]: Конфигурирован как выходной порт, создающий TTL совместимый
//        сигнал для управления затвором
//авторы: Steve Barrett and Daniel Pack
//создан: февраль 20, 2003
//последняя редакция: март 3, 2004

//*****************************************************************
//включенные файлы
//*****************************************************************
#include<912b32.h>

//функции прототипов
void initialize_ports(void); //инициализация портов
void shutter(int);           //открытие/закрытие створок
void position_laser(unsigned char, unsigned char); // положение лазера
void delay(void);

#define open 1
#define close 0

//main program*****************************************************

//global variables
unsigned char new_PORTA, old_PORTA = 0xFF;
int i ;
int go;

void main(void)
{
//инициализировать вектор reset B32
asm(" .area vectors(abs)\n"
     ".org 0xFFF8\n"
     ".word 0x8 0 00, 0x8000, 0x8000, 0x8000\n"
     " .text");

go = 1;         //начало цикла while
initialize_ports();          //инициализировать порты
shutter (close);             //закрыть створки
position_laser(0x80,0x80);   //расположить луч в центре
while(go){      //продолжать, пока не нажмут кнопку Stop
```

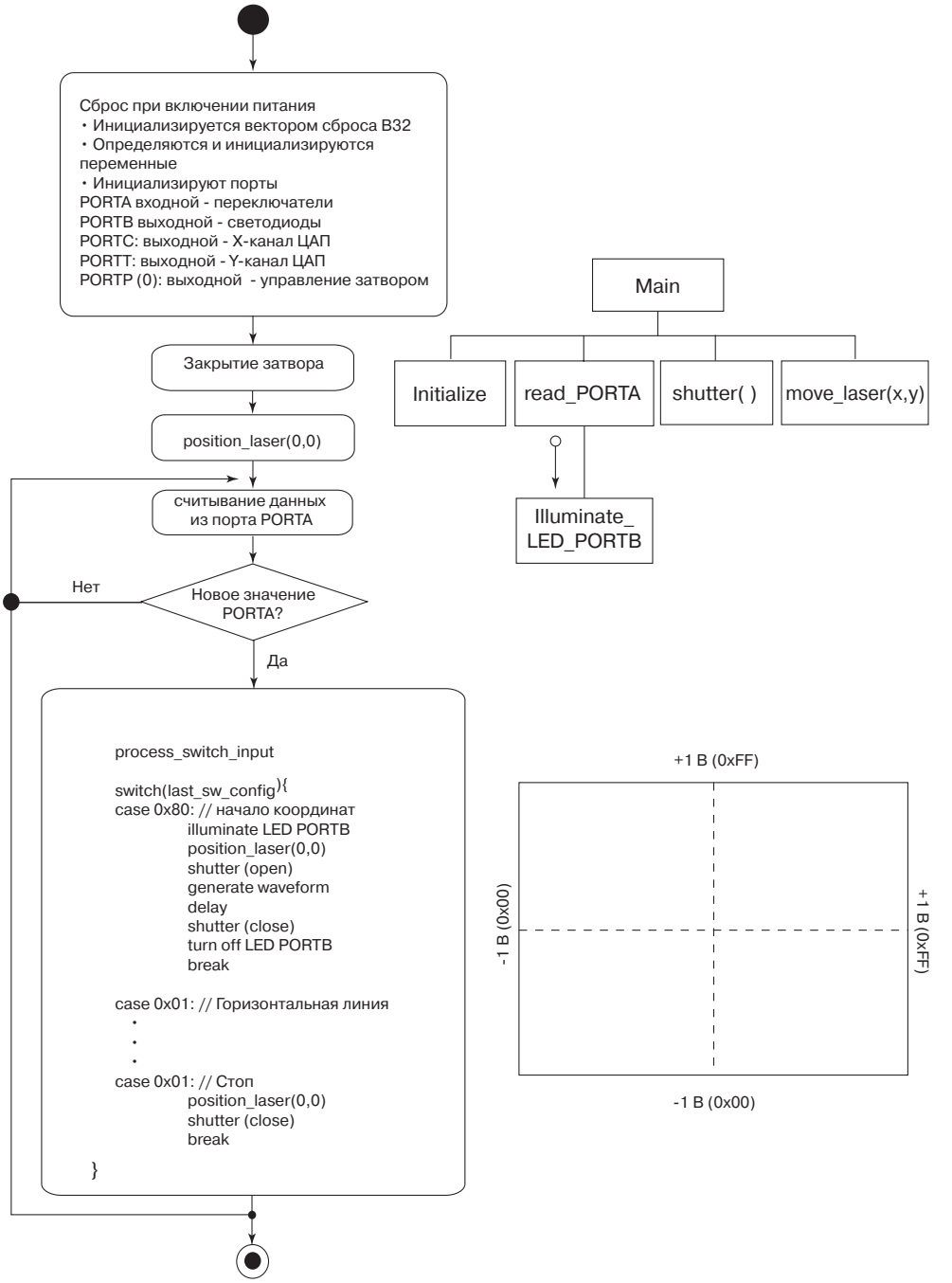


Рис. 7.12. *К программе управления лазерным проектором: блок-схема алгоритма (слева), структура программы (вверху с права), шестнадцатеричные значения к диаграмме ЦАП (внизу справа)*

```
new_PORTA = PORTA; //read PORTA input switches
if(new_PORTA != old_PORTA)
switch(new_PORTA){           //формировать образ по значению
            //нажатой клавиши
      case 0x7F:      //PA7 - Точка в центре экрана
            PORTB = 0x80;// подсвечивает светодиод порта PORTE
            position_laser(0x80,0x80);
            shutter(open);
            delay();
            shutter(close);
            PORTB=0x00; //включает красные светодиоды индикатора
            break;

      case OxBF:      //PA6- горизонтальная линия
            position_laser(0x00,0x80);
            shutter(open);
            for{i=0; i<=255; i++)
                  {
                  i = (unsigned char)(i);
                  position_laser(i, 0x80);
                  delay();
                  }
            shutter(close);
            PORTB-0x00;  //включает красные светодиоды индикатора
            break;

      case OxDF:              //PA5 - Вертикальная линия
            PORTB = 0x20; // подсвечивает светодиод порта PORTB
            position_laser(0x80,0x00);
            shutter(open);
            for(i=0; i<=255;i++)
                  {
                  i=(unsigned char)(i) ;
                  position_laser(0x80,i);
                  delay();
                  }
            shutter(close);
            PORTB=0x00;  //включает красные светодиоды индикатора
            break;

      case 0xFF:              //PA4 - Правая пила под углом 45 градусов
                  // с ЮгоЗапада на СВ
            PORTB = 0x10;  // подсвечивает светодиод порта PORTB
            position_laser(0x0 0,0x00);
            shutter(open);
            for(i=0; i<=255; i++)
                  {
                  i = (unsigned char)(i);
                  position_laser(i, i);
                  delay();
                  }
            shutter(close);
            PORTB=0x00; //включает красные светодиоды индикатора
            break;

      case 0xF7:      //PA3 - Левая пила под углом 45 градусов
                  //с ЮВ на СЗ
```

```
            PORTB = 0x08; // подсвечивает светодиод порта PORTB
            delay();

            PORTB=0x00;  // включает красные светодиоды индикатора
            break;

      case 0xFB:      //PA2 Окружность
            PORTB = 0x04;      // подсвечивает светодиод порта PORTB
            delay();
            PORTB=0x00;        //включает красные светодиоды индикатора
            break;

      case 0xFD:      //PA1 Синусоида
            PORTB = 0x02;      // подсвечивает светодиод порта PORTB
            delay{);
            PORTB=0x00;        // включает красные светодиоды индикатора
            break;

      case 0xFE:      //PA0 Остановка
            PORTB = 0x01; // подсвечивает светодиод порта PORTB
            position_laser(0x0 0,0x00); shutter(close);
            delay();
            PORTB=0x00;  // включает красные светодиоды индикатора
            go = 0;
            break;

case 0xFF:   break;
      default:   ;     //все остальные случаи

      }//конец switch(new_PORTA)
      old_PORTA = new_PORTA;
     }//конец if(new_PORTA ! = old_PORTA)
   }//конец while(go)
}//конец main

//*********************************************************************
// initialize_ports: производится конфигурация портов в качестве
//входных/выходных
**********************************************************************

void initialize_ports(void)
{
DDRA=0x00;    //установить PORTA в качестве входного порта
DDRB=0xFF;    //установить PORTB в качестве выходного порта
PORTB=0x00;   //включить красные светодиоды индикатора
DDRS=0xFF;    //установить PORTT в качестве выходного порта
DDRP=0xFF;    //установить PORTP в качестве выходного порта
}

//*********************************************************************
/// /shutter(int action) :  открытие и закрытие затвора
//*********************************************************************

void shutter(int action)
{
if (action = = open)
PORTP = 0x01; if
```

```
(action == close)
PORTP = 0x00;
}

//*********************************************************************
//position_laser(unsigned char x_pos, unsigned char y_pos): посылает
//сигнал управления для каналов X и Y  гальванометра из портов PORTS
//и PORTT соответственно.
//*********************************************************************
void position_laser(char x_pos,char y_pos)
{
PORTS = x_pos;
PORTT = y_pos;
}

//*********************************************************************
//delay(void): создает задержку
//*********************************************************************

void delay(void)
{
int j ;
for(j=0x0000;j<0xl000;j=j+0x0l)
     {
     asmf"пор");
     }
}
//*********************************************************************
//*********************************************************************
```

7.2.7. Испытания устройства

До подсоединения компонентов системы к МК 68HC12, мы должны полностью проверить схему. В главе 5 мы рассматривали методики проверки, позволяющие моделировать входы системы переключателями, а выходы светодиодами. Наше устройство уже содержит переключатели и светодиоды для такой проверки. Однако, как мы проверим аналоговые сигналы? Имеется два метода, позволяющих легко проверить связь друг с другом сигналов на каналах X и Y:

1) использование перьевого X-Y графопостроителя,

2) использование классической контрольно-измерительной методики, связанной с получением так называемых фигур Лиссажу.

При первой методике, выходной аналоговый сигнал, формируемый X-каналом ЦАП, переключается с X-канала гальванометра на X-канал перьевого X-Y графопостроителя, а сигнал с Y-канала гальванометра на Y-канал на графопостроителя. Сигнал управления затвором может быть подан на драйвер пера графопостроителя, смещающийся вверх и вниз. Необходим плоттер со специальными характеристиками, чтобы определить, требуется ли схема интерфейса между ТТЛ совместимым сигналом для управления затвором от 68HC12 и управления движением пера вверх и вниз. После подключения микроконтроллера 68HC12 к графопостроителю, каждое из изображений может быть полностью проверено.

Вторая методика испытаний использует классический метод фигур Лиссажу. Чтобы получить фигуры Лиссажу, выходные сигналы с X-канала и Y-канала ЦАП подаютс на соответствующие каналы осциллографа.

После подключения 68HC12 к осциллографу, каждое из изображений также может быть полностью проверено. Дополнительную информация о фигурах Лиссажу можно найти в [2].

7.2.8. Заключительные испытания системы управления

После полной проверки программного обеспечения, оно может быть испытано совместно с устройством управления лазером. Реальные устройства описываются в литературе, выпускаемой изготовителями оптических устройств и в учебниках, посвященных оптическим блокам [3, 8].

7.3. Цифровой вольтметр

7.3.1. Описание проекта

Для этого проекта мы должны разработать цифровой вольтметр (ЦВ), способный измерять входной аналоговый сигнал в диапазоне от +10 до –10 В. Измеряемое напряжение, отображается на ЖК дисплее, число знакомест которого позволяет отображать числа от 0 до 100.

Диапазон входных измеряемых напряжений для модуля аналого-цифрового преобразрвания ATD МК 68HC12 составляет 0...5 В. Чтобы измерить входное напряжение в более широком диапазоне, необходим внешний интерфейс. Поэтому мы преобразуем входной сигнал ±10 В в сигнал, лежащий в диапазоне от 0 до 5 В.

7.3.2. Системы 68HC12 используемые в проекте

Для реализации проекта мы должны будем использовать следующие модули в составе МК 68HC12, внешние устройства и программы управления:

- Модуль ATD в составе МК 68HC12;
- Символьный ЖК индикатор;
- Интерфейс преобразователя диапазона входного сигнала;
- Алгоритм преобразования измеряемого напряжения в ASCII код для отображения на ЖК индикаторе.

Прежде чем разрабатывать программное обеспечение, рассмотрим некоторые аппаратные решения.

7.3.3. Расчет интерфейса модуля ATD

В разделе 5.9 мы описали, как подключить аналоговое устройство ввода данных к МК 68HC12, использовав методику расчета интерфейса преобразователя. Мы мо-

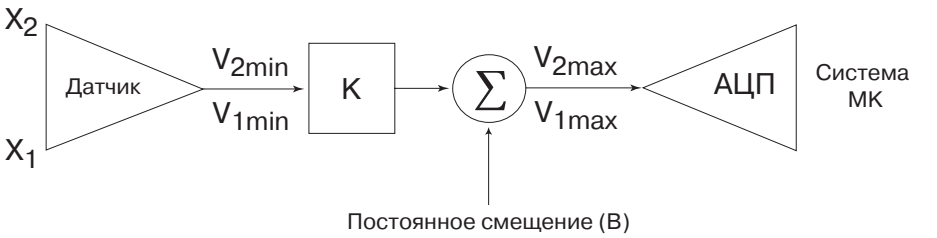


Рис. 7.13. *К расчету согласующего устройства*

жем применить этот материал для разработки устройства, изменяющего диапазон входного напряжения от исходного ± 10 В до диапазона от 0 до 5 В, совместимого с подсистемой аналого-цифрового преобразования для 68HC12.

Структурная схема согласующего устройства приводится на рис.7.13. Это устройство должно сформировать напряжение 5 В на входе АЦП микроконтроллера, когда на вход вольтметра подается 10 В, и 0 В на входе АЦП при напряжении в –10 В на входе вольтметра. Чтобы выполнить такое преобразование, входной сигнал должен быть умножен на масштабирующий коэффициент, и, кроме того, должно быть создано напряжение смещения. В нашей схеме операцию масштабирования выполняет блок К, и его выходной сигнал суммируется с сигналом смещения В.

По рассмотренной ранее методике составим два уравнения с двумя неизвестными, чтобы описать работу интерфейса преобразователя нашего проекта:

$$V_{2max} = V2min * K + B$$

$$V_{1max} = V1min * K + B$$

Нетрудно установить, что $V_{1min} = -10$ В, а $V_{2min} = +10$ В, в то время как $V_{1max} = 0$ В, и $V_{2max} = 5$ В. Подставим эти значения в нашу систему уравнений:

$$5 = 10 * K + B$$

$$0 = (-10) * K + B$$

В результате решения системы получим масштабный множитель K = 0.25, и напряжение смещения B = 2.5 В. Создадим схему на ОУ с коэффициентом передачи 0.25, и добавим напряжение смещения в 2.5 В.

При работе вольтметра после преобразования входного напряжения встроенным в МК АЦП, мы должны выполнить пересчет кода оцифровки, чтобы получить фактически измеренное входное напряжение для вывода его на дисплей. Эта операция выполняется с помощью программного обеспечения.

7.3.4. Структура программы и блок-схема алгоритма

Мы приводили структуру программы и блок-схему алгоритма для каждого из рассматриваемых проектов. Для этого проекта мы оставляем разработку структуры и блок схемы читателю в качестве домашней работы (задание 17).

7.3.5. Программа управления

```
/*****************************************************************/
/* Имя файла: voltmeter2.c                                       */
/* Это программа для реализации простого вольтметра на базе АЦП, */
/* встроенного в МК HC12. Приведенный программный код выполняет  */
/*одно преобразование и затем программа может вручную            */
/* перезапускаться пользователем для змерения другого напряжения */
/*****************************************************************/
#include <912b32.h>
#include <stdio.h>

/*функции-прототипы*/
void delay_100us(void);
void ADC_convert(void);
void delay_5ms(void);

void main(void)
{

asrn(" .area vectors (abs) \n" /*код инициализации вектора reset B32 */
     " .org 0xFFF8\n"
     "1 .word 0x800 0, 0x8000, 0x8000, 0x8000\n"
     " .text");

initialize_LCD();    /*инициализация ЖК дисплея                    */
 ATDCTL2 = 0x80;     /*подача питания на АЦП, разрешение прерываний */
delay_5ms(); /*ожидание входа АЦП в рабочий режим                  */
ATDCTL3 = 0x00;
ATDCTL4 = 0x01; /*8-разрядный результат, время выборки 2 АЦП      */
            /*clk, коэффициент деления 4                          */
ADC_convert();   /*АЦП реобразование                              */
}
/*****************************************************************/
/* void ADC_convert(void): функция, осуществляющая одно преообразо- */
/* вание и сохраняющая доступ пользователя к нему. Затем функция  */
/* преобразует текущий результат в цифровое значение. Таким образом */
/* каждое отдельное число может быть выделено, преобразовано в ASCII*/
/* код и выведено на ЖКД                                         */

void ADC__convert ()
{
unsigned int sumadr;
unsigned int avg__bin_voltage;
unsigned int int_voltage;
unsigned int tens_int;
unsigned int ones__int;
unsigned int tenths_int;
unsigned int hundreths_int;
double   voltage, abs_voltage;
char tens;
char ones;
char tenths;
char hundreths;
ATDCTL5 = 0x06; /*проводится 4 преобразования, канал 6           */
```

```
while((ATDSTAT & 0x8000)!= 0x8000)
     {       /*Подождите окончания преобразования                    */
     }
             /*усреднение по четырем результатам                     */
sumadr = ADR0H + ADR1H + ADR2H + ADR3H;
avg_bin_voltage = sumadr/4;
/* преобразование результата в напряжение, лежащее в диапазоне от   */
/* 0.00 до 5.00 В                                                    */
voltage = (avg_bin_voltage/256)*5;

/*приведение напряжения к диапазону от  -10.00 до +10.00 В          */
/*обращение процесса, выполняемого аналоговым интерфейсом           */
abs_voltage = (fabs)((voltage - 2.5) * 4);

/*преобразование результата в целое число в диапазоне от  -1000     */
/* до +1000          */
int_voltage = (100*voltage);

/*Выделение и преобразование наибольшей значащей цифры в            */
/* ASCII код десятичного значения, прибавление 48,                   */
/* результат дает ASCII код                                          */
tens_int = int_voltage/1000;
tens = (char)(tens_int + 48);

/*Выделение и преобразование следующей наибольшей значащей цифры в  */
/* ASCII код десятичного значения, прибавление 48,                   */
/* результат дает ASCII код*/

ones_int - int_voltage/100;
ones = (char)(ones_int + 48);
/*Выделение и преобразование  следующей наибольшей значащей цифры в*/
/* ASCII код десятичного значения, прибавление  48,                  */
/* результат дает ASCII код                                          */
tenths_int = (int_voltage - ones_int*100)/10;
tenths = (char)(tenths_int + 48);

/*Выделение и преобразование  следующей наибольшей значащей цифры в*/
/* ASCII код десятичного значения, прибавление  48,                  */
/* результат дает ASCII код
          */
hundreths_int = (int_voltage - ones_int*100 - tenths_int*10)/1;
hundreths = (char)(hundreths_int + 48);

/*Вывод результата на ЖКД                                           */
if (voltage < 0)
     putchars{'-'); /*Вывести отрицательный знак
else
     putchars('+'); /*Вывести положительный знак
putchars(tens);
putchars(ones) ;
putchars('.');
putchars(tenths);
putchars(hundredths);
putchars(' ');
putchars ( 'V );
}
/*******************************************************************/
```

```
/*********************************************************************/
/*задержка в 100 мкс, на базе  таймера с частотой 8 МГц             */
/*********************************************************************/

void delay_100us(void]
{
int i;

for (i=0; i<50; i++
     {
     asm("пор");
     }
}

/*********************************************************************/
/*задержка в 5 мс, на базе  таймера с частотой 8 МГц */
/*********************************************************************/

void delay_5ms(void)
{
int i ;
for (i=0; i<50; i++)
     {
     delay_100us();
     }
}

/*********************************************************************/
/*Функции инициализации посылают на ЖКД необходимую стартовую       */
/* последовательность. Формируется последовательность команд инициали- */
/* зации соответствующих техническим данным производителя дисплея.  */
/*********************************************************************/

void initialize_lcd(void)
{
delay_5ms();
delay_5ms();
delay_5ms(); /*задержка на 15 мс перед включением ЖКД               */

putcommands(0x38);   /*установочный интерфейс                       */
delay_5ms();
putcommands(0x38);
delay_100us();
putcommands(0x38)
putcommands(0x38)
putcommands(OxOC)
putcommands(0x01)    /*Очистить дисплей                             */
putcommands(0x06)    /*Установить режим увеличения адреса на единицу*/
putcommands(OxOE)    /*Включить дисплей,вывести мигающий курсор     */
putcommands(0x02)    /*Возврат                                      */
}
/*********************************************************************/
/* Функция вывода инициализирует порт данных, создает сигналы RS и  */
/* разрешения и посылает их на соответствующий порт                  */
**********************************************************************/
```

```
void putchars(unsigned char c)
{
DDRP =OxFF;                    /*установить Port  P как выходной      */
DDRDLC= DDRDLC OxOC;           * установить PORTDLC[3:2] как выходной */
PORTP = c;                     /* присвоить знак С порту данных       */
PORTDLC= PORTDLC|0x08          /* установить RS в 1 для данных        */
PORTDLC= PORTDLC|0x04          /* установить E в 1 ( PORTDLC[5] = 1)  */
PORTDLC = 0;                   /* установить E и RS в 0               */
delay__5ms();                  /* подождать 5мс                       */
}
/*********************************************************************/
/*Функция putcommand пересылает данные в контроллер на ЖК дисплея   */
/*********************************************************************/

void putcommands(unsigned char d)
{
DDRP = OxFF;          /*установить порт  PORTP в качестве выходного  */
DDRDLC = DDRDLC|OxOC;         /*установить выводы PORTDLC[3:2]       */
                              /* в качестве выходных                 */
PORTDLC = PORTDLC & 0xF7;     /* команда RS = 0                      */
PORTP = d;                    /*передача команды на  на ЖКД          */
PORTDLC = PORTDLC|0x04;       /*E = 1                                */
PORTDLC =0;                   /*E = 0                                */
delay_5ms();                  /*пауза 5 мс                           */
}

/*********************************************************************/
/*Функция lcd_print посылает строку на ЖК дисплей                    */
/*********************************************************************/

void lcd_print(char *string)
{
putcommands(0x02);            /*установка курсора на первую строку   */
                              /*putcommand для выделения строки      */
     while(*(string) != '\0')
          {
          putchars(*string);
          string++;
          }
}

/*********************************************************************/
/*задержка в 5 мс, на базе  таймера с частотой 8 МГц                 */
/*********************************************************************/

void delay_5ms(void)
{
int i ;

For(i=0; i<50; i++)
     {
     delay__100us() ;
     )
)

/*********************************************************************/
```

```
/*****************************************************************/
/*задержка в 100 мкс, на базе  таймера с частотой 8 МГц          */
/*****************************************************************/

void delay_100us(void)
{
int i;
for(i=0; i<50; i++)
      {
      asm("nop");
      }
}
/*****************************************************************/
```

7.3.6. Измерение неэлектрических величин

В представляемых примерах функция вольтметра состоит просто в измерении напряжения с выхода датчика некоторого внешнего физического параметра.

Датчик температуры. Например, мы можем подключить к МК 68HC12 градуированный по шкале Фаренгейта прецизионный датчик температуры LM34, выпускаемый компанией National Semiconductor. Этот датчик имеет постоянный коэффициент преобразования в + 10 мВ на градус Фаренгейта, в диапазоне от –50 до + 300 Ф. Схема подключения LM34 приведена на рис.7.14. Она состоит из собственно датчика LM34 и цепи фильтра, образованной резистором и конденсатором. Такая схема обеспечивает прямое преобразование измеряемой температуры. Например, при 70 Ф, LM34 создает выходной сигнал в 700 мВ. Это напряжение умножается на 100, чтобы обеспечить прямое преобразование напряжения в вольтах в температуру, выраженную в градусах Фаренгейта для вывода на дисплей. Это значение должно затем быть преобразовано в ASCII код для вывода на ЖК дисплей.

Датчик влажности. Фирма Honeywell производит ряд датчиков влажности (Humidity/Moisture) HIH-3610 [5]. Эти датчики калиброваны при питающем напряжении 5 В. Датчики обеспечивают линейный выход постоянного напряжения от 0,8 к 4,07 В при относительной влажности, изменяющейся от 0 до 100%, соответственно. Эти датчики могут быть связаны со встроенным АЦП МК 68HC12 непосредственно без согласующей схемы.

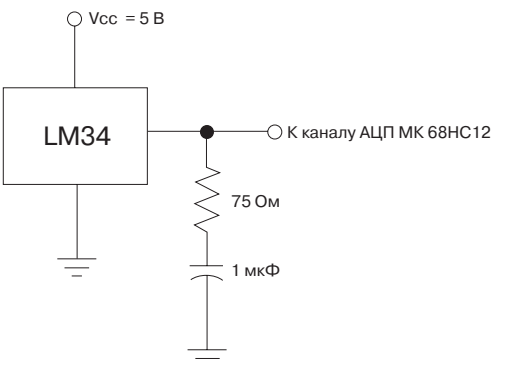


Рис. 7.14. *Интерфейс для прецизионного датчика температуры по шкале Фаренгейта LM34 компанаии National Semiconductor*

7.4. Стабилизация скорости вращения двигателя с использованием оптического тахометра

7.4.1. Описание проекта

Целью этого проекта является стабилизация скорости вращения электродвигателя. Чтобы осуществить такую стабилизацию, используем управление с обратной связью. Мы будем постоянно контролировать скорость вращения двигателя в рабочем режиме и корректировать ее, измененяя напряжение питания, подводимое к двигателю. Мы также будем показывать текущую скорость вращения двигателя, выраженную в оборотах в минуту, на ЖК индикаторе.

В проекте используется двигатель постоянного тока с постоянным магнитом, выпускаемый компанией Electro-Craft Corporation и оптический кодер [11]. Двигатель имеет следующие характеристики:

- Постоянное питающее напряжение: 12 В;
- Скорость на холостом ходу: 2500 об/мин при 12 В;
- Пусковой ток: 2 А;
- Ток холостого хода: 370 мА;
- Ток при умеренной нагрузке: 600 мА;
- Ток останава: 4 А;
- Двигатель имеет один канал оптического кодера (Servo-Tek # PMBX-60-05). Кодер формирует 60 ТТЛ совместимых импульсов за один оборот двигателя.

После этого краткого описания проекта возникает ряд вопросов:

1. Как контролировать скорость двигателя?
2. Каким образом регулировать напряжение питания двигателя, чтобы изменять его скорость?
3. Микроконтроллер 68HC12 питается от постоянного напряжения в 5 В при очень малом выходном токе источника. Как управлять с его помощью двигателем с высоким рабочим током, питающимся от постоянного напряжения в 12 В?
4. Как можно совместить управление скоростью двигателя с выполненем других задач, стоящие перед системой?
5. Какие системы, встроенные в 68HC12, должны использоваться, чтобы выполнить эту задачу?

Как и прежде, мы обратимся сначала к нашему «сундучку инструментов», чтобы выяснить, какие инструментальные средства мы уже использовали в проектах и какие придется разрабатывать. Давайте сделаем это одновременно с рассмотрением каждого из сформулированных выше вопросов. При обсуждении воспользуемся схемой регулирования частоты вращения двигателя, представленной на рис. 7.15.

1. Как контролировать скорость двигателя? Как было упомянуто, мы используем для измерения скорости оптический кодер, формирующий 60 импульсов за оборот двигателя.

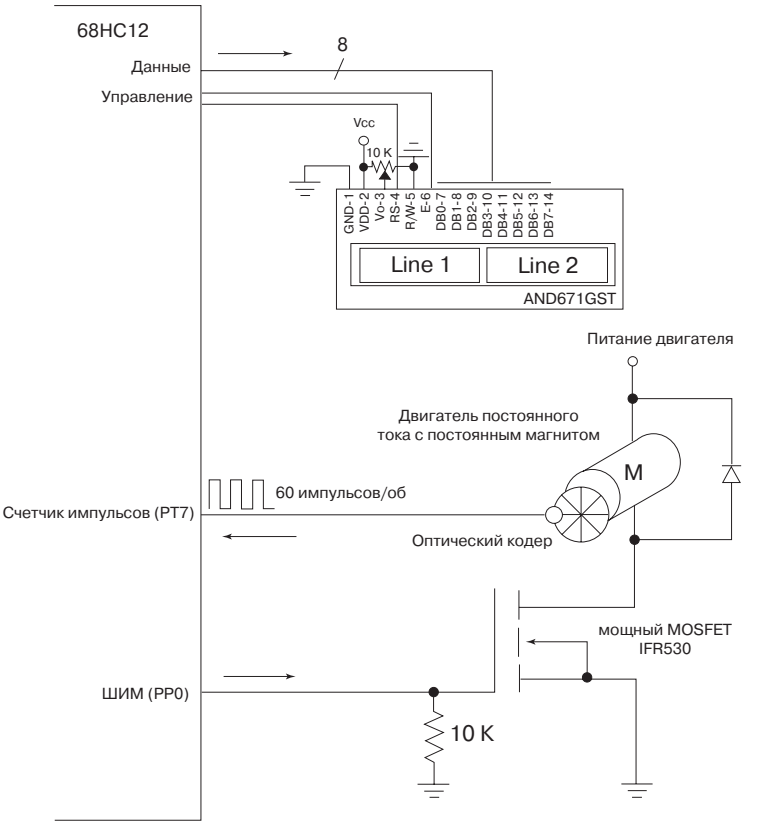


Рис. 7.15. *Схема стабилизации скорости вращения двигателя*

Подсчитав число импульсов на фиксированном временном интервале, можно определить скорость двигателя. Число импульсов, появляющихся внутри данного временного интервала, можно измерить, используя систему прерываний МК 68HC12 в режиме реального времени (RTI) и аккумулятор импульсов (PA). Обе эти системы обсуждались в главе 4. Как мы уже упоминали, система RTI генерирует прерывания через равные временные интервалы, определенные пользователем, чтобы «напоминать» МК о необходимости периодически выполнять задачу стабилизации, а аккумулятор PA может быть конфигурирован, чтобы считать импульсы. Объединив эти системы, мы сможем подсчитать число импульсов оптического кодера, появляющихся внутри данного временного интервала и, затем вычислить скорость вращения двигателя в оборотах в минуту.

Рассмотрим конкретный пример. Предположим, что мы конфигурировали RTI на прерывание через каждые 32.768 мс, и в течение этого интервала, PA зафиксировал появление 52 импульсов. Какова будет при этом скорость вращения двигателя в оборотах в минуту? Чтобы вычислить необходимо выполнить следующий перевод единиц измерения:

(52 импульса/32.768 мс) (l об/60 импульсов) (l, 000 мс/1 секунда) (60с/1 мин)
= 1,586 об/мин

2. Каким образом регулировать напряжение питания двигателя, чтобы изменять его скорость?

В главе 4 мы обсуждали концепцию широтно-импульсной модуляции (ШИМ).

Было показано, что среднее значение напряжения на двигателе изменяется при изменении коэффициента заполнения питающего его импульсного напряжения. Коэффициент заполнения определяется как отношение времени включенного состояния ключа к периоду переключения. Если мы используем ШИМ, то сможем изменять питающее напряжение и, как следствие, скорость двигателя. Примем для простоты, что скорость двигателя линейно связана с питающим напряжением. Проверим это предположение экспериментально. Сигналы ШИМ могут быть сформированы за счет функции сравнения таймера (для всех моделей МК 68HC12) или с помощью широтно-импульсных модуляторов (только для B32 и для всех МК семейства HCS12). Используем модуль ШИМ микроконтролеров B32, который обсуждался в главе 4. Можно вспомнить также, что мы уже использовали ШИМ в настоящей главе, чтобы генерировать сигналы управления двигателем робота, проходящего через лабиринт.

3. Микроконтроллер 68HC12 питается от постоянного напряжения в 5 В при очень малом выходном токе источника. Как управлять с его помощью двигателем с высоким рабочим током, питающимся от постоянного напряжения в 12 В? В главе 5, мы обсуждали, как двигатель может управляться от 68HC12 с использованием силовых полупроводниковых ключей. Сигнал управления ШИМ подается на затвор МОП-транзистора. Двигатель подключен между полюсом напряжения питания и стоком МОП-транзистора, как показано на рис.7.15. Для этого конкретного проекта, мы используем мощный МОП-тран-

Рис. 7.16. *Программа управления двигателем в режиме прерываний*

зистор IRF530N HEXFET компании International Rectifier. Он выпускается в корпусе TO-220AB с тремя выводами и рассчитан на напряжения переключения в 100 В и рабочие токи в 17 А. Внимание!: Не забудьте включить обратный диод параллельно двигателю.

4. Как можно контролировать скорость двигателя и одновременно выполнять другие задачи, стоящие перед системой? Если мы будем использовать обычную последовательную обработку данных, наш процессор будет постоянно привязан к операциям контроля скорости двигателя, регулирования его питающего напряжения и отображения текущего значения скорости на ЖК индикаторе. Однако, как мы упомянули в ответе на вопрос 1, мы используем RTI систему, чтобы реализовать задачи, связанные с этим управлением, один раз в 33 мс. Между этими периодическими прерываниями, МК может выполнять другие задачи. Концепция периодических прерываний для управления двигателем иллюстрируется на рис.7.16. Мы исследуем связанные с ней понятия в следующей главе.

5. Какие подсистемы, встроенные в 68HC12, должны быть использованы, чтобы выполнить эту задачу? На этот вопрос мы в сущности уже ответили. Однако для завершенности обсуждения составим список подсистем необходимых для выполнения проекта:

- Модуль ШИМ;
- Модуль меток реальногьвремени;
- Счетчик внешних событий (Аккумулятор импульсов);
- Силовой коммутатор для управления двигателем;
- Оптический кодер;
- Жидкокристаллический (ЖК) индикатор.

7.4.2. Немного теории

В этом разделе мы более подробно исследуем некоторые из концепций, связанных с проектом, рассмотрев требования к двигателю, работу оптического кодера и конфигурирование прерываний в режиме реального времени на языке Си, а также программу на Си для подсчета импульсов с помощью подсистемы аккумулятора импульсов.

Требования к двигателю. Специфический двигатель, который мы используем – это двигатель постоянного тока, выпускаемый компанией Electro-Craft Corporation. Мы уже рассмотрели основные характеристики двигателя. Однако, имеется ряд дополнительных характеристик, которые требуются для этого проекта, таких, например, как зависимость скорости двигателя от его тока . Так как они не приводятся в информационных данных, мы получим их экспериментально. Будем изменять питающее напряжение, поданное на двигатель, и соответствующую этому напряжению скорость. Для этого подключим частотомер на выход оптического кодера, чтобы определить частоту следования импульсов при заданном питающем напряжении двигателя, и затем вычислим скорость вращения. Одновременно будем измерять также токи двигателя. Результаты эксперимента показаны на рис.7.17.

Оптический кодер. Существует широкое разнообразие оптических кодеров для определения скорости вращения вала двигателя. Эти кодеры закрепляются непосредственно на валу, или могут быть связаны с валом какими-либо устройствами. При установке кодера на вращающийся вал, это устройство обеспечивает на выходе прямоугольное напряжение. Кодеры питаются от постоянного напряжения 5 В, и рассчитаны на максимальную скорость вращения в 12 000 об/мин. Мы используем оп-

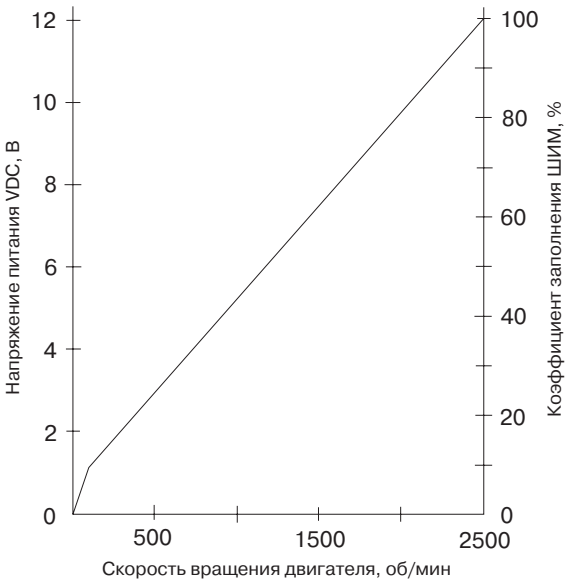
тические кодеры, чтобы обеспечить измерение скорости вращения двигателя, как описано в [11]. Установка для измерения частоты вращения показана на рис. 7.18.

Прерывания в режиме реального времени. Мы используем в МК 68HC12 модуль меток реального времени (RTI), периодически прерывая работу 68HC12, чтобы измерить скорость двигателя и скорректировать коэффициент заполнения ШИМ, если это необходимо для стабилизации скорости вращения двигателя. Перед обсуждением программного кода для управления RTI, мы советуем Вам, возобновить в памяти информацию о сбросах и прерываниях. Ниже приведен краткий обзор действий, необходимых, чтобы инициализировать прерывание RTI:

- Инициализируют вектор прерывания по запросу RTI;
- Устанавливают масштабирующий коэффициент RTR[2:0];
- Устанавливают флаг RTIE разрешения прерываний от RTI в регистре RTICTL;
- Очищают флаг RTIF в регистре RTIFLG;
- Разрешают все маскируемые прерывания прерывания (команда CLI).

Приведенные ниже программный код поможет Вам ознакомиться с работой RTI. В этом примере, мы переходим к программе ISR обработки прерывания RTI, переключая флаг порта PORTP. Если вы исследуете возникающую в результате форму сигнала, показанного на рис.7.19, с помощью осциллографа или логического анализатора, то сможете измерить период повторения системы RTI прерываний.

```
//*****************************************************************
//имя файла: RTI_test.c//
//Port P[0]: Конфигурируется как цифровой выходной порт, обеспечивающий TTL
// совместимый сигналs для управления затвором.
// авторы: Стив Баррет и Даниель Пак
//дата создания: Mar 10, 2003
//последняя редакция: Mar 10, 2004
//*****************************************************************
#include<912b32.h>

#pragma interrupt_handler RTI__isr

//Функции-прототипы
************************************************
void initialize_ports(void);  //инициализация портов
void RTI_isr(void);
void initialize_RTl(void);

//main program* ********************************************************
void main(void)
{
//инициализация вектора сброса для B32
asm(" .area vectors(abs)\n"
     ".org 0xFFF0\n"         // инициализация вектора прерывания RTI
     ".word _RTI_isr\n"
     " .org 0xFFF8\n"        // инициализация вектора сброса для B32
     " .word 0x800 0, 0x8000, 0x800 0, 0x8000\n"
     " .text");

initialize_RTI();            //инициализация модуля RTI
initialize_ports();          //инициализация портов
PORTP = 0x01;        // разрешение PORTP вывод 0
```



Серводвигатель компании Electro-Craft
Рабочее напряжение: 12 В Скорость на холостом ходу: 250 об/мин Пусковой ток на холостом ходу: 2А Рабочий ток на холостом ходу: 370мА Ток под нагрузкой: 600 мА Ток останова: 4А

Рис. 7.17. *Результаты испытаний двигателя*

```
asm("cli");     //разрешение всех маскируемых прерываний
.
.
.
}
//*********************************************************************
// определения функций
// *********************************************************************
//Initialilize_ports: начальная конфигурация
//для портов входа/выхода
// *********************************************************************

void initialize__ports (void)
{
DDRP=0xFF;      //порт PORTP устанавливается как выходной
}
// *********************************************************************
//RTI_isr: подпрограмма обслуживания прерываний по RTI //
**********************************************************************

void RTI_isr(void)
{
RTIFLG = 0x80;              //сбрасывается флаг RTIF
PORTP =~ (PORTP);           //переключить выходы PORTP
}

// *********************************************************************
```

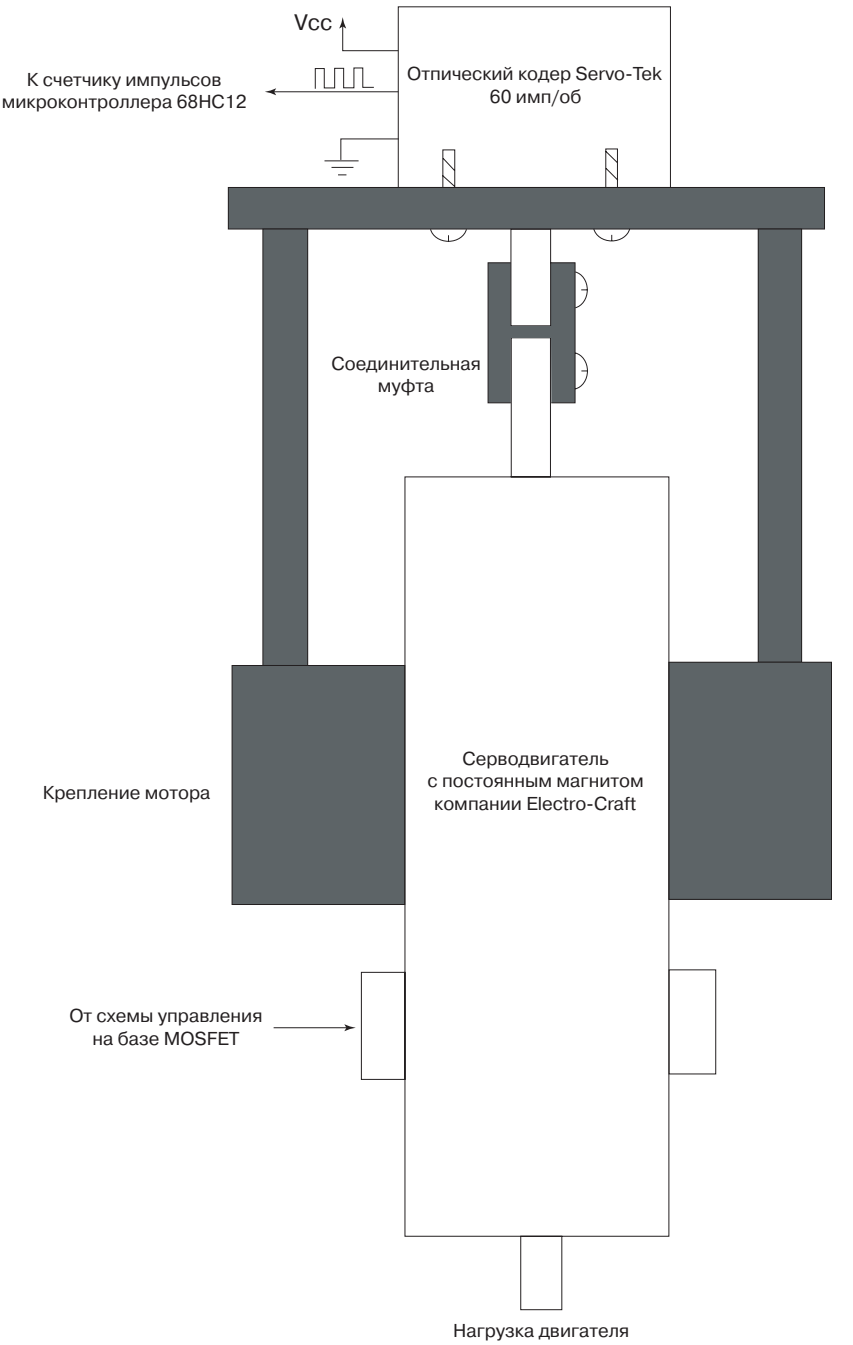


Рис. 7.18. *Установка для измерения скорости вращения двигателя*

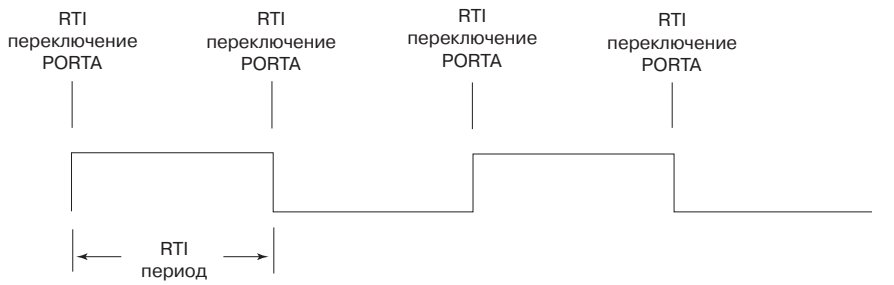


Рис. 7.19. *Прерывания в режиме реального времени*

```
// ********************************************************************
//initialize_RTI:конфигурирует модуль RTI
// ********************************************************************

void initialize_RTI(void)
{
RTICTL - 0x86;       // устанавливается таймер RTI
RTIFLG = 0x80;       //сбрасывается флаг RTIF
}
// ********************************************************************
```

Аккумулятор импульсов. Число импульсов, поступающих от оптического кодера, подсчитывается аккумулятором импульсов (PA). Он инициализируется в начале программы и затем фиксирует текущее число имульсов за интервал каждого прерывания RTI. По известному интервалу времени между прерываниями RTI (32.768 мс) можно определить число импульсов, поступивших на PA между двумя прерываниями, и затем вывести на дисплей скорость вращения двигателя. Для конфигурации PA системы существует специальный код.

```
// ********************************************************************
//initialize_PA: инициализация модуля аккумулятора импульсов МК 68HC12
// ********************************************************************

void initialize_PA(void]
{
TIOS  = 0x00;     //Конфигурировать канал 7 для работы счетчика импульсов
TCTL1 = 0x00;     //кодера - 3 оператора
0C7M  = 0x00;
TSCR  = 0x80;     //Установить бит разрешения таймера
PACTL = 0x70;     //Разрешить работу PA, режим счета событий, событие
                  // по фронту импульса,
}
```

Объединим теперь эту программу с программой для RTI, чтобы подсчитать число импульсов кодера, появившихся за интервал прерывания. Получив, этот результат, преобразуем его в число об/мин и выведем полученное значение на ЖК индикатор.

```
// ********************************************************************
//имя файла: motor.c
//автор: Steve Barrett and Daniel Pack
//дата создания: Mar 10, 2003
//последняя редакция: Mar 25, 2004
// ********************************************************************

// включенные файлы ***************************************************
#include<912b32.h>

#pragma interrupt_handler RTI_isr //объявить подпрограмму прерывания по RTI

// функции-прототипы **************************************************

void initialize_ports(void); //инициализировать порты
void RTI_isr(void);          //подпрограмма обслуживания прерываний RTI
void initialize_RTI(void);   // инициализировать систему RTI
void initialize_PA(void);    //инициализировать аккумулятор импульсов (PA)
void initialize_LCD(void);   // инициализировать ЖК индикатор
void display_count_LCD(unsigned int);
               //вывод текущего содержимого аккумулятора PA
               // на ЖК индикатор
void putchars(unsigned char); //функция поддержки ЖКИ - вывести символ
void putcommands(unsigned char);// функция поддержки ЖКИ вывести команду
void delay_5ms(void);        //задержка 5 мс
void delay_100us(void);      //задержка 100 мкс

unsigned int old_count;      //последнее значение, записанное в (PA)
int RTI_int_count =0;        //используется для подсчета  RTI прерываний

//главня программа***************************************************

void main(void)
{
asm(" .area vectors(abs) \n" //inline assembly statement
     ".org OxFFF0\n"         //инициализация вектора прерываний  RTI
     ".word _RTI_isr\n"
     " .org 0xFFF8\n"        // инициализация вектора reset для 68HC12 B32
     " .word 0x8000, 0x8000, 0x8000, 0x8000\n"
     " .text");

void initialize_ports(void); // инициализация портов
initialize_LCD()             //инициализация ЖКИ
initialize_RTI()             //инициализация модуля RTI
initialize_PA();             //инициализация аккумулятора импульсов
asm("cli");     //разрешение глобального маскирования
               //прерываний
while(1)        //продолжение цикла до следующего прерывания
     {
     ;
     }
}
//********************************************************************
```

```
//*********************************************************************
//initialize_ports: определяет направление передачи портов
//*********************************************************************

void initialize_ports(void)
{
DDRP = 0xFF; // порт PORTP устанавливается как выходной для ШИМ
DDRT = 0x00; // PORTT устанавливается как входной, вывод PORTT[7]
             // в качестве входа аккумулятора импульсов PA
DDRB    = 0xFF;  // PORTB устанавливается как выходной - порт
             // данных для ЖКД
DDRDLC = 0xFF;  // PORT DLC устанавливается как выходной - сигналы
             // управления для ЖКИ
}
//*********************************************************************
//RTI_isr: подпрограмма прервания по RTI
//*********************************************************************
void RTI_isr(void)
{
unsigned int new_count;
unsigned int pulse_count;
float max_count = 65535.0;

new_count = PACNT;   //передается текущее число импульсов, записанное в PA

if (new_count > old_count)   //определяется приращение числа импульсов
     pulse_count = new_count - old_count;
else
     pulse_count = (unsigned int)(max_count-(float)
(old_count = new_count));

RTI_int_count = RTI_int_count + 1;// изменяется показание счетчика
               //RTI-прерываний
if(RTI_int_count == 10)      // изменяется показание ЖКД через
               // каждые 10 прерываний

     {
     display_count_LCD(pulse_count); //изменяется ЖКИ
     RTI_int_count = 0;      //сбрасывается счетчик прерываний RTI
     }
old_count = new_count;
RTIFLG = 0x80;       //сбрасывается RTI
}
// *********************************************************************
// initialize_RTI:конфигурирует регистры, связанные с RTI
// - регистр RTICTL
// -- разрешает работу модуля RTI установкой бита RTIE
// -- период RTI в 32.768 мс
// - сбрасывает бит RTIF в регистре флагов (RTIFLG)
// *********************************************************************

void initialize_RTI(void)
{
RTICTL = 0x86;       // устанавливается период RTI на 32.768 мс
RTIFLG = 0x80;       //сбрасывается флаг  RTIF
```

```
}

// ********************************************************************
//initialize_PA: инициализация модуля аккумулятора импульсов
// ********************************************************************

void initialize_PA(void]
{
TIOS  = 0x00;        // Конфигурирует канал 7 для работы
TCTL1 = 0x00;        // счетчика импульсов оптического кодера
0C7M  = 0x00;
TSCR  = 0x80;   // устанавливает бит разрешения работы таймера
PACTL - 0x70;        // разрешает работу PA, режим счета событий,
               // по фронту импульса,
}

/*******************************************************************/
/* initialize_LCD: инициализации ЖКИ                               */
/* передает на ЖКИ стартовую последовательность команд управления  */
/*  - PORTDLC[3]: линия управления  RS ЖКИ                         */
/*  - PORTDLC[2]: линия управления  E для ЖКИ                      */
/* - PORTB: двунаправленная магистраль данных для ЖКИ              */
/*******************************************************************/

void initialize_LCD(void)
{
delay_5ms();
delay_5ms();
delay_5ms(); /*ожидание в течение 15 мс перед включением ЖКИ */
putcommands(0x38);   /*разрядность данных интерфейса  8 бит       */
delay_5ms(); /*задержка      */
putcommands(0x38);   /*интерфейс в 8 бит                          */
delay_100us();  /*задержка   */
putcommands(0x38);   /* разрядность данных интерфейса  8 бит      */
putcommands(0x38);   /* интерфейс в 8 бит                         */
putcommands(OxOC);   /*включение диплея                           */
putcommands(0x01);   /*очистка дисплея                            */
putcommands(0x06);   /*установка режима инкремента адреса         */
putcommands(0x00);
putcommands(0x00) ;
putcommands(OxCO);   /*курсор на линию 2 знакоместо 1             */
putchars('R');  /* вывести "PRM" - скорость                       */
            /*в об/мин на строку 2 ЖКИ                            */
putchars('P');
putchars('M');
}

/*******************************************************************/
/*putchars: функция посылает ASCII код для вывода на ЖКИ           */
/*******************************************************************/

void putchars(unsigned char c)
{
PORTB = c;   /*вывести на порт PORTB код символа */
```

```
PORTDLC = PORTDLC|0x08;      /*установить RS в 1 для передачи данных */
PORTDLC = PORTDLC|0x04;      /*установить  E в 1                     */
PORTDLC = 0x00;      /* установить E и RS в 0                       */
delay_100us(); delay_100us();
}

/*********************************************************************/
/*putcommands: функция посылает команду управления ЖКИ               */
/*********************************************************************/

void putcommands(unsigned char d)
{
PORTDLC = PORTDLC&0xF7;      /*установить RS в 0 для передачи команды */
PORTDLC = PORTDLC|0x04;      /*установить  E в 1                     */
PORTDLC = 0x00;              /* установить E и RS в 0                */
delay_100us(); delay_100us();
}

/*********************************************************************/
/* delay_5ms: программная задержка 5 мс                               */
/*********************************************************************/

void delay_5ms(void)
{
int i;
for(i=0; i<50; i++)
     {
     delay_100us();
     }
}

/*********************************************************************/
/* delay_100us:программная задержка в 100 мс                          */
/*********************************************************************/

void delay_100us(void)
(
int i;
for(i=0; i<800; i++)
     {
     asm("nop");/*выполнение команды  nopассемблера занимает 1 период*/
     }
}

/*********************************************************************/
/* display_count_LCD: преобразует целое число в ASCII символ         */
/* для вывода на ЖКИ                                                 */
/*********************************************************************/

void display_count_LCD(unsigned int count)
{
unsigned int thousands_int;
unsigned int hundreds_int;
unsigned int tens__int;
unsigned int ones__int;
```

```
char thousands;
char hundreds;
char tens;
char ones;

/*выбирает и преобразует наибольшую значащую цифру в десятичное      */
/* значение  + 48, образуя ASCII код                                  */
thousands_int = count/1000;
thousands = (char)(thousands_int + 48);

/*выбирает и преобразует следующую наибольшую значащую цифру         */
/* в десятичное значение  + 48, образуя ASCII код                     */
hundreds_int = (count - thousands_int*1000)/100;
hundreds = (char) (hundreds__int + 48);

/*выбирает и преобразует следующую наибольшую значащую цифру         */
/* в десятичное значение  + 48, образуя ASCII код                     */
tens_int = (count - thousands_int*10 0 0 - hundreds_int*100)/10;
tens = (char) (hundreds__int + 48);

/*выбирает и преобразует следующую наибольшую значащую цифру         */
/* в десятичное значение  + 48, образуя ASCII код                     */
ones_int = (count-thousands_int*1000-hundreds_int*100-tens_int*10);
ones = (char)(ones_int + 48);

/*выводит результат на ЖКИ*/
putcommands(0x80);   /*курсор ЖКИ преводится на строку 1, позицию 1*/
putchars(thousands);
putchars(hundreds);
putchars(tens);
putchars(ones);
}

/*********************************************************************/
/*********************************************************************/
```

7.4.3. Анализ

В предыдущем разделе мы рассказали, как измерить скорость вращения двигателя и отобразить ее в оборотах в минуту. В этом разделе мы замкнем контур обратной связи. Для этого мы сравним мгновенную скорость вращения с опорным значением, чтобы затем скорректировать коэффициент заполнения ШИМ и стабилизировать скорость вращения двигателя на желательном уровне при изменении нагрузки двигателя.

Стабилизируем скорость двигателя на уровне 1600 об/мин. Чтобы сделать это, мы должны определить требуемый коэффициент заполнения ШИМ для 1600 об/мин. Из полученного нами графика (рис. 7.17) мы видим, что для обеспечения скорости вращения приблизительно в 1600 об/мин, неободимо подать на двигатель напряжение в 8 В. При питании двигателя от источника в 12 В сответствующий коэффициент заполнения ШИМ составит 66.7 % (8 В/ 12 В).

Чтобы достичь коэффициента заполнения ШИМ в 66.7%, установим значение периода ШИМ на 256 единиц, а коэффициент заполнения ШИМ на 172 единицы.

Как мы уже указывали при предыдущем обсуждении системы ШИМ, эти значения устанавливаются в регистрах PWPER0 и PWDTY0, соответственно.

Мы используем сигнал управления ШИМ с частотй 976 Гц. Для этого частоту таймера ШИМ, равную 8 МГц, необходимо поделить на 32. При этом мы получим частоту в 250 кГц (период 4 мкс) и используем ее в качестве синхронизирующей для системы ШИМ. Поскольку мы используем период в 256 ипульсов, частота управления ШИМ будет равна 976 Гц (период = 4 мкс/импульс х 256 импульсов).

Ниже приведен исходный текст программы, позволяющей изменять коэффициент заполнения ШИМ, чтобы стабилизировать скорость на уровне 1600 об/мин. Эта функция используется начальной установки скорости вращения двигателя.

```
/*****************************************************************/
/*init_PWM(): инициализация модуля ШИМ контроллера 68HC12          */
/*****************************************************************/

void init_PWM() {
PWTST = 0x00;   /*Установить порт ШИМ в нормальный режим          */
PWCTL - 0x00;   /*установить режим фронтовой ШИМ                  */
PWCLK = 0x28;   /*без объединения каналов, ECLK/128*/
PWPOL = 0x01;   /*активный уровень 1                              */
DDRP = OxFF; /*Порт P конфигурировать как выходной               */
PWEN = 0x01; /*установить выход ШИМ                              */
PWPERO = 255;   /*установить для ШИМ период, соответствующий 976 Гц */
PWDTYO = 171;   /*установить начальный коэффициент заполнения    */
             /* на отсутствие движения                           */
}
/*****************************************************************/
```

Если в процессе выполнения программы обработки прерывания RTI скорость вращения превышает 1600 об/мин, коэффициент заполнения ШИМ уменьшается на 1. С другой стороны, если скорость вращения меньше, чем 1600 об/мин, коэффициент заполнения ШИМ увеличивается на 1. Каждое приращение или снижения коэффициента заполнения ШИМ приводит к изменению скорости на 8.5 в об/мин. Попробуйтевычислить это изменение в качестве домашней работы (задание 19).

Код, позволяющий сравнить опорную скорость с действительной и скорректировать ее обеспечивается в программе обработки RTI прерывания (RTLISR). В этой распечатке кода, функции поддержки ЖКД были помещены в файл для включения LCD.H с их функциями-прототипами.

7.4.4. Структура программы и блок-схема алгоритма

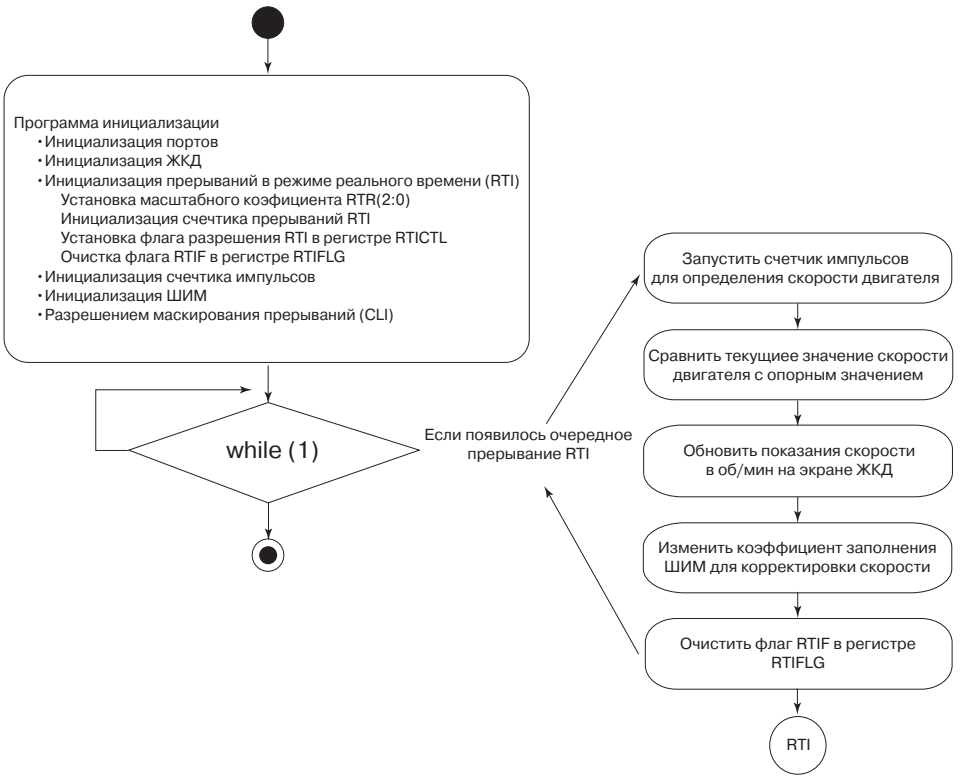
На рис. 7.20 показаны блок-схема алгоритма и структура программы регулирования частоты вращения маломощного двигателя постоянного тока. В следующем параграфе приведен полный текст программы для этого проекта. Попробуйте разработать блок-схемы алгоритмов для каждой функции проекта в качестве домашней работы (задание 20).

7.4.5. Программный код

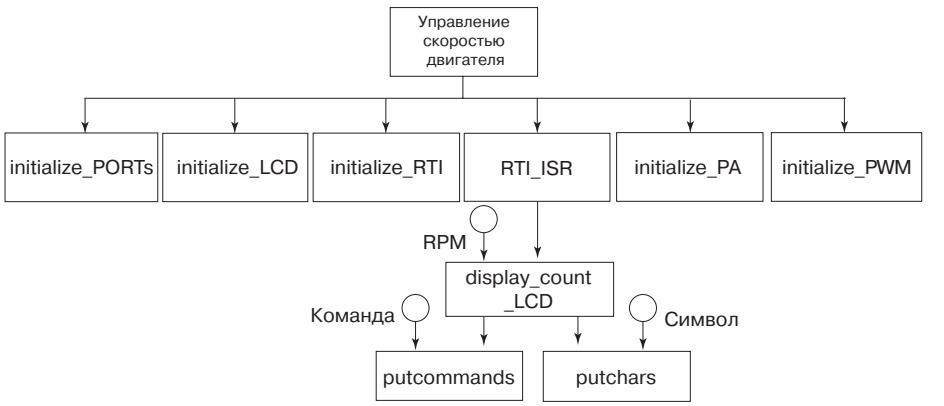
В этом разделе мы использовали методику проектирования сверху-вниз, чтобы разработать, выполнить, и документировать систему стабилизации частоты вращения двигателя. Приводится полный код программы. Если бы мы привели этот код полностью в начале раздела, его сложность могла бы вас испугать. Использование же восходящей методики проектирования позволяет проще и удобнее разбираться с работой каждой функции кода и связью их между собой.

```
//*********************************************************************
//имя файла: motor.c
//авторы: Стив Баррет и Даниэль Пак
//дата создания: 10 марта  2003
//последняя редакция: 2 марта 2004
//*********************************************************************

//включенные файлы
#include<912b32.h>
#include<LCD.h>   //Функции поддерждки   ЖКИ
#pragma interrupt_handler RTI_isr//Объявляет подпрограмму обработки
                            // прерывания по запросу (RTI)

//используемые подпрограммы
void initialize_ports(void); // Инициализия портов
void RTI_isr(void);          //программа обработки прерывания по RTI
void initialize_RTI(void);   // Инициализия модуля RTI
void initialize_PA(void) ;   // Инициализия модуля счетчика событий PA
void initialize_PWM(void) ;  // Инициализия модуля ШИМ

//глобальные переменные
unsigned int old_count;      // последнее содержимое PA
int RTI_int_count =0;        // Используется для подсчета числа
               // прерывания RTI
unsigned char PWM_duty_cycle=172;// Начальный коэффициент заполнения ШИМ
unsigned int desired_motor_RPM = 1600; //эталон скорости двигателя

//*********************************************************************

void main(void)
{
asm(" .area vectors(abs)\n"  // команда встроенного ассемблерного кода
     ".org 0xFFF0\n"         // Инициализировать вектор прерывания RTI
     ".word _RTI_isr\n"
     ".org 0xFFF8\n"
     ".word 0x8000, 0x8000, 0x8000, 0x8000\n"
     ".text");

initialize_ports();          // Инициализация портов
initialize_LCD();            // Инициализация ЖКИ
initialize_RTI();            // Инициализация модуля RTI
initialize_PA();             // Инициализация модуля PA
initialize_PWM();            //Инициализация модуля ШИМ
asm("cli");                  // Разрешить все маскируемые прерывания
while(1)                     // Непрерывный цикл в ожидании прерывания
```

```
      {
      ;
      }
}

//*********************************************************************
//initialize_ports: обеспечивает начальную конфигурацию портов
//ввода - вывода
//*********************************************************************

void initialize__ports (void)
{
DDRP    = 0xFF;       // Установить PORTP как выходной для ШИМ

DDRT    = 0x00;       //Установить P0RTT как входной, а вывод P0RTT[7]
                      // как вход аккумулятора PA
DDRB    = 0xFF;       //PORTB на вывод - двунаправленная

                      //магистраль данных для ЖКИ
DDRDLC = 0xFF;        //порт PORT DLC на вывод - линии управления ЖКИ
}

//*********************************************************************
//RTI_isr: Программа обработки прерывания по меткам реального времени
//*********************************************************************

void RTI_isr (void)
{
unsigned int new_count;
unsigned int pulse_count;
unsigned int current_RPM;
float max_count = 65535.0;

new_count = PACNT;   // Получить текущее число импульсов в аккумуляторе
if (new_count > old_count)   // Определить текущее число импульсов
  pulse_count = new_count - old_count;
else
  pulse_count = (unsigned int)(max_count-(float)

                (old_count +new__count));

                      // Преобразовать число импульсов кодера в об/мин
current_RPM = (unsigned int)(pulse_count/0.032768);

// Изменить число прерываний RTI в счетчике
RTI_int_count = RTI_int_count + 1;

if(RTI_int_count == 10) // Изменить на ЖКИ значение после 10 прерываний  RTI
{
display_count_LCD(current_RPM); // Изменить показания ЖКИ
RTI_int_count=0;
                      // Сбросить счетчик прерываний RTI
}
                      // Изменить значение скорости двигателя
```



а). Управление скоростью двигателя. Блок-схема алгоритма



б). Управление скоростью двигателя. Структура программы

Рис. 7.20. *К программе управления скоростью двигателя*

```
if(current_RPM < desired_motor_RPM)
PWM_duty_cycle = PWM_duty_cycle + 1; // Ускорить двигатель
else
PWM_duty_cycle = PWM_duty_cycle - 1; // Замедлить двигатель

// Изменить скорость двигателя с помощью ШИМ
PWDTY0= PWM_duty_cycle;      //коэффициент заполнения
old_count = new_count;
RTIFLG = 0x80;               //сбросить флаг прерывания RTI
}

//*********************************************************************
// Initialize_RTI: конфигурирует регистры связанные с RTI
//- Регистр управления RTI (RTICTL):
// - разрешение RTI, флаг RTIE
// - установка периода RTI на 32.768 мс
// - сброс RTI, бит RTIF регистре Флагов RTI (RTIFLG)
//*********************************************************************

void initialize_RTl(void)
{
RTICTL = 0x86; // Установить период RTI 32.768 мс
RTIFLG = 0x80; // сброс RTI
}
//*********************************************************************
// Initialize_RTl_PA: инициализация 68HC12 аккумулятора импульсов PA
//*********************************************************************

void initialize_PA(void)
{
TIOS  = 0x00; // конфигурировать канал 7 для работы в качестве счетчика
TCTL1 = 0x00; // импульсов оптического кодера
OC7M  = 0x00; // Включить бит разрешения таймера
TSCR  = 0x80; // Установить разрешение для аккумулятора по фронту
           //импульса,
PACTL = 0x50;   // Режим счета событий
}
/*********************************************************************/
//initialize_PWM: генерировать сигнал ШИМ 976 Гц  с                  */
// коэффициентом заполнения 67.2%                                    */
/*********************************************************************/

void initialize_PWM(void)
{
PWTST =0x00;    /*установить ШИМ в режим нормальной работы          */
PWCTL =0x00;    /*установить  выравнивание  по левой границ         */
PWCLK = 0x28;   /*без конкатенации, разделить clk на 32              */
PWPOL = 0x01;   /*состояние : высокое затем переход к низкому       */
DDRP  = OxFF;   /*установить порт PORTP как выходной                 */
PWEN  = 0x01;   /*разрешение на  ШИМ для канала 0                    */
PWPER0= OxFF;   /* установить период равным 256                      */
PWDTY0= PWM duty_cycle; /* установить коэффициент заполнения равным 172 */
}
/*********************************************************************/
```

7.4.6. Испытания

Установка для испытания системы стабилизации скорости вращения была показана на рис.7.18. Это – очень сложная система, содержащая целый ряд компонентов и микроконтроллерную систему 68HC12. Чтобы гарантировать успешную работу системы, настоятельно рекомендуется применять обсужденный в главе 2 метод измерений. Собрав и запустив всю систему, можно включить двигатель на умеренную нагрузку, и убедиться, что система стабилизирует скорость вращения на уровне в 1600 об/мин.

7.5. Парящий робот

7.5.1. Описание проекта

Летающий робот, разработку которого мы собираемся обсудить – это парящий робот, который может зависать непосредственно над областью, представляющей интерес, пока не выполнит всю необходимую работу. За последние двадцать лет рядом исследователей в университетах и научно-исследовательских лабораториях были изучены и разработаны летающие роботы различных размеров и форм для ряда применений. Одни использовали при этом существующие образцы самолетов и вертолетов, другие изобретали механические системы с машущим крылом, третьи использовали дирижабли. Все эти усилия были предприняты из-за огромных выгод, которые сулят такие роботы в военных и промышленных применениях, включая исследование отдаленных планет, внутреннее и наружное наблюдение, распознавание и локализация целей при военных действиях.

Опишем сначала общие характеристики системы, составляя список требований к проекту, а затем представим проект парящего робота, удовлетворяющий всем этим требованиям. Описание системы включает в себя описание конструкции устройства, структуру программы, и блок-схему алгоритма. После этого приводится пример программы на языке Си.

Цель этого проекта состояла в том, чтобы разработать и изготовить автономную систему парящего робота, управляемую микроконтроллером 68HC9S12. Наиболее трудным аспектом системы является проблема стабильного положения робота, которое должно постоянно поддерживаться контроллером. На рис. 7.21 показана фотография робота на базе парящей рамы. Этот проект был создан студентом старших курсов.

Чтобы обеспечить работоспособность проекта, мы должны сначала определить все требования к системе. При создании проекта необходимо решить ряд сложных проблем, в числе которых: создание прочной, но легкой летающей рамы, которая удовлетворяла бы летным требованиям, выбор и размещение датчиков, выбор привода пропеллеров, проектирование системы питания и решение задачи стабилизации положения.

Парящий робот должен удовлетворять следующим общим требованиям:

1. Быть автономным;
2. Иметь рабочую область размером 16 x 16 футов. (40,6 x 40,6 см);

3. Весить не больше, чем 2,5 фунта (1,14 кг);
4. Обладать собственными средствами взлета и посадки;
5. Использовать двигатели постоянного тока;
6. Поднимать до 4,5 фунтов (2,05 кг), включая собственный вес;
7. Поддерживать постоянную высоту парения;
8. Осуществлять движение во всех направлениях;
9. Избегать препятствий;
10. Иметь энергонезависимую память данных;
11. Стоить не более $500,00.

7.5.2. Системы HCS12 используемые в проекте

В этом проекте будут использоваться следующие под системы HCS12:

- Подсистема входного захвата таймера;
- Модуль АЦП;
- Подсистема ШИМ.

7.5.3. Теоретическое обсуждение

Для парящего робота была разработана простая воздушная рама, подобная существовавшему ранее покупному радиоуправляемому устройству, показанному на рис.7.21. Она собрана из четырех стержней, скрепленных в центре рамы, имеющей форму креста. К каждому концу стержня прикреплен двигатель постоянного тока и механизм привода для пропеллера. Два соседних пропеллера вращаются во встречных направлениях, чтобы предотвратить рыскание рамы, при этом два пропеллера вращаются, по часовой стрелке, а два других - против часовой стрелки. В обычных вертолетах встречное вращение обеспечивается хвостовым пропеллером.

Для управления использовалась Т-плата компании ImageCraft с МК семейства HCS12 (рис. 7.22). Частота тактирования МК равна 25 МГц, резиденная Flash-память программ МК равна 256 Кб, оперативную память – 12 Кб и энергонезависимая память данных типа EEPROM – 4 Кб. Для бортовой схемы весьма желательна миниатюризация платы. Для управления двигателями постоянного тока используется встроенный модуль ШИМ. Пьезогирометрические датчики для трех осей вращения обеспечивают изменение углов тангажа, крена и рыскания для управления парящим роботом. Выходы гиродатчика поданы на вход таймера МК и используются, чтобы корректировать скорости вращения всех четырех двигателей парящего робота. В дополнение к гиродатчикам, на роботе установлены четыре инфракрасных датчика. Они обнаруживают преграды, когда робот приближается к стенкам или препятствиям. Выходы датчиков поданы на входной порт АЦП, их сигналы обеспечивают выбор алгоритма управления полетом, позволяющего избежать столкновений со стенками или препятствиями.

В качестве гиродатчиков используются три пьезогиродатчика GYA350 компании Futaba. Мы выбрали эти датчики, поскольку они специально предназначенны для авиамоделей. Датчик весит 26 г и размещается в корпусе 27 мм x 27 мм x 20 мм. Он обеспечивают сигналы ШИМ частотой 55 Гц, при этом изменение ширины импульса указывает направление движения датчика и, следовательно, направление движения парящего робота. Этот гиродатчик может также работать в режиме поддержки

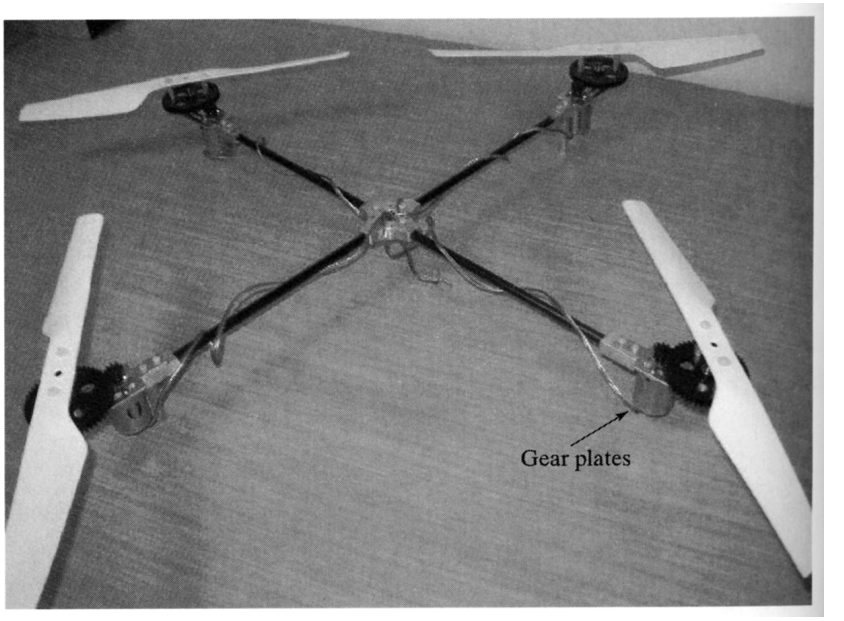


Рис. 7.21. *Робот на базе парящей рамы*

заданного направления (режим heading-hold), при котором микроконтроллер может определить ширину импульса на выходе датчика летящего робота. При использовании функции входного захвата таймера, робот проверяет направление, оценивая выходные ШИМ сигналы гиродатчиков.

В дополнение к гиродатчикам робот использует для измерения расстояния четыре инфракрасных пары передатчик/приемник GP2D12 фирмы Sharp, чтобы избежать столкновения с любыми объектами. Эти датчики могут обеспечивать диапазон выходных напряжений, соответствующих расстоянию обнаружения от 10 до 80 см. Эти напряжения преобразуются в соответствующие цифровые значения с помощью модуля ATD микроконтроллера. Датчик легок и размещается в корпусе 45 мм x 14 мм x 20 мм.

Для взлета использовались четыре двигателя постоянного тока Graupner Speed300, питающиеся от постоянного напряжения 6 В. Каждый двигатель весит 50 г при диаметре вала в 2 мм и потребляемом токе до 5 А. На валу двигателя установлена маленькая пластмассовая шестерня, которая приводит во вращение большую шестерню на оси пропеллера, как показано на рис.7.21.

7.5.4. Структура программы и блок-схема алгоритма

Схема подключения на рис.7.23 показывает связи компонентов парящего робота с контроллером. На рис. 7.24 схематично показана конструкция робота со всеми компонентами. Обратите внимание, что ШИМ-сигналы регулирующие скорость вращения подаются на ИС силового коммутатора, которая обеспечивают достаточный ток для двигателей постоянного тока. Чтобы обеспечить питание системы используется несколько батарей. Сигналы поступают в МК от датчиков, а выходные сигналы МК управляют ИС коммутаторов двигателей.

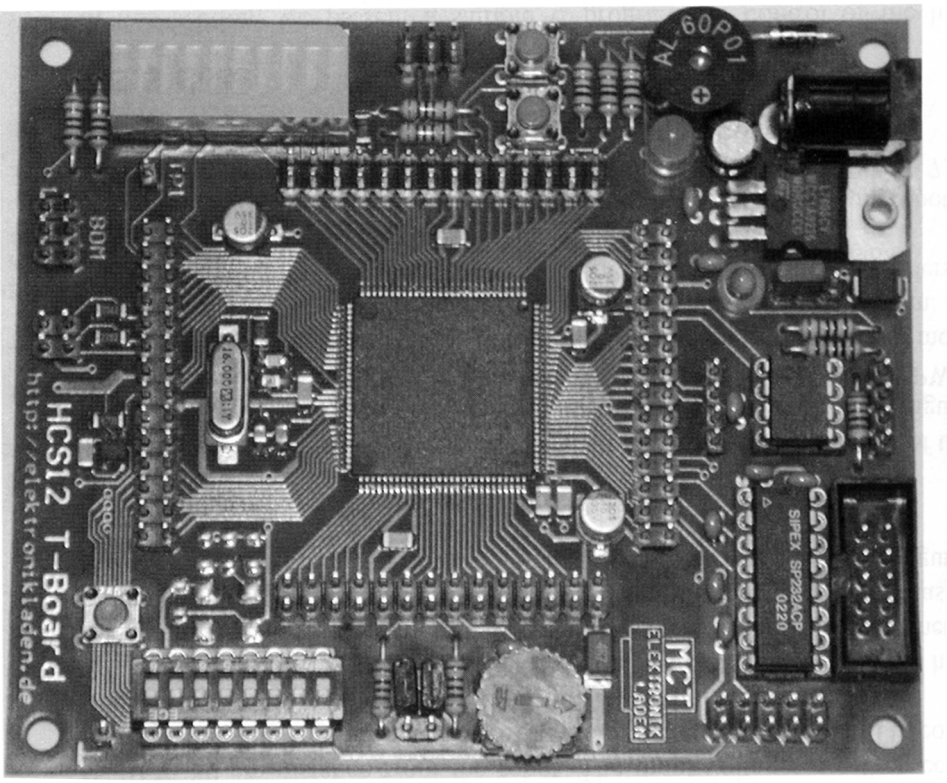


Рис. 7.22. *Плата с МК HCS12*

На рис. 7.25 представлена структура программы для парящего робота, в которой показаны направления потоков данных между программными модулями. На рис. 7.26 показан блок-схема алгоритма программы для управления роботом.

7.5.5. Программный код

В этом разделе, мы представляем программу для управления парящим роботом на языке Си. Программа была первоначально написана Джоулом Перлином, одним из наших студентов, и скорректирована авторами настоящей книги.

```
//*********************************************************************
// filename: flying.c
// Описание программы: Эта программа запускает четыре двигателя и затем
// управляет скоростью вращения каждого из них.  После взлета робота
// программа проверяет каждый из датчиков для определения положения
// робота. При обнаружении преграды или крыши, скорости вращения двигателей
// корректируются, чтобы  обеспечить правильное направление движения.
//
// Авторы: ДжоэльПерлин, Даниэль Пак, Стив Баррат
// Дата создания: 27 июля 2004
```

```
// используемая память: программа - 0x1000, данные - 0x3000 и
// стек -  0x4000
//*********************************************************************

#include <stdio.h>
#include ''hcs12dp256.h''

#pragma abs_address 0x3000
unsigned int countl; //счетчики переменных
unsigned int count2;
unsigned int sensor;
char sensoravg; //сохраняют данные датчиков в виде 8-разрядных чисел
volatile unsigned p; // текущий счетчик
#pragma end__abs_address

//*********************************************************************

void main(void){

//Инициализация робота
PWME = 0x00;    //запрет ШИМ
DDRA = OxFF;    //конфигурирование портов  A и B как выходных
DDRB = OxFF;
PORTA = OxAA;   //подача питания на датчики
PORTB = OxFF;   //индикация режима программы на линейке светодиодов

//Инициализация модуля ATD
PORTAD1 = 0x00;      //конфигурирование портов как входных
ATD1CTL2 = 0xC2;     //инициализация ATD с установкой флагов
                     //преобразования в каналах
ATD1CTL3 = 0x00;     // функция  установки ATD
ATD1CTL4 = 0x80;     //установить 8-разрядный режим
PORTB = OxFE;        //показать состояние программы на линейке
                     // светодидов

// Инициализация режима входного захвата
TSCR1 = 0x80;   //включение таймера (нормальный режим)
TSCR2 = 0x80;   //установить период переполнения счетчика временной
                // базы 8.192 мс, коэффициент деления = 1
TIOS = 0x00;    //установить каналы таймера в режим входного захвата
TMSK1 = 0xE0;//биты разрешения прерывания по событиям на линиях [7:5]
TFLG1 = 0xE0;   //очистить флаги прерываний TFLG1
PORTB = 0xFC;   // показать состояние программы на линейке светодидов

// инициализация ШИМ
PWMCTL = 0x00;  //установить 8-разрядный режим
PWMCAE = 0x10;  //установить фронтовую ШИМ

PWMPOL = 0x5F;  //выбрать активным высокий логический уровень

//назначить режим ШИМ для каналов 0,1,2,3,4 и 6

PWMCLK = 0x50;  //каналы 0,1,4 тактируются CLOCK_A и каналы
                            //2,3,6 - CLOCK_B
PWMSCLA  =  0x20;    //период A 0x20 = 4.1 мс
```

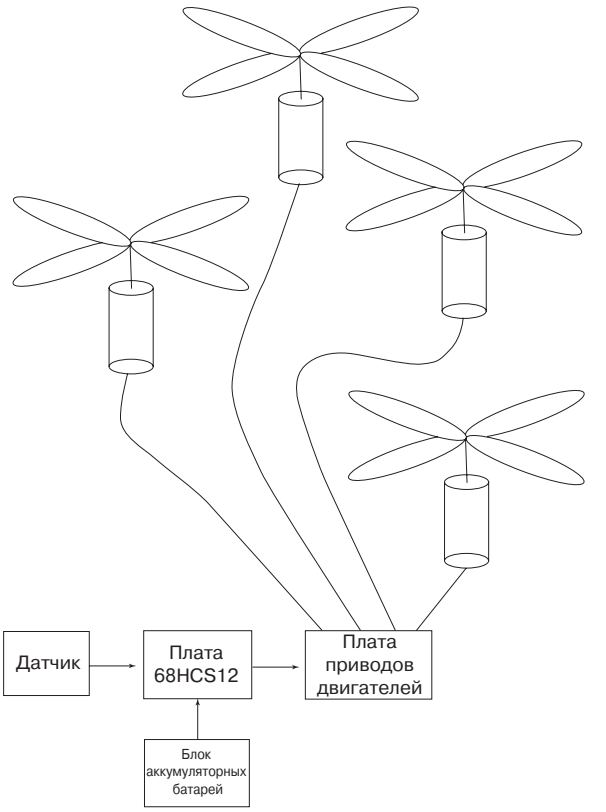


Рис. 7.23. *Схема подключеня управляющих компонентов парящего робота*

```
PWMSCLB = 0x04; //период  В 0x02 = 255 мкс, выбрать
                // наибольший коэффицент заполнения для всех каналов
PWMPER0 = 255;
PWMPER1 = 255;
PWMPER2 = 255;
PWMPER3 = 255;

//Запуск двигателей
PWME= PWME 0x5F;      // разрешения режима ШИМ для каналов 0,1,2,3,4 и 5
PWMDTY0 = 80;         // установить коэффициенты заполнения
                      // для каналов 0,1,2 и 3
PWMDTY1 = 80;
PWMDTY2 = 80;
PWMDTY3 = 80;

// Взет
while (PWMTY0 < 200)
     {
     PWMDTY0 = PWMDTY0 + 1
     PWMDTY1 =PWMDTY1 + 1
```

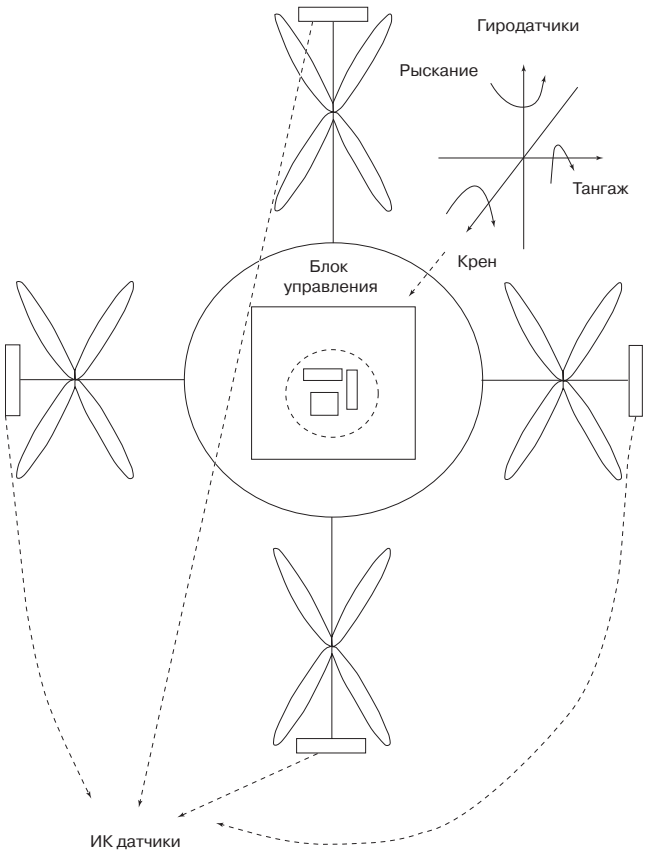


Рис. 7.24. *Диаграмма движения робота со всеми бортовыми компонентами*

```
      PWMDTY2 = PWMDTY2 + 1
      PWMDTY3 =  PWMDTY3 + 1
      }

P = 20;

// установить коэффициенты заполнения для двигателей на режим полета
while (battery = = 1)        //проверить включение батареи
          {
// двигатель 1
ATD0CLT5 = 0x04;             //режим оцифровки многоканальный
while ((ATD0STAT0 & 0x80) = = 0);
sensoravg = ATD0DR4H;        //установить PAD00 для датчика 1
if((int)sensoravg > 80)
     {
     PWMDTY0 = PWMDTY0 + 20;
     delay2();
     for (i=0; i<p; i++)
```

Рис. 7.25. *Структура программы парящего робота*

Рис. 7.26. *Блок-схема алгоритма программы парящего робота*

```
      PWMDTY0--;
      }

// двигатель 2
ATD0CLT5 = 0x04;
while ((ATD0STAT0 & 0x80) = = 0);
sensoravg = ATD0DR5H;        //установить PAD00 для датчика 2
if((int)sensoravg > 80)
      {
      PWMDTY1 = PWMDTY1 + 20;
      delay2();
      for (i=0; i<p; i++)
            PWMDTYl--;
      }

// двигатель 3
ATD0CLT5 = 0x04;
while ((ATD0STAT0 & 0x80) = = 0);
sensoravg = ATD0DR6H; //установить PAD00 для датчика 3
if((int)sensoravg > 80)
      {
      PWMDTY2 = PWMDTY2 +  0;
      delay2();

      for (i=0; i<p; i++)
            PWMDTY2--;
      }

// двигатель 4
ATD0CLT5 = 0x04;
while {(ATD0STAT0 & 0x80} = = 0);
sensoravg = ATD0DR7H; //установить PAD00 для датчика 4
if{(int)sensoravg > 80)
      {
      PWMDTY7 = PWMDTY7 + 20;
      delay2();
      for (i=0; i<p; i++)
      PWMDTY7--;
      }
}//конец while

// снизить скорость двигателей для посадки
while (PWMDTY0 > 80)
      {
      PWMDTY0--;
      PWMDTY1--;
      PWMDTY2--;
      PWMDTY3--;
      delayl();
      }

// остановить двигатели
      PWME = 0x00;
}
//*********************************************************************
```

7.5.6. Некоторые комментарии

Вышеупомянутая программа – неполная версия программы парящего робота. Читатели обратят внимание, что она содержит только простой алгоритм управления. Кроме того, из-за проблем дрейфа, полученных в экспериментах с гиродатчиками, для предотвращения столкновений с препятствиями в программе используются только инфракрасные датчики.

7.6. Система защиты компьютера, основанная на нечеткой логике

7.6.1. Описание проекта

В этом разделе описана основанная на нечеткой логике система защиты от вторжения, которая может защитить ваш компьютер от несанкционированного внешнего доступа. В частности, рассматриваемая система может с помощью встроенных в МК 68HC12/HCS12 функций поддержки нечеткой логики обнаруживать внедрение постороннего гипертекста (HTTP) при туннелировании информации по сетевой шине компьютера. Туннелирование применяется для легальной установки сеансов связи между ведущим компьютером внутри компьютерной сети и компьютером, внешним по отношению к этой сети. «Злоумышленник» использует такой внешний компьютер, чтобы установить тайный сеанс связи, формируя несанкционированные сообщения внутри пакетов обычного протокола HTTP. Чтобы предотвратить такие действия, общество защиты компьютерной информации разработало в прошедшем десятилетии ряд информационных систем. Цель описываемой микроконтроллерной системы защиты состоит не в том, чтобы заменить существующие коммерческие системы, а в том, чтобы помогать таким системам, оценивая присутствие в данных сетевого трафика внедренных посторонних HTTP сообщений.

Цель данного проекта состоит в создании переносной микроконтроллерной системы для анализа туннелирования HTTP. Эта система должна обнаруживать следующие злонамеренные и несанкционированные действия при туннелирования HTTP:

- атаки на интерактивные сеансы туннелирования;
- атаки с помощью внедрения скриптов в сеансах туннелирования;
- внедрение несанкционированных видео и звуковых сигналов в передаваемых потоках информации.

Система защиты должна удовлетворять следующим системным требованиям:

1. Использовать МК семейства;
2. Иметь светодиодное табло для отображения состояния трафика;
3. Обнаруживать интерактивные сеансы туннелирования;
4. Проводить поиск внедренных скриптов при туннелировании;
5. Обнаруживать несанкционированное видео и звуковые сеансы передачи;
6. Использовать алгоритмы нечеткой логики для определения состояния сетевого трафика.

7.6.2. Использование системы HCS12

Этот проект широко использует команды для поддержки алгоритмов нечеткой логикой, которые имеются в списке команд процессорного ядра HCS12.

7.6.3. Основы теории

На рис. 7.27 показан полный процесс, который должна реализовать микроконтроллерная система защиты, чтобы идентифицировать нежелательный сетевой трафик. Система обрабатывает набор из шести входных сигналов, создаваемых ПК, используя алгоритмы нечеткой логики. (Обзор команд 68HC12 для реализации законов управления с нечеткой логикой см. в [11]). Входными сигналами для системы являются оценки уровня членства; нулевая оценка означает отсутствие членства, в то время как максимальная оценка представляет совершенное членство. Первые три входных сигнала отражают оценки членства для профилей поведения. Профиль поведения составлен из таких атрибутов сеанса, как размер пакета, число пакетов сеанса, продолжительность сеанса, соотношение между большими и малыми пакетами, направления передачи данных, средний размер пакета, изменение стандартного размера пакета и общих размеров полученных пакетов.

При сравнении каждого файла на атакующее поведение (у нас три таких файла, описывающих три типа несанкционированной активности при туннелировании HTTP: атаки на интерактивные сеансы туннелирования, атаки с помощью скриптов и внедрение видео и аудио потоков) с профилем данных обычного трафика шины ПК система обнаружения вторжения формирует оценку членства во множестве действий туннелирования HTTP (tunneling activity detection – TAD). Другие три входа системы формируют оценки членства в множестве ключевых слов, характерных для каждого из видов атаки.

Пакеты данных в сеансе туннелирования HTTP, содержат одно или несколько ключевых слов, по которым можно идентифицировать присутствие туннелирования. Испытуемый ПК сравнивает 40 ключевых слов со словами, которые появляются в данных сеанса. Оценка соответствия передается переносной системе в качестве второй входной величины.

На рис. 7.27 показана вся система с необходимыми входными сигналами. На рис. 7.28 приведено размещение переносной системы для обнаружения вторжения в сетевую среду. В беспроводной сети переносная система может быть связана с персональным компьютером. Библиотечные подпрограммы Libpcap используются, чтобы выбрать пакеты сеанса из необработанных данных Internet, программа Psplice проводит синтаксический анализ отдельных пакетов, а модуль предварительной обработки сравнивает данные пакета по трем профилям поведения и по оценке членства во множестве ключевых слов. Более подробно о библиотечных подпрограммах, программе Psplice и модуле предварительной обработки, см. в [11].

7.6.4. Структура программы и блок-схема алгоритма

На рис. 7.29 приведена структура аппаратных средств системы HTTP. Структура программы для системы защиты относительно проста. Она состоит из трех машин с нечеткой логикой, каждая из которых работает как программный модуль, показанный на на рис.7.27. Чтобы увеличить быстродействие, можно использовать три платы HCS12 для обнаружения на шине трех действий вторжения. На рис. 7.30 показана блок-схема алгоритма для системы защиты.

7.6.5. Описание системы

На рис. 7.31а, показан внешний вид переносной системы при работе. ЖК дисплей показывает следующее начальное сообщение:

```
Portable HTTP

TAD System

Version 1.0
```

На рис. 7.31б, показан следующий типовой экран после того, как сеанс оценен:

```
IA: 60 Med Alert

SA: 40 Low Alert

S : 40 Low Alert
```

Сообщение показывает ряд оценок текущего сеанса для трех типов атак на туннелирование HTTP и соответствующих им сообщения.

Процессы для каждой из трех подсистем с нечеткой логикой в переносной микроконтроллерной системе TAD идентичны. Каждая подсистема состоит из модуля с нечеткой логикой и модуля дефаззификации. Используются две оценки членства: для профиля поведения и для ключевых слов, позволяющие определить состояние защищаемой сессии. Состояние соответствует одной из трех категорий, и соответствующие сообщения отображаются на экране графического ЖК дисплея.

В модуле предварительной обработки, стандартные профили поведения сравниваются с полученными в сеансе, чтобы вычислить оценку членства для интерактивной подсистемы TAD, подсистемы скриптового вторжения и подсистемы внедрения потоков. Точно так же три набора ключевых слов соответствуют сеансу вторжения в интерактивное туннелирование HTTP, сеансу скриптового вторжения и сеансу внедрения потоков. После расшифровки стенограммы, формируются три оценки членства во множестве ключевых слов,. Шесть числовых множеств затем посылаются на микроконтроллерную систему.

Как только переносная система безопасности TAD получает набор шести оценок, он пересылается на три подсистемы с нечеткой логикой – подсистемы фаззификации. Каждая из подсистем TAD вычисляет оценку для каждого из видов атаки на туннелирование. Опишем теперь системы фаззификации.

Три подсистемы фаззификации HTTP TAD обнаруживают каждый из трех рассматриваемых типов атаки. Так как все три подсистемы имеют одинаковую структуру, мы опишем только одну из них, указав отличия для других двух подсистем. Как только две оценки сеанса, созданные модулем предварительной обработки посылаются одной из подсистем, подсистема сначала их фаззифицирует: сравнивает значения членства по правилам нечеткой логики с набором соответствующих лингвистических переменных, называемых функциями членства. Машина с нечеткой логикой преобразует входные лингвистические переменные к выходные. После этого пробразования, подсистема выполняет процесс дефаззификации, чтобы вычислить численное значение, которое указывает состояние защиты исследуемого сеанса.

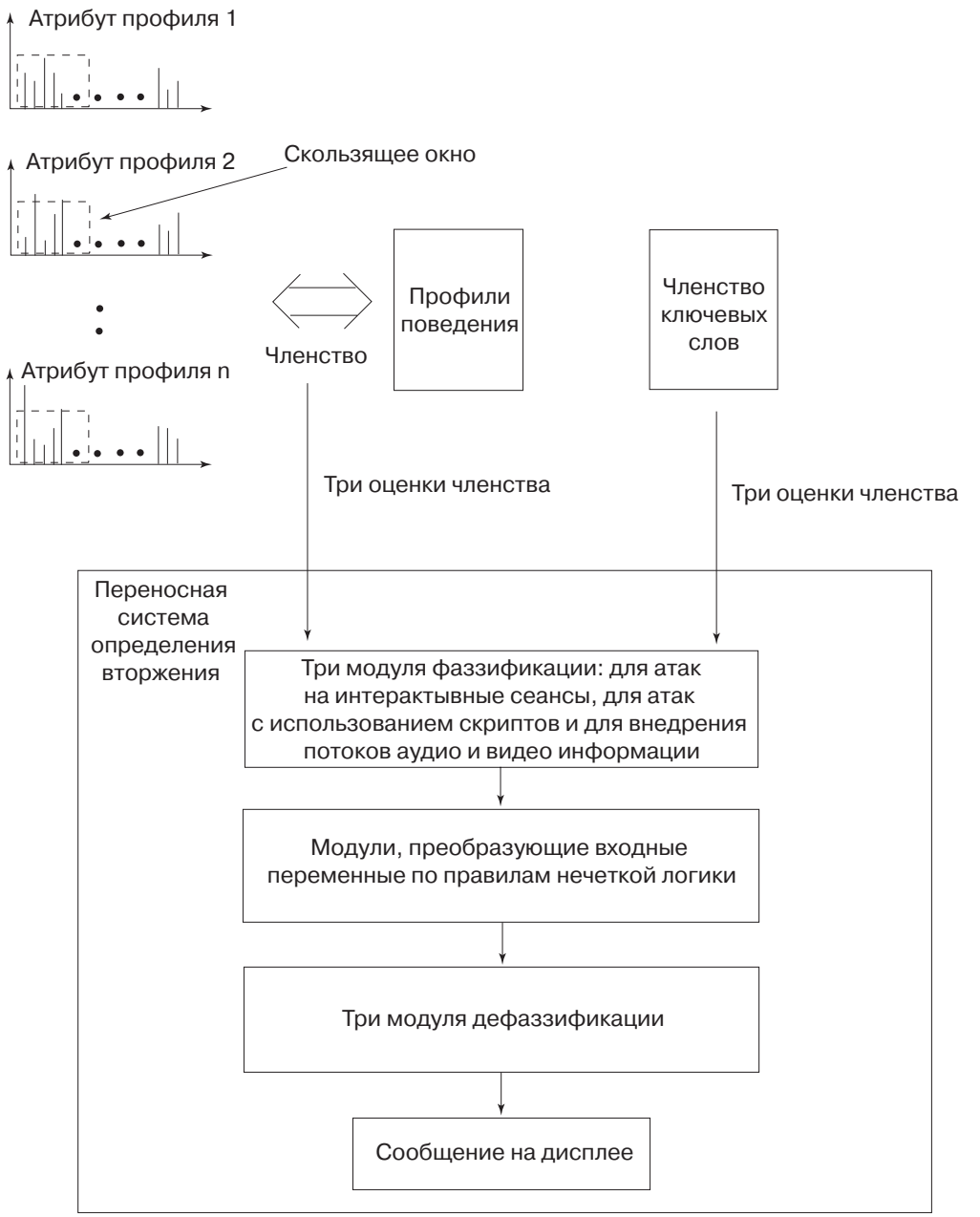


Рис. 7.27. *Определение оценок членства для трех наборов и двух множеств (шесть входов) Оценивается соответствие конфигурации поведения при различных типах атки и соответствие набору ключевых слов, характерных для таких атак. Исследуются три типа атак:на интерактивное туннелирование, с помощью внедренных скриптов и с помощью несанкционированных видео и звуковых потоков. Каждому из типов атаки сопоставляется две оценки. Это множество из шести преобразуется в результрующую оценку сеанса трафика*

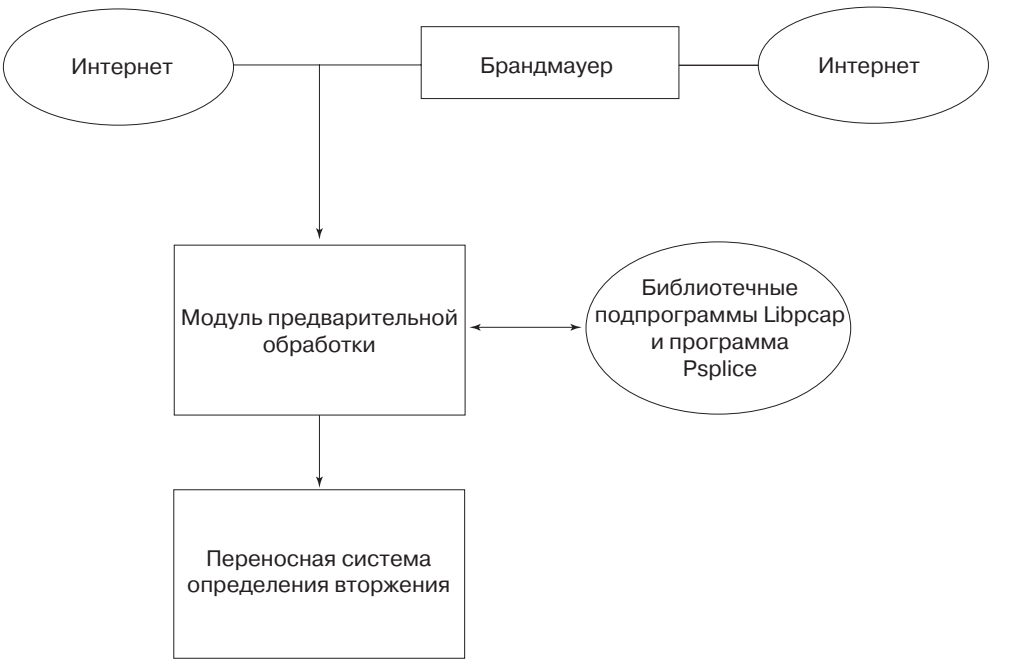


Рис. 7.28. *Структура связей переносной системы обнаружения вторжения с общей сетью*

Начнем с входных лингвистических переменных. Все подсистема имеют одинаковую градацию входных переменных. Имеются три значения переменной, описывающей уровень пар конфигурации поведения: низкий уровень членства, средний уровень и высокий уровень. Входные функции членства показаны на рис.7.32.

Функции членства получены в форме, удобной для того, чтобы воспользоваться встроенными функциями нечеткой логики МК HCS12. Диапазон значений по оси Y показывает, что максимальное значение функции членства не может превышать 255, предельного значения для 8-разрядного регистра, используемого в МК. Значения по оси X соответствуют оценке членства, полученной от модуля предварительной обработки: оценка 0 указывает на отсутствие членства, оценка 255 выражает абсолютное членство. Выбран диапазон числовых значений, позволяющий совместить оценку членства с 8-разрядным числовым форматом, используемым в системе микроконтроллера.

В дополнение к профилю поведения, подсистема получает также соответствующую оценку членства ключевых слов, выраженную натуральным числом пределах от 0 до 255. Функцию на рис. 7.32 можно применить и для оценки членства ключевых слов. Как показано на рисунке, использовано идентичное описаниие входной функции членства. Снова ось y представляет уровень членства от 0 до 255, и значение оси x определяет оценку членства для полученных ключевых слов.

Из-за перекрытий функций членства, одно числовое значение может соответствовать нескольким входным функциям членства. Процесс идентификации входной функции членства и соответствующего значения членства для всех возможных входных функций членства назван процессом фаззификации (fuzzification).

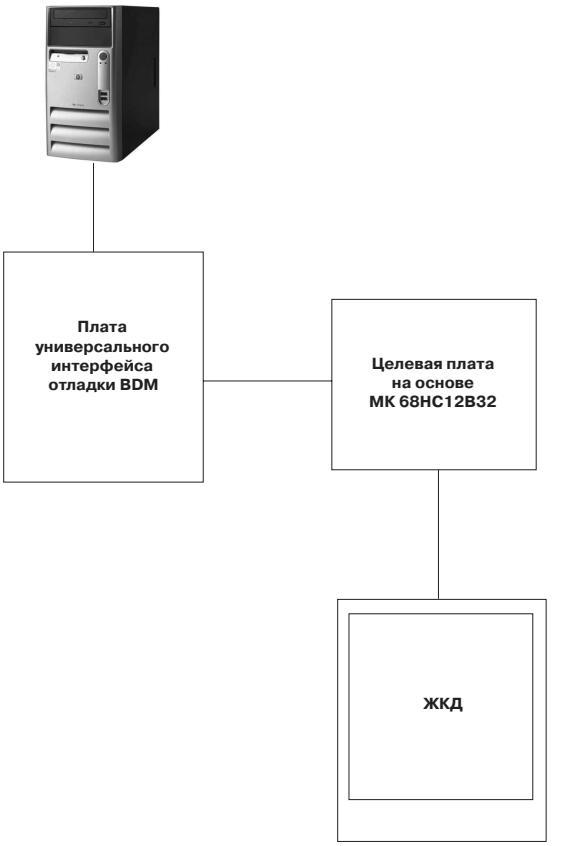


Рис. 7.29. *Структура аппаратных средств системы защиты*

В конце процесса фаззификации, мы имеем одну или несколько входных функций членства с соответствующими значениями. Следующий шаг в процессе должен отобразить нечеткие входные функции в нечеткие выходные функции, используя определенный набор правил. Прежде, чем мы обсудим процесс отображения в логической машине с нечеткой логикой, необходимо определить нечеткие выходные функции членства. Пять из них показаны на рис.7.33: (1) нет опасности, (2) низкая опасность , (3) средняя опсаность, (4) опасность и (5) высокая опасность.

Вывод лингвистических переменных функции членства на рис.7.33 может выглядеть необычно для экспертов по система с нечеткой логикой. Как мы покажем далее, проектировщики МК HCS12 фирмы Motorola упростили процесс дефаззификации, сопоставив каждой выходной функции с нечеткой логикой одно значение. Для текущей подсистемы, мы применяем девять правил, показанных в табл. 7.1, чтобы преобразовать входные переменные в выходные.

Мы использовали общий метод минимакса, показанный на рис.7.34, чтобы получить значения членства для выходных переменных. То есть наибольшее из значений членства входных функций отображается в соответствующей выходной функции. Если после применения всех правил, и выходная функция имеет более одного

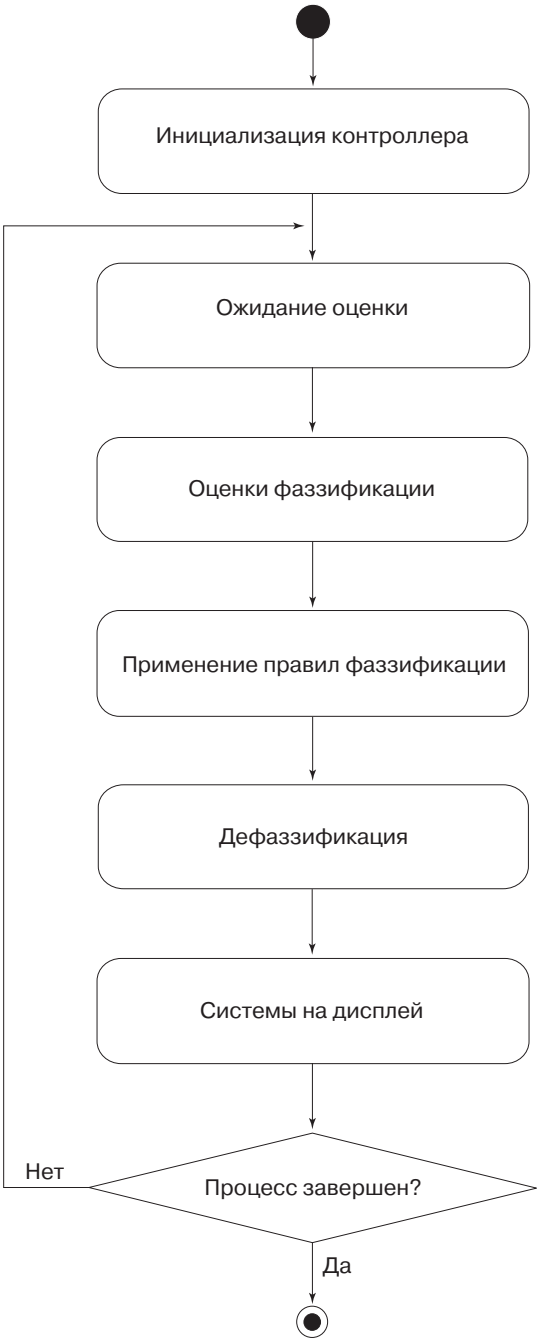
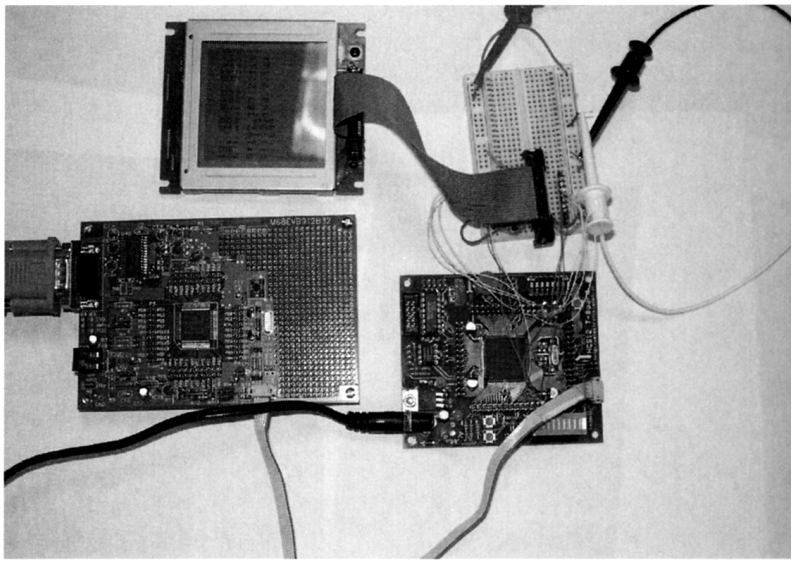
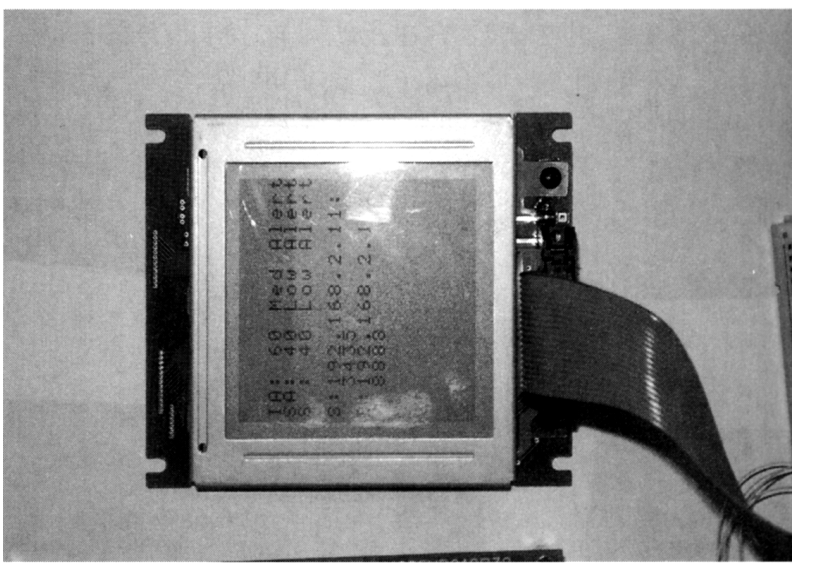


Рис. 7.30. *Блок-схема алгоритма UML для системы защиты*



а) Компоненты, которые составляют систему контроллера. В левой нижней части показана плата 68HC12, используемая для интерфейса ПК с целевой платой, показанной в правом нижнем углу фотографии. Целевая плата использует микроконтроллер 68HC129S12DP256B, входящий в семейство контроллеров HCS12



б) Жидкокристаллический дисплей отображает типовое сообщение

Рис. 7.31. *Микроконтроллерная система оценки атак на HTTP туннелирование (TAD)*

значения, то для выходной функции выбирается наименьшее значение. Для дефаззификации выходного значения, все выходные функции и соответствующие значения членства используются, чтобы вычислить центральное выходное значение. Полученные значения выходных функций функций (показанные на оси X рис.7.33), умножаются на значения членства, вычисленные по девяти правилам, результаты складываются и сумма делится на число членов. Конечное числовое значение и соответствующее сообщение распечатано на экране ЖК дисплее, чтобы сообщить пользователю состояние сеанса. Таким образом, для каждого сеанса, ЖК дисплей высвечивает три числа, характеризующих три типа активности вторжения: атаки на интерактивные сеансы туннелирования, атаки с помощью скриптов и внедрение видео и аудио потоков.

7.6.6. Обсуждение проекта

В последнее время среди различных систем обнаружения вторжения наибольший интерес проявляется к системе обнаружения атак на туннелирование HTTP. Основной причиной такого внимания служит широкое распространение трафика HTTP в Internet. Большинство брандмейстеров не выполняет обширных проверок, чтобы проверить все атаки на туннелирование HTTP. Такие проверки весьма трудоемки, поскольку трафик HTTP составляет наибольшую часть приходящего и уходящего трафика для любой большой организации.

Микроконтроллерная система может взять на себя часть функций системы обнаружения вторжения, связанных с атаками на туннелирование HTTP. Система использует свойства сессии, чтобы проверить данные трафика Internet, которые составляют сессию Internet – программные связи между двумя компьютерами для

Рис. 7.32. *Входные функции членства для вторжения в интерактивное туннелирование подсистемы TAD*

передачи данных. Хорошим примером сессии является набор адреса сети в браузере. Ваш компьютер запрашивает сессию у ведущего компьютера, который содержит страницу сети, которая вам необходима. Как только вы закрываете веб-сайт и переходите к другому сайту, вы закрываете одну сессию и запускаете другую.

7.6.7. Программный код

```
//*****************************************************************
// Файл: micro.c
// Функция программы: Устанавливает шесть IDS оценок членства и создает
// оценку опасности вторжения
// Авторы: Даниэль Пак, Барри Муллинз, Стив Баррет
// Даиа создания: 17 июня 2004
// Установки: Program=0xl000, Data=0x3000, Stack=0x4000
//*****************************************************************

ftinclude <stdio.h>
#include "hcsl2dp256.h"

//*****************************************************************
//функци поддержки
//=================================================================
// _HC12Setup: выключить сторожевой таймер COP
//=================================================================
```

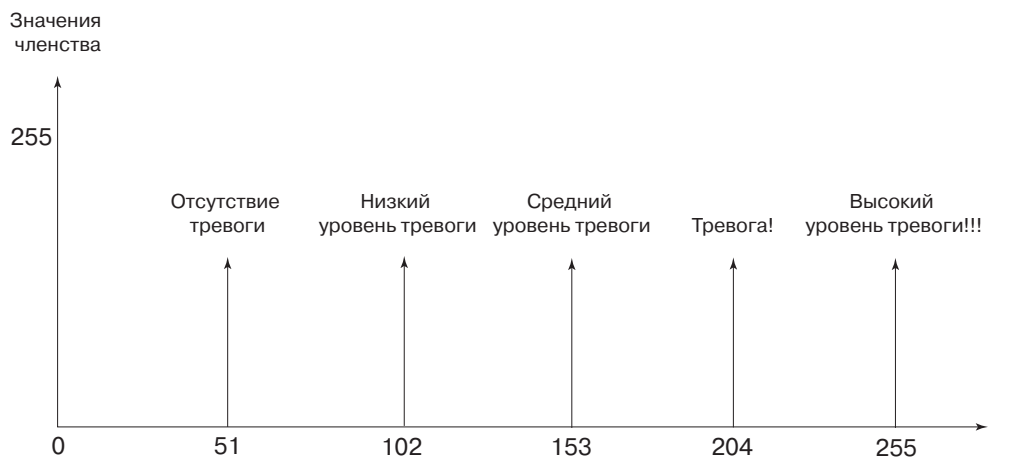


Рис. 7.33. *Выходные функции членства для интерактивного вмешательства*

Значения для профиля поведения	Значения для ключевых слов	Выходные величины
Низкое	Низкое	Нет опасности
Низкое	Среднее	Низкая опасность
Низкое	Высокое	Опасность
Среднее	Низкое	Низкая опасность
Среднее	Среднее	Средняя опасность
Среднее	Высокое	Опасность
Высокое	Низкое	Средняя опасность
Высокое	Среднее	Опасность
Высокое	Высокое	Высокая опасность

Таблица 7.1. *Правила фаззификации, позволяющие преобразовать входные переменные в выходные*

Рис. 7.34. *Процедуры передачи членства оценивают, используя метод Минимакса*

```
void _HC12Setup(void)
{
COPCTL = 0x00;       // выключить сторожевой таймер COP
}
//*********************************************************************
//delay: подпрограмма задержки
//*********************************************************************

void delayfvoid)
     {          //подпрограмма задержки
volatile unsigned n, m;
m = 10;
     do
     {
     n = 0;
     do
          {
          n--;
          } while(n);
     m-- ;
     } while(m);
}
//*********************************************************************
// status_wait: время ожидания подпрограмма установки связи с ЖКД
//*********************************************************************

void status_wait(void)
{          //время ожидания подпрограмма установки связи с ЖК дисплеем
char temp = 0x00;

DDRA = 0x00;
PORTB = 0xF9;
while((temp & 0x03) != 0x03)
{
PORTB = 0xFF;
temp = PORTA;
PORTB = 0xF9;
}
PORTB = 0xFF;
DDRA = 0xFF;
}
//*********************************************************************
// command: пересылка команд на ЖК  дисплей
//*********************************************************************
void command(unsigned char n)
{            // пересылка команд на ЖК дисплей
status_wait();
PORTA = n;
PORTB = 0xFF;
PORTB = PORTB & 0xFA;
PORTB = 0xFF;
}
//*********************************************************************
```

```
//*********************************************************************
//data: пересылка данных на ЖК джисплей
//*********************************************************************

void data(unsigned char n)
{            // пересылка данных на ЖК дисплей
status_wait();
PORTA = n;
PORTB = PORTB & 0xF2;
PORTB = 0xFF;
}

//*********************************************************************
// LCD_char: функция пересылки символа на ЖК дисплей
//*********************************************************************

void LCD_char(unsigned char n)
{
     // функция пересылки символа на ЖК дисплей
data(n - 0x20);
command(0xC0);
}

//*********************************************************************
// newline: пересылка новой строки на ЖК дисплей
//*********************************************************************

void newline(void)
{
     // пересылка новой строки на ЖК дисплей
int i ;

for(i=0; i<16; i++)
     LCD_char (' ')
}

//*********************************************************************
// LCD_output: пересылка последовательности символов на ЖК дисплей
//*********************************************************************

void LCD_output(char s[])
{          // пересылка последовательности символов на ЖК дисплей
int n = 0;

while (s[n] != '\0')
     {
     LCD_char(s[n]);
     + +n;
     }
}
//*********************************************************************
// Reset_cursor: возврат курсора
//*********************************************************************
```

```
void Reset_cursor(void)
{               // возврат курсора
data(0x00);
data(0x10);
command(0x24);
}

//*********************************************************************
******
//Clearscreen: очистка экрана ЖКД
//*********************************************************************
******

void Clearscreen(void)
{               // очистка экрана ЖКД
int i,j;

Reset_cursor();

     for(i=0; i<16; i++)
          for(j=0; j<16; j++)
             LCD_char(' ');
Reset_cursor();
}

//*********************************************************************
// Initlcd: инициализация ЖКД
//*********************************************************************

void Initlcd(void)
{               // инициализация ЖКД
PORTB = 0xEF;        //принудительный сброс
delay();

PORTB = 0xFF;        //все линии команд на высоком уровне
status_wait();

command(0x80);       // установить режим текста
data(0x00);     // установить младший байт адреса текста (L)
data(0xl0);     // установить младший байт адреса текста (H )
command(0x40);       //установить адрес команды текста

data(0xl0);     //установить область текста
data(0x00);
command(0x41);
command(0x94);       //включить текстовый дисплей
command(0xA7);       //курсор 8 x 8 позиций
Clearscreenf];
Reset_cursor();
}

//*********************************************************************
// InitMes: начальное сообщение
//*********************************************************************
```

```
void InitMes(void)
{               // начальное сообщение
unsigned char k;

for(k=0; k<3; k++;
     newline();
LCD_output(" Portable HTTP
newline();
LCD_output("    TAD System.
newline();
LCD_output("    version 1.0
}

//*********************************************************************
// numdisplay: отображение чисел на ЖК дисплее

//*********************************************************************


void numdisplay(char s)
{               //отображение чисел на ЖК дисплее
char к;

newline();
к = s;
s = s>>4;

if(s > 0x08)
     data(s + 0x17);
else
     data(s + 0x10};

command(0xC0);
к = к & 0x0F;

if(к > 0x08)
     data(к + 0x17);
else
     data(k + 0x10);

command(0xC0);
}

//*********************************************************************
// Секция данных - инициализация табличных данных
//*********************************************************************


#pragma abs_address 0x3000

char BeP[12] = {0x00, 0x70, 0x00, 0x10,
                0x40, OxCO, 0x10, 0x10,
                0x90, OxFF, 0x10, 0x00};
```

```
char KeM[12] = {0x00, 0x70, 0x00, 0x10,
                0x40, OxCO, 0x10, 0x10,
                0x90, OxFF, 0x10, 0x00};

char OT[5] = {0x40, 0x60, 0x80, OxAO,OxCO};

char IMV[6] =  {0x00, 0x00, 0x00,
                0x00, 0x00, 0x00};

char OMV[5] =  {0x00, 0x00, 0x00, 0x00, 0x00};

// правило
char rules[45] =  {0x00,0x03,0xFE,0x06, 0xFE,
                   0x00,0x04,0xFE,0x07, 0xFE,
                   0x00,0x05,0xFE,0x08, 0xFE,
                   0x01,0x03,0xFE,0x07, 0xFE,
                   0x01,0x04,0xFE,0x08, 0xFE,
                   0x01,0x05,0xFE,0x09, 0xFE,
                   0x02,0x03,0xFE,0x08, 0xFE,
                   0x02,0x04,0xFE,0x09, 0xFE,
                   0x02,0x05,0xFE,Ox0A, 0xFF};

char result[3] =  (0x00, 0x00, 0x00};
#pragma end_abs_address

//*********************************************************************
//Основная программа
//*********************************************************************

void main(void)
{
int index;
char temp = 0x00

             /* определение интерактивного туннелирования */
asm("LDX #$3000");
asm("LDY #$301D");
asm("LDAA $4000");   //оценка профиля поведения
asm("MEM");
asm("MEM");
asm("MEM");          //фаззификация

asm("LDAA $4001");   //оценка членства ключевых слов
asm("MEM");
asm("MEM");
asm("MEM");          //фаззификация

asm("LDY #$301D");
asm("LDX #$3028");
asm("LDAA #$FF");    //инициализация минимума и   бита V
asm("REV");          //применение правил фаззификации

asm("LDX #$3018");   //дефаззификация
asm("LDY #$3023");
asmC'LDAB #$05");
```

```
asm("WAV");
asm("EDIV");
asm("TFR Y,D");
asm("STAB $3 055");  //сохранение результата

PORTB - Oxff;
DDRB = Oxff;

delay2();
PORTB = 0x7F;        //проверка платы с использованием ЖКД
delay2();
PORTB = OxFF;

                     /*Определение туннелирования скриптов*/

asm("LDX #$3000");
asm("LDY #$301D");
asm("LDAA $4002");   //оценка профиля поведения

asm("MEM");
asm("MEM");
asm("MEM");          //фаззификация

asm("LDAA #$PF");    //оценка членства ключевых слов
asm("MEM");
asm("MEM");
asm("MEM");          //фаззификация

asm("LDY #$301D");
asm("$3028");
asm("$4003");        //инициализация минимума и бита V

asm("REV");          //применение правил фаззификации

asm("LDX  #$3018");  //дефаззификация
asm("LDY  #$3023");
asm("LDAB  #$05") ;
asm("WAV");
asm("EDIV");
asm("TFR  Y,D");
asm("STAB $3056");   //сохранение результата

PORTB = Oxff;
DDRB = Oxff;

delay2();
PORTB = 0x7F;        //проверка платы с использованием ЖКД
delay2();
PORTB = 0xFF;

             /*Проверка внедрения потоков*/
asm("LDX #$3000");
asm("LDY #$301D");
asm("LDAA $4004");   //оценка профиля поведения
asm("MEM");
```

```
asmf"MEM");
asm("MEM");          //фаззификация

asm("LDAA $4005");   //оценка членства для ключевых слов
asmf"MEM");
asm("MEM");
asmf"MEM");          //фаззификация

asmC'LDY #$301D");
asmC'LDX #$3028");

asmf"LDAA $4003");   //инициализация минимума и бита V
asmf"REV");          //применение правил фаззификации

asmC'LDX #$3018");   //дефаззификация
asmfLDY #$3023");
asral"LDAB #$05");
asm("WAV");
asmf"EDIV");
asm|"TFR Y,D");
asm("STAB $3057");   //сохранение результата

PORTB = 0xff;
DDRB = 0xff;

delay2();
PORTB = 0x7F;        //проверка платы с использованием ЖКД
delay2();
PORTB = 0xFF;

                     //Конфигурация ЖКД
DDRA = 0xFF;
PORTB = 0xFF,

Initlcd0;            //Инициализация ЖКД
InitMes();           //Инициализация сообщений

delay2();

Clearscreen();       //Очистить экран ЖКД
Reset_cursor;        //Возврат курсора ЖКД

newline();           //Создать новую строку на ЖКД
newline();

LCD_OUtput("IA:  ");
numdisplay(result[0]);

if (result[0] > 0xA0)        //Вывести на дисплей предупреждение
     LCD_output(" High Alert");
     else if (result[0] > 0x60)
     LCD_output(" Med Alert");
else
     LCD_output(" Low Alert");
```

```
newline();

LCD_output("SA:     ");
numdisplay(result[1]);

if (result[0] > 0xA0)
     LCD_output(" High Alert");
else if (result[1] > 0x60)
     LCD_output(" Med Alert");
else
     LCD_output(" Low Alert");

newline();

LCD_output{"S: ");
nuradisplay(result[2]);

if (result[2] > 0xA0)
     LCD_outpiit (" High Alert");
else if (result[0] > 0x60)
     LCD_output(" Med Alert");
else
     LCD_output(" Low Alert");
}

//=====================================================================
```

7.6.8. Некоторые комментарии

Вы можете задать вопрос о необходимости иметь отдельную микроконтроллерную систему HTTP TAD. Почему бы просто не включить функцию системы HTTP TAD в состав стандартной системы обнаружения «злоумышленника»? Цель упражнения состоит в том, чтобы показать большие возможности МК HCS12 и использование их в ряде проектов. Конечно, система, описанная в этом разделе, может быть включена в состав большой системы обнаружения «злоумышленника». Одно из преимуществ создания отдельной системы – мобильность. Это обеспечивает администраторов защиты гибким инструментом, чтобы изменять одиночную переносную систему и проверять несколько модулей внутри сети. Материалы для более подробного изучения систем информационной защиты мы вынесли в раздел «Что еще прочесть» в конце главы.

7.7. Электронная версия игры в «15»

7.7.1. Описание проекта

Наверное, вы когда-нибудь передвигали фишки игры в «пятнадцать», показанной на рис.7.35. Этой игрой дети развлекаются дома, в летних лагерях и в поездках на заднем сиденье автомобиля. Мы были свидетелями перевода целого ряда игр с пе-

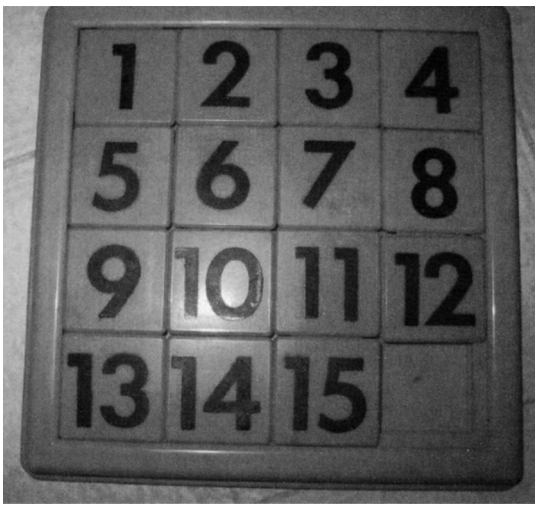


Рис. 7.35. *Игра в «15»*

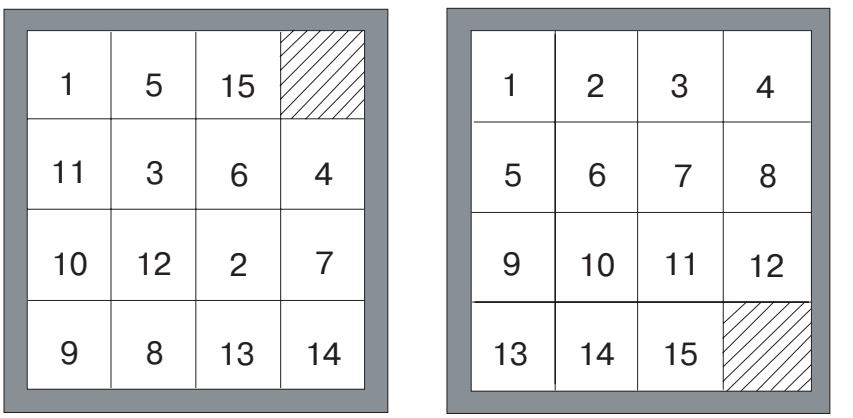


Рис. 7.36. *Игра в «15». Левое поле показывает произвольное начальное состояние фишек, а на правом приведена окончательная позиция, достижение которой является целью игры*

редвижением фишек в электронную форму. В этом разделе мы представляем электронную версию игры в «пятнадцать». Проект был создан Скоттом Льюисом в качестве работы для получения звания «senior project design» старшего программиста.

Цель игры заключается в следующем: необходимо упорядочить данный произвольно расположенный набор из 15 пронумерованных фишек и одного пустого промежутка, расположив фишки в порядке возрастания номеров (рис.7.36 внизу справа). Сделать это необходимо, многократно повторяя операцию передвижения одной из соседних фишек на пустое место.

Цель нашего проекта заключается в создании электронной версии игры. Для этого необходимо обеспечить средства для допустимого изменения положения фишек и устройство для отображения текущего положения фишек после каждой операции.

Требования к системе для игры в «15» следующие:
1. Использовать микроконтроллер;
2. Позволить пользователю идентифицировать номер фишки с помощью устройства ввода данных;
3. Позволить пользователю перемещать выбранную фишку на пустое поле;
4. Отображать текущие расположения всех фишек.

7.7.2. Системы HCS12 используемые в проекте

Этот проект использует различные модули 68HC12, и интерфейс, связывающий их с несколькими переключателями и графическим ЖК дисплеем.

7.7.3. Основы теории

Система использует отладочную плату 68HC12B32EVB, графический ЖК дисплей AND 1391, ИС внешнего ОЗУ (RAM 6264) и внешнего ПЗУ (EPROM 27256), а также программируемую ИС (GAL16V8), набор кнопок и ИС триггера защелки 74HC373. На рис. 7.37 показана структурная схема игровой системы. Хотя мы могли бы использовать внутреннюю память 68HC12, но для расширения опыта читателя мы использовали внешнюю память и расширенный режим работы микроконтроллера. Для создания соответствующих сигналов управления и декодирования адреса памяти, мы использовали программируемую ИС GAL16V8, включив ее между микросхемами памяти и МК. Интегральная схема GAL16V8 программируется так, чтобы ОЗУ размещалось в адресном пространстве начиная с $ 2000 и заканчивая $3FFF, а программируемое ПЗУ – в ячейках памяти с адресами $4000...$7FFF. Как вы увидите, программа, представленная в этом разделе, запускается из ОЗУ. Разместив программу во внешнем ПЗУ, можно сделать систему автономной.

7.7.4. Схемное решение, структура программы и блок схема алгоритма

На рис.7.38 показана структура микроконтроллерной системы для игры в «15». Встроенные порты отладочной платы 68HC912B32EVB используются, чтобы получать и послать данные от вспомогательной клавиатуры и модуля ЖК дисплея. Принципиальная схема этой системы с расширением памяти приведена на рис.7.39.

На рис 7.40 показана структура программы для игры в «15», в которой главная программа обращается к шести субмодулям. На рисунке показан также ряд функций, выполняемых каждым субмодулем. Соответствующая блок схема алгоритма приведена на рис.7.41.

7.7.5. О компонентах системы

Интеллектуальный точечный графический дисплей AND 1391– это полная точечная матрица ЖКД со встроенным модулем контроллера дисплея и ОЗУ буфера экрана. Он может выводить 21 символ на каждой из 18 строк дисплея. Вы можете прочесть об этом дисплее и функциях его обеспечения в главе 5.

7.7.6. Программный код

```
//Имя файла: sliding.c
//Программа: Игра в "15"
//Авторы: Scott Lewis, Daniel Pack,
//Дата создания: Начата 27 апреля 2004
//окончена 7 Мая 2004
// Описание: Эта программа реализует игру в "15". Цель игры
//           заключается в том, чтобы выстроить фишки по поряду номеров. Вы
//           можете только перемещать на пустое место любую соседнюю фишку.
//           Программа постоянно находится в памяти системы по адресу
//           $2000. Программа отображает исходное состояние фишек, затем
//           показывает состояние к которому необходимо прийти в
//           результате. Программа размещает все фишки в случайном
//           порядке, и затем позволяет пользователю начать игру

// Конфигурация системы
//   Программа: 0x2000
//   Данные: 0x3500
//   Стек:   0x4000
//
//   PDLC0: /WR          PP0 Data 0     PS0: Не используется
//   PDLC1: /RD          PP1 Data 1     PS1: Не используется
//   PDLC2: /CE          PP2 Data 2     PS2: Кнопка "Влево"
//   PDLC3: C/D          PP3 Data 3     PS3: Кнопка "Выбор"
//   PDLC4: /Reset       PP4 Data 4     PS4: Кнопка "Вниз"
//   PDLC5: NC           PP5 Data 5     PS5: Кнопка "Вправо"
//   PDLC6: NC           PP6 Data 6     PS6: Кнопка "Вверх"
//   PDLC1: NC           PP7 Data 7     PS7: Не используется
//
//   Выводы ЖКД

//         GND       - 2       1   - GND
```

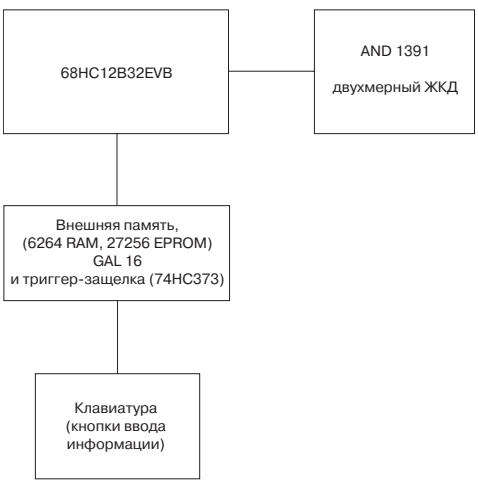


Рис. 7.37. *Блок-диаграмма микропроцессорной системы для игры в «15»*

```
//          -14V        - 3         4   - +5V
//          /WR         - 5         6   - /RD
//          /CE         - 7         8   - C/D
//          NC          - 9         10  - /Reset
//          DO          - 11        12  - Dl
//          D2          - 13        14  - D3
//          D4          - 15        16  - D5
//          D6          - 17        18  - D7
//          GND         - 19        20  - NC

//*****************************************************

#include <912b32.h> // описание портов - header file
                    // приведен в приложении

//*********************************************************************
// Постоянные
//*********************************************************************
#define ARRAY_MAX 15
#define ZERO    0x00

#define L_BUTT0N 0x04        // сигналы кнопок

#define R_BUTT0N 0x20
#define U_BUTTOH 0x40
#define D_BUTTON 0x10
```

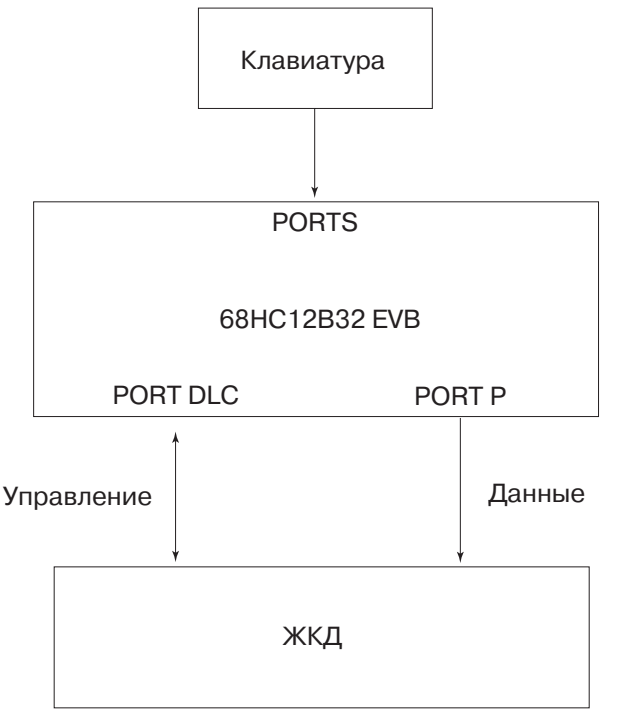


Рис. 7.38. *Структура микропроцессорной системы для игры в «15»*

```
#define S_BUTTON 0x08

#define LEFT 1        // выбор направления
#define RIGHT 2
#define UP 3
#define DOWN 4
#define SELECT 5

#define SIZE 4        // Размер строки/колонки
```

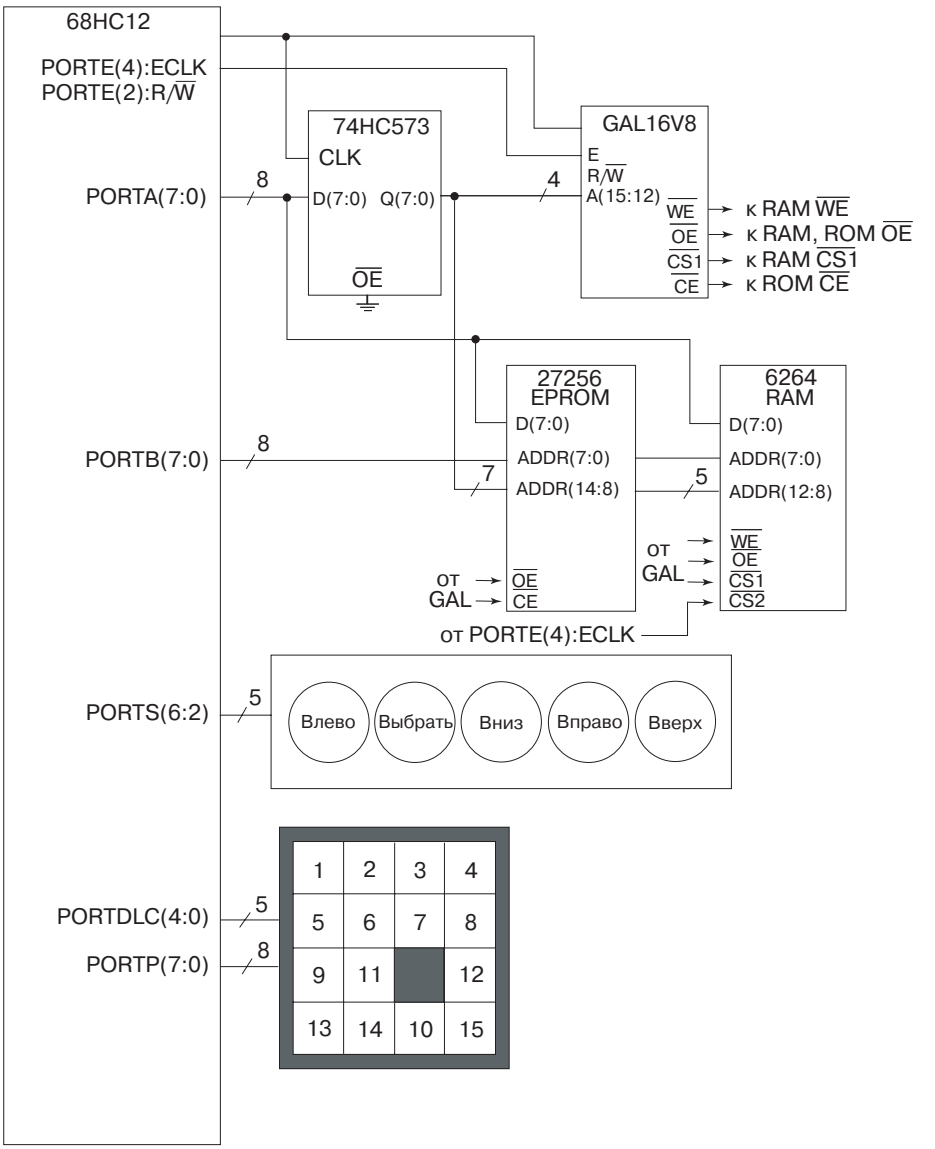
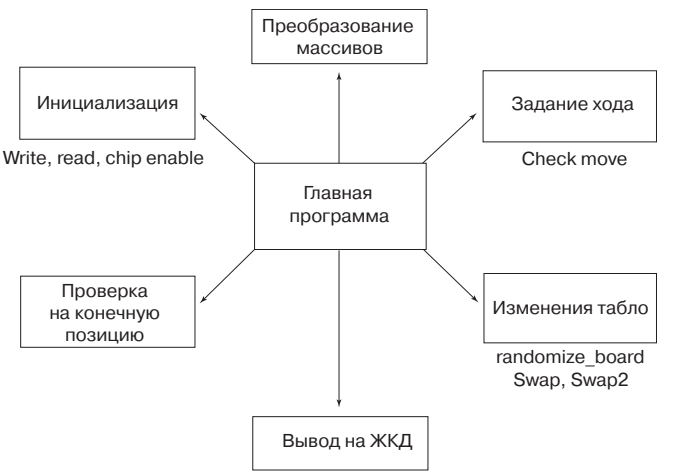


Рис. 7.39. *Принципиальная схема системы для игры в «15»*

```
//*********************************************************************
// Используемые функции
//*********************************************************************

int check_win(int array[ARRAY_MAX+1]);

void convert_multi_single(int A[SIZE][SIZE], int B[SIZE][SIZE]);
void convert_single_multi(int A[ARRAY_MAX+1], int B[SIZE] [SIZE]);
void display_board
                              int A[SIZE][SIZE], int row, int col, int
direction);
void display_board2(int board_array[ARRAY__MAX+l]);
void display_error(int n);
void display_intro (void) ;
void display_win(void) ;
void get_move(int *direction, int *row, int *col, int *select);
void randorni2e_board(int board__array []);
void swap(int row, int col, int new_row, int new_col,
               int array[SIZE][SIZE]);
void swap2(int from, int to, int array[ARRAY_MAX+1]);
void try_move(int move, int row, int col, int array[SIZE][SIZE]);
unsigned char mode(unsigned char num, int modulus);
void LCD_output(char s[]);
void int2char(int i);
```



Clear Screen	Display Current Board
Display Message	Display Winning Pattern
LCD_output	Reset Cursor
Status_wait	Newline
int2char	data
pause	command
LCD_char	delay

Рис.7.40. *Структура программы для игры в «15»*

```
void pause(void); void delay(int usec);
void enable(void);
void initialize_LCD(void);
void command(unsigned char n);

void data(unsigned char n);
void write(void);
void read(void);
void status_wait(void);
void LCD_char (unsigned char n) ;
void Clearscreen(void);
void newline(void) ;
void Reset_cursor(void) ;
```

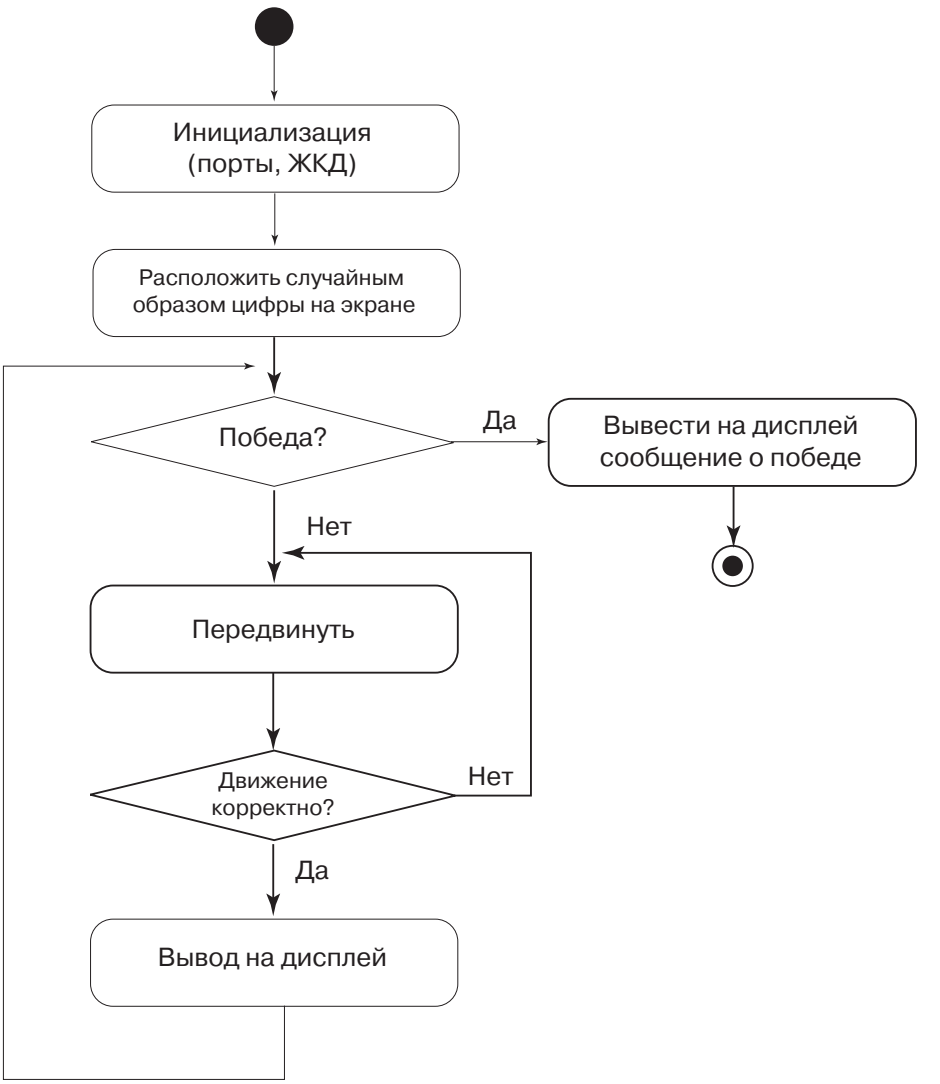


Рис. 7.41. *Блок схема алгоритма программы для игры в «15»*

```
//*****************************************************************
//Переменные
//*****************************************************************

#pragma abs_address 0x3600
static int win_array[ARRAY_MAX+1]
                          = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0};
#pragma end_abs_address

#prama abs_address 0x3500
int board_array_single [ARRAY_MAX+1];
int board_array_multi[SIZE][SIZE];
int win; //использовать как булевы переменные
int direction; //направление движения
int row; //выделить ряд
int col; //выделить колонку
int select;  //если кнопка нажата
int i; //текущая переменная
#pragma end_abs_address
//*****************************************************************
// Основная программа
//*****************************************************************
void main()
{
win = 0;        //инициализировать все переменные
direction = 0;
row = 1;
col = 1;
select = 0;
i = 0;
DDRDLC = 0x1F        //конфигурировать порт  DDRDLC как выходной
DDRP = 0xFF     // конфигурировать линии порта P как выходные
DDRS - 0x00     // конфигурировать линии порта S как входные
TSCR = 0x80     //включить таймер
initialize_LCD();            //инициализировать ЖКД
Clearscreen();
Reset cursor();
display_intro();
for(i=0; i< ARRAY JtfAX+l; i++)
  board_array_single[i] - win_array[i];
convert_single_multi(board_array_single, board_array_multi);
display_board(board_array_multi,row,col, direction);
pause ();    // ожидание, пока пользователь не нажмет кнопку X
LCD_output(''Now the board is");       // вывод сообщения
LCD_output(''randomized. You ");
LCD_output(''may begin by     ");
LCD_output(''choosing a piece");
LCD_output(''to move, and its");
LCD_output(''direction.       ");
newline();
pause();
randornize_board (board_array_single); // Случайный выбор исходного
                                       // положения фишек
convert_single_multi(board_array_single, board_array_multi);
display_board(board_array_multi, row, col, direction);
```

```
while(win ==0)
     {       //цикл повторяется до успешного окончания игры
     while (select = = 0)
     {       //если была нажата кнопка x, то задается направ-
             // ление движения ряд и колонка выделенной фишки
get_move(^direction, &row, &col, ^select);
Reset_cursor();      // установить курсор в верхнее положение
if(select = = 0)
  display_board(board_array_multi,row,col, direction);
}
          //проверить корректность движения, повторить его, если
          // оно некорректно, или вывести сообщение об ошибке
try__move (direction, row-1, col-1 ,board_array_multi);
select = 0;
convert_multi_single (board_array_multi, board__array_single);
win  = check__win(board_array_single);
Clearscreen();
             // показать текущее состояние игры
display_board(board_array_multi,row,col, direction);
}
display_win{};  // вывести сообщение об успешном окончании игры
}
//*****************************************************************
// display_intro: DISPLAY INTRO MESSAGE
//*****************************************************************

void display_intro
{
newline();
LCD_output('' WELCOME          ");
LCD_output('' TO               ");
LCD_output('' SLIDING PUZZLE   ");

new_line();
pause();

LCD_output(''The object of     ");
LCD_output(''this game is to   ");
LCD_output(''move each #'ed    ");
LCD_output(''puzzle piece so   ");
LCD_output(''that you end up   ");
LCD_output(''in the order      ");
LCD_output(''seen below. The   ");
LCD_output(''star shows the    ");
LCD_output(''current piece     ");
LCD_output(''selected. You     ");
LCD_output(''can choose a      ");
LCD_output(''different piece   ");
LCD_output('' by using the     ");
LCD_output('' arrow buttons    ");

newline();
pause() ;

LCD_output(''and select the    ");
```

```
LCD_output(''piece you want    ");
LCD_output(''to move by        ");
LCD_output(''pressing the X    ");
// кнопка "Выбор"
LCD_output(''button. Choose    ");
LCD_output(''the direction to ");
LCD_output(''move that piece   ");
LCD_output(''with the arrows. ");

new_line() ;

LCD_output(''WINDING            ");
LCD_output(''CONFIGURATION:    ");

newline();
}
//*****************************************************************
// display_win: ВЫВОД СООБЩЕНИЯ О ПОБЕДЕ
//*****************************************************************

void display_win()
{
LCD_output(''    YOU WIN!!!     ");
LCD_output(''CONGRATULATIONS   ");
}

//*****************************************************************
// get_rnove: ДВИЖЕНИЕ ФИШКИ: задается позиция фишки, выбранной игро-
ком и направление ее движения
//*****************************************************************

void get_rnove (int * direct ion, int *row, int *col, int * select)
{
int n = 0;
int button = 0;
unsigned char temp = ZERO;
newline();
LCD_output(''Choose move or   ");
LCD_output(''select piece:    ") ;

while(button = = 0)
{             // цикл выполняется, пока нажата кнопка
temp = PORTS;
temp = temp & 0x7C

switch (temp)   //какая кнопка нажата?
     {
     case L_BUTTON:
          button = LEFT;
          break;
     case R_BUTTON:
          button = RIGHT;
          break;
     case U_BUTTON:
          button = UP
```

```
            break;
      case D_BUTTON:
            button = DOWN;
            break;
      case S_BUTTON:
            button = SELECT;
            break;
      } // конец цикла switch
}//конец цикла  while

n = 0;
switch (button)      //реакция на нажатие кнопки
      {
      case UP:
            if (*row > 1)
            *row -= 1;

      else
            display_error(UP);
            break;

      case DOWN:
            if (*row < SIZE)
            *row += 1;
            else
            display_error(DOWN);
            break;

      case LEFT:
            if <*col > 1)
            *col -= 1;
            else
            display_error (LEFT) ;
            break;

      case RIGHT:
            if (*col < SIZE)
            *col += 1
            else
            display_error(RIGHT);
            break;

      case SELECT:
            *select = 1;
            LCD_output(''Pick a direction");
            *direction = 0;

            while (*direction == 0)
               {
               temp = PORTS;
               temp = temp &. 0x7C;

               switch(temp)
                  {
                  case L_BUTTON:
```

```
                    *direction = LEFT;
                    break;

               case R_BUTTON:
                    *direction = RIGHT;
                    break;

               case U_BUTTON:
                    *direction = UP;
                    break;

               case D_BUTTON:
                    *direction = DOWN;
                    break;
               }
            }
          break;
     }
}
//*********************************************************************
// randoraize_board: ВЫБОР СЛУЧАЙНОГО ИСХОДНОГО СОСТОЯНИЯ ФИШЕК
//*********************************************************************

void randoraize_board(int board_array[])
{
int temp = 0;
int i;
unsigned char temp2 - 0x00;
for(i=0; i<ARRAY_MAX+l; i++)
     {
     temp2 = TCNTL;
     temp - mod(temp2,15);   //случайное значение  using the TCNT counter
     swap2(i,temp, board_array);
     }
}
//*********************************************************************
// MOD: МАТЕМАТИЧЕСКАЯ ФУНКЦИЯ
//*********************************************************************

unsigned char mod(unsigned char num, int modulus)
{
while ((num - modulus) > 0)
     num - num -modulus;

return num;
}

//*********************************************************************
// display_board2: Выводит табло, как одну колонку значений
//*********************************************************************

void display_board2(int board_array[ARRAY_MAX+1])
{
int = 0;
int n;
```

```
for (i=0; i<ARRAY_MAX+l; i++)
     {
     n = board_array[i];
     int2char(n);
     }
LCD__output ( ' ' \n" ) ;
}

//*********************************************************************
// display_board: Выводит табло как массив 4x4
//*********************************************************************

void display_board(int A[SIZE][SIZE], int row, int col,
                              int direction)
{
#pragma abs_address 0x0800
int i;
int j;
int num;
#pragma end_abs_address

newline();
LCD_output(''|   Column    ");
LCD_output(''| 1   2   3   4 ");
LCD_output(''-------------------");

for (i=0; i < SIZE; i++
     {
     j=0;
     switch(i)
          {
          case 0: LCD_output(''R 1 |'')
               break;

          case 1: LCD_output(''o 2 |'')
               break;

          case 2: LCD_output(''w 3 |'')
               break;

          case 3: LCD_output(''  4 |'')
               break;

          for (j=0; j < SIZE; j++)
             {
             num = A[i] [ j];
             if (num = = 0)

               LCD_output('' ");
             else
               int2char(num);
             if ((i+1 == row) && (j+1) == col))
               LCD_putput(''*");
             else
               LCD_outpiat ('' ");
```

```
            }
      }
newline();
LCD_output(''You are at (R,C)");
LCD_output(''    (");
int2char(row);
LCD_output(",");
int2char(col);
LCD_output('') ="j;
int2char(A[row-l];
LCD_output(;'   ");
newline();
}

//*********************************************************************
// INT2CHAR: задается как integer и выводится на ЖКД в виде
// двух символов
//*********************************************************************

void int2char(int i)
{
if (i > 9)
      {
      LCD_output(''Iм);
      i -= 10;
      }
else
      {
      LCD_output('; ");
      }

switch(i)
      {
      case 0: LCD_output(''0");
              break;

      case 1: LCD_output{''1");
              break;

      case 2: LCD_output(''2");
              break;

      case 3: LCD_output (''3");
              break;

      case 4: LCD_output(''4");
              break;

      case 5: LCD_output(''5");
              break;

      case 6: LCD_output(''6");
              break;

      case 7: LCD_output(''7");
```

```
            break;

     case 8: LCD_output(''8");
            break;

     case 9: LCD_output(''9");
            break;
  }
}

//*********************************************************************
//Check_Win: Эта функция проверяет соотвествие текущего положения ситуации
// победы при победе  win - 1, в остальных случаях - 0
//*********************************************************************

Int check_win(int array[ARRAY_MAX+1]
{
int i;
int win - 1;
for (i=0; i<ARRAY_MAX+l; i++);
     {
     if (array[i] 1= win_array[i])
          win - 0;
     }
return win;
}

//*********************************************************************
//Convert_multi_single:   Эта функция преобразует двухмерный массив
//в одномерный
//*********************************************************************


int convert_multi_single(int A[SIZE][SIZE], int B[ARRAY_MAX+1])
{
int n = 0;
int i = 0;

int j = 0;
for (i=0; i<SIZE; i++)
     {
     for (j=0; j<SIZE; j
          {
          B[n] = A[i][j] ;
          n++;
          }
     }
}

//*********************************************************************
//Convert_single_multi: Эта функция преобразует одномерный массив
//в двухмерный
//*********************************************************************

void convert_single_multitint A[ARRAY_MAX+1], int B[SIZE][SIZE]
```

```
{
B[0][0] = A[0];
B[0][1] = A[1];
B[0][2] = A[2];
B[0][3] = A[3];
B[1][0] = A[4];
B[1][1] = A[5];
B[1][2] = A[6];
B[1][3] = A[7];
B[2][0] = A[8];
B[2][1] = A[9];
B[2][2] = A[10];
B[2][3] = A[11];
B[3][0] = A[13];
B[3][2] = A[14];
B[3][3] = A[15];
}
//*****************************************************************
// Try_move: Эта функция позволяет игроку определить некорректность
// своего хода. Если он корректен,  то движение выполняется,
// если же некорректен, то выводится соответствующее сообщение
//*****************************************************************
void try_move(int move, int row, int col, int array[SIZE][SIZE])
{
switch(move)
     {
     case UP:
               if ((row-1 >=0) && (array[row-1][col] == 0))
               swap(row,col, row-1,col,array);
            else
               display_error(UP);
            break;

     case DOWN:
               if ( (row+1 <= SIZE) Sc& (array [row+1] [col] == 0))
                    swap(row,col, row+1,col,array);
               else
                    display_error(DOWN); break;

     case LEFT:
               if ((col-1 >=0) && farray[row][col-1] = = 0))
                    swap(row,col, row,col-1,array);
               else
                    display_error(LEFT);
               break;

     case RIGHT:
               if ((col+1 < = SIZE) && (array[row][col+1] == 0))
          swap{row,col, row,col+1,array);
     else
          display_error(RIGHT);
     break;
     }
}
//*****************************************************************
```

```
//*********************************************************************
//Swap: Эта функция заменяет два значения двухмерным масивом.
//*********************************************************************
void swapfint row, int col, int new_row, int new_col,
                              int array[SIZE][SIZE]
{
int temp;
temp = array[row] [col];
array[row][col] = array[new_row][new_col];
array[new_row][new_col] = temp;
}

//*********************************************************************
//Swap2: Эта функция заменяет два значения одномерным масивом.
//*********************************************************************

void swap2(int from, int to, int array[ARRAY_MAX+1]
{
int temp - array[from];
array[from] = array[to];

array[to] - temp;
}

//*********************************************************************
//ERROR: Эта функция выводит сообщения об ошибке
//*********************************************************************

void display_error(int n)
{
LCD_OUtput(''ERROR: ");

switch(n)
     {
     case LEFT:
                    LCD_output(''no move L");
                    break;

     case RIGHT:
                    LCD_output(''no move R");
                    break;

     case UP:
                    LCD_output (''no move U" );
                    break;

     case DOWN:
                    LCD_output(''no move D" );
                    break;
     }

pause();
}

//*********************************************************************
```

```
//*********************************************************************
//LCD_output: Эта функция выводит на дисплей строку
//*********************************************************************

void LCD_output(char s[])
{
int n = 0;

while (s[n] != '\0')
     {
     LCD_char(s[n]);
     ++n;
     }
}

//*********************************************************************
//Pause: Функция реализует ожидание, пока игрок не нажмет кнопку
//              "Выбор"
//*********************************************************************

void pause ()
{
unsigned char c = ZERO;
LCD_output(''(Please press X)"};

while (c != S_BUTTON)
     {
     c = PORTS;
     c = c& 0x7C;

Clearscreen();
     Reset_cursor();
}

//*********************************************************************
//Delay: Эта функция вводит задержку на  n мкс, если входная
// величина равна  n
//*********************************************************************

void delay (int usec)
{
int i,j;

for (i=0;i<usec;   i++)
     {
     for     (j=0; j < 7; j++)
          {
          }
     }
}

//*********************************************************************
//Initialize_LCD: Функция инициализирует ЖКД
//*********************************************************************
```

```
void initialize_LCD(void)
{
char temp = 0x00;
PORTDLC = 0xFF;
PORTDLC = PORTDLC & 0xEF;    // сброс экрана (RESET = 0)
delay(2000)                  // задержка в  2 мс
PORTDLC = 0x7F;              // выключение сброса

write ( );                   //включение записи
command(0x80);
//установка текстового режима

data(0x0);                   //проверка установки слова
data(0xl0);
command(0x40);

data(0xl0);                  // устанавливается размер области текста (1E)
data(0x00);                  // - 0x1000
command(0x41);
               //включается дисплей текста , курсор, выключается мигание
command(0x94);
command(0xA7);               //курсор 8 х 8 точек
}

//*********************************************************************
//Enable: Функция разрешает работу ИС
//*********************************************************************

void enable(void)
{
PORTDLC = PORTDLC | 0x04;    // Установить 1 на линии enable
PORTDLC = PORTDLC & 0xFB;    // Установить 0 на линии enable
}

//*********************************************************************
//Disable: Функция запрещает работу ИС
//*********************************************************************

void disable(void)
{
PORTDLC = PORTDLC | 0x04;
}
//*********************************************************************
//Command: Функция посылает команду отключения на ЖКД
//*********************************************************************

void command(unsigned char n)
{
status_wait();
PORTP = n;
PORTDLC = 0xFF;
PORTDLC = PORTDLC & 0xFE;    // сброс записи
enable;                      // сброс флага CE
```

```
delay(10)                    // задержка не менее 80 нс
disableO;                    // включение флага CE
}

//*********************************************************************
//Data: Функция пересылает данные на ЖКД
//*********************************************************************

void data(unsigned char n) {
status_wait f) ;
PORTP = n;
PORTDLC = OxFF;
PORTDLC - PORTDL>C & 0xF7;   // перевести C/D на низкий уровень
PORTDLC = PORTDLC & 0xFE;    // перевести  WR на низкий уровень
PORTDLC = PORTDLC & 0xFB;
delay(10);
disable();
}
//*********************************************************************
//Write: Функция конфигурирует порт P как выходной
//*********************************************************************

void write()
{
DDRP = 0xFF;
}

//*********************************************************************
//Read: Функция конфигурирует порт P как входной
//*********************************************************************

void read()
{
DDRP = 0x00;
}

//*********************************************************************
//Status_wait: Создает соответствующие задержки между командами ЖКД
//*********************************************************************

void status_wait()
{
char temp = 0x00;
DDRP = 0x00;
PORTDLC = PORTDLC | 0x0F;    // сбросить все
PORTDLC = PORTDLC & 0xFD;    // сброс флага RD
enable();
delay(10);
while ((temp & 0x03) != 0x03)
     {
     temp = PORTP;
     }

disable();
DDRP = 0xFF;
```

```
}

//*********************************************************************
//LCD_char: Функция выводит ASCII код на экран ЖКД
//*********************************************************************

void LCD_char(unsigned char n)
{
data(n-0x20);
command(0xC0);
}

//*********************************************************************
//Clearscreen: Функция очищает экран ЖКД
//*********************************************************************

Void Clearscreen()
{
Int i,j;

Reset_cursor();

for (i=0; i < 16; i++)
     for (j=0; j<16; j++)
          LCD_char(' ');

Reset_cursor();
}

//*********************************************************************
//Newline: Функция выводит пустую строку на экран ЖКД
//*********************************************************************


void newline()
{
int i;

for (i=0; i < 16; i++)
          LCD_char(' ');
}

//*********************************************************************
//Reset_cursor: Функция возвращает курсор ЖКД в начальную позицию
//*********************************************************************

void Reset_cursor()
{
data(0x00);
data(0x10);
command(0x24);
}

//*********************************************************************
```

7.7.7. Некоторые комментарии

Система для игры в «15» использует пять кнопок вспомогательной клавиатуры, чтобы перемещать фишки: стрелку «Вверх», стрелку «Вниз», стрелку «Вправо», стрелку «Влево» и кнопку «Выбор». Кроме того, расположение фишки определяется номерами строки и столбца. Однажды сформированная, система может использоваться для игры взрослыми и детьми.

7.8. Программирование резидентного Flash ПЗУ микроконтроллера B32 в составе платы отладки MC68HC912B32EVB

В главе 4 мы обсуждали конфигурацию памяти МК B32 семейства 68HC12. Как было упомянуто, в состав МК B32 входит память программ типа Flash объемом 32Кб. В установленном на плате отладки MC68HC912B32EVB микроконтроллере в области Flash ПЗУ записана программа монитора D-Bug12. Если вы желаете использовать отведенный под нее объем для записи части рабочей программы, то необходимо стереть программу монитора D-Bug12. Но неплохо было бы сохранить ее, чтобы снова записать в память при необходимости.

Имеется несколько способов, с помощью которых можно перепрограммировать резидентную память МК на плате отладки MC68HC912B32EVB. Все они предполагают использование двух плат отладки, одна из которых реализует функцию интерфейса BDM для связи МК второй платы с ПК с целью программирования микроконтроллера второй платы. Подробная методика организации программирования таким способом изложена в [6].

В этом разделе мы расскажем, как программировать резидентную Flash памят МК семейства 68HC12, воспользовавшись кабелем интерфейса BDM типа CABLE12 фирмы P&E Microcomputer Systems и программным обеспечением PROG12Z FLASH/EEPROM. Заметим, что, несмотря на название, CABLE12 не является лишь соединителем. В его состав входят аппаратные средства интерфейса между ПК и МК семейства 68HC12, которые позволяют осуществлять взаимодействие этих двух устройств в процессе отладки программ и для занесения программы в энергонезависимую память МК.

Конфигурация системы, использующей эти инструментальные средства, показана на рис.7.42. Как видно из рисунка, CABLE12 связан с ведущим ПК через стандартный 25-жильный кабель параллельного порта. С другой стороны CABLE12 связан с программируемой B32 EVB шестижильным BDM кабелем.

Этот кабель подключается к разъему W9 отладочной платы MC68HC912B32EVB. Красный провод кабеля соответствует штырьку 1 на колодке W9.

Программируемый МК B32 EVB связан также с ведущим ПК через последовательный порт связи. Он должен, кроме того, иметь обычные кабели подключения питания (+ 5 В, земля). Дополнительно, в процессе программирования должны быть поданы напряжение + 12 В (Vpp) и земля на колодку W8. Будьте внимательны, не перепутайте полярность при подключении этого питания. Кроме того, колодка W7 должна быть конфигурирована для Vpp.

Как только аппаратные средства сконфигурированы, можно программировать флеш-память EEPROM в следующей последовательности:

1. Подать питание на B32 (+ 5 В и + 12 В).
2. Найти программное обеспечение PROG12Z на вашем компьютере (c: \pemicro\ Progl2z\progl2z).
3. Дважды щелчкнуть на progl2z.
4. Появляется окно Connect Assistant, которое поможет устанавить связь с переходная приставка CABLE12 BDM интерфейс.
   - Проверьте параметры настройки окна, затем нажмите OK.
   - В окне состояния должно появиться сообщение о появлении связи программы prog12z с интерфейсом CABLE12 BDM.
   - Если связь ПК с интерфейсом CABLE12 отсутствует, на экране ПК появится информация для поиска неисправностей.
5. Должно появиться всплывающее меню Specify Programming Algorithm to Use!
   - Выберите правильный программный модуль с расширением `'*.12P'` P&E, чтобы использовать необходимый драйвер программирования. Например, чтобы программировать Flash память объемом 32Кб, входящую в состав МК B32, выберите `912B32_32K.12P`.
   - Окно состояния показывает, что выбранный драйвер проггаммирования загружен.
6. Затем появляется окно Base Address.
   - Вы должны определить базовый адрес программируемой Flash памяти.
   - Информация о базовом адресе обеспечивается в карте памяти программируемого МК, которая, в том числе, приводится в Руководстве пользователя отладочной платы MC68HC912B32EVB *(68HC12 M68EVB912B32 Evaluation Board User's Manual, Таблица 3–5, страницы 3–55)*
   - Адрес начала блока Flash памяти равен $8000.
   - Введите это значение в ПК и нажмите OK.
7. Щелкните SM Show Module, чтобы отобразить текущее содержание модуля Flash памяти в МК.
8. Выполните операцию стирания текущего модуля перед программированием нового модуля во Flash память.
   Предостережение: программа монитора D-BUG12 пока еще во Flash – памяти, это и есть текущий модуль в памяти МК B32. Как только вы стотрете модуль, код монитора отладки D-BUG12 будет потерян.
   Сотрите модуль.
9. Выберите `SS Specify Record` и затем `*.S19` для загрузки в буфер программы программатора на ПК модуля, подлежащего программирования в МК.
10. Выберите PM, чтобы инициировать программирование модуль во Flash память МК.

7.9. Заключение по главе 7

В этой главе мы рассмотрели ряд встроенных микроконтроллерных систем на базе МК 68HC12 и HCS12. В частности мы описали робот, движущийся в лабиринте, лазерный проектор, цифровой вольтметр, стабилизатор скорости вращения

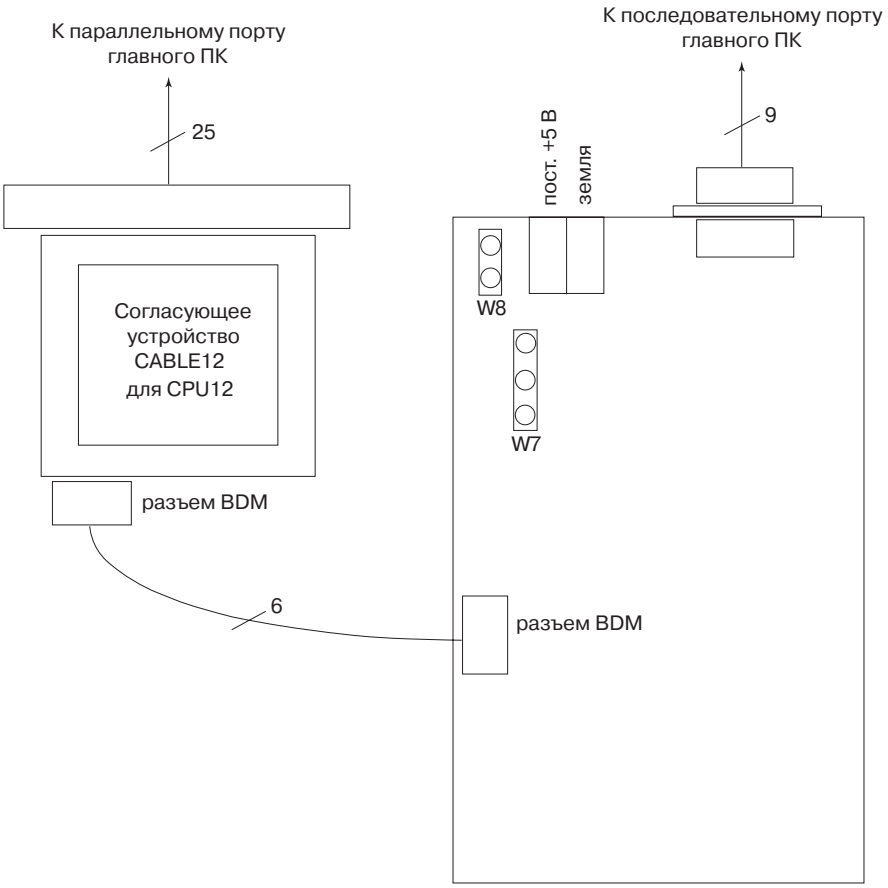


Рис. 7.42. *Схема соединения компонентов для программирования резидентной энергонезависимой памяти МК семейства 68HC12/HCS12*

двигателя с оптическим тахометром, парящий робот, систему защиты компьютерной сети на базе нечеткой логики и электронную версию популярной игры в «15». Для всех этих систем мы привели описание проекта, системные требования, основную информацию, структуру программы, блок схему алгоритма и код программы на языке Си. Хотя функции встроенных систем существенно различаются, мы показали, что для их создания можно использовать одну и ту же методику.

7.10. Что еще прочитать?

1. American National Standards Institute (ANSI) Z136.1, *Safe Use of Lasers* (ANSI Z136.1), 1993.
2. Cooper, W. D. *Electronic Instrumentation and Measurement Techniques.* Upper Saddle River, NJ: Prentice-Hall, 1970.

3. Edmund Industrial Optics, Barrington, NJ, www.edmundoptics.com, 2004.
4. GSI Lumonics, «General Scanning Scanners and Drivers.» www.gsilumonics.com, 2004.
5. Honeywell Sensing and Control, www.honeywell.com/sensing, 2004.
6. Lind, Magnus. *Motorola M68HC912B32EVB Evaluation Board: PC-POD-Target Set Up,* Western Washington University Electronics Engineering Technology, Bellingham, WA, http://eet.etec.www.edu.
7. 68HC12 M68EVB912B32 Evaluation Board User's Manual, 68EVB912B32 UM/D, Motorola Inc., 1997.
8. LINOS Photonics, Milford, MA, www.linos.com. Newport Corporation Irvine, CA, www.newport.com, 2004.
9. Pack, D. J. and S. F. Barrett. *68HC12 Microcontroller: Theory and Applications.* Upper Saddle River, NJ: Prentice-Hall, 2002.
10. Pack, D. J., W. Strelein, S. Webster, and R. Cunningham. «Detecting HTTP Tunneling Activities.» Paper presented at the annual Information Assurance Workshop, West Point, NY, June 2002.
11. Servo-Тек, «Encoders and Other Position/Velocity Sensors for Motion Control, www. servotek.com, 2004.
12. Vij, D. R. and K. Mahesh. *Medical Applications of Lasers.* Kluwer Academic Publishers, 2002.
13. Vincent Associates, «Uniblitz Electronic Drive Equipment and Shutters.» www.uniblitz. com, 2004.

7.11. Вопросы и задания

Основные

1. Опишите принцип работы ИК локатора (пары излучател-приемник), который используется роботом для обнаружения стенок лабиринта.
2. Зачем нужен роботу датчик Холла?
3. Опишите принцип работы ЦАП с параллельным и последовательным интерфейсом.
4. Что такое лазер?
5. Что такое гальванометр?
6. Опишите два способа увеличения разрешающей способности модуля ATD в системах на базе МК 68HC12.
7. Опишите принцип действия оптического кодера.
8. Для чего во встроенных системах применяется прерывания от модуля меток реального времени RTI ?
9. Какие преимущества можно извлечь, применяя при проектировании функциональные схемы системы, структуры программы и блок-схемы алгоритма?
10. В чем различия между системными требованиями и параметрами системы при проектировании встроенных МП систем?
11. Какая процедура может помочь в выборе конкретной модели МК семейства 68HC12, удовлетворяющего требованиям проекта, из большого числа модификаций МК этой серии?

Более сложные

1. Создайте блок-схемы алгоритмов для каждой функции, используемой системой робота, движущегося в лабиринте (раздел 7.1.5).
2. Разработайте эксперимент, позволяющий точно установить, удовлетворяет ли проектному заданию время запаздывания, формируемое устройством задержки в лазерном проекторе.
3. Каковы различия между лазерами различных типов?
4. Если удалить инверторы 74HC04 из структуры входного интерфейса лазерной системы, то как можно обеспечить программную корректировку такого изменения?
5. Пересчитайте значения сопротивлений и Vref, чтобы обеспечить работу DAC0808LCN при питании этой ИС напряжением ± 4 В.
6. Разработайте структуру программы и блок-схему алгоритма для поддержки функций цифрового вольтметра.
7. Какова разрешающая способность 8-разрядного цифрового вольтметра, описанного в разделе 7.3? Чем она определяется? Опишите метод, позволяющий увеличить разрешающую способность. Подсказка: вспомните назначение бита S10BM в регистре ATDCTL4. В чем заключается решение проблемы?
8. В разделе, описывающем проект стабилизации частоты вращения двигателя, мы указали, что каждое инкрементирование коэффициента заполнения ШИМ приводит к изменению скорости приблизительно на 8.5 об/мин. Вы согласны с результатами этого анализа? Подтвердите ваш ответ вычислениями.
9. Создайте блок-схемы алгоритмов для каждой функции, используемой системой стабилизации частоты вращения двигателя, рассмотренной в разделе 7.4.5.
10. Разработайте методику испытаний системы стабилизации частоты вращения двигателя, основанную на восходящих методах проектирования.

Исследовательские

1. Замените параллельный интерфейс цифро-аналогового преобразователя DAC0808LCN в устройстве управления лазерным проектором на последовательный в стандарте SPI. Внесите изменения в схему электрических соединений для ЦАП и также в текст функции `move-laser (x,y)` для программного обслуживания ЦАП.
2. Измените код, содержащийся в `laser.c` (раздел 7.2.6) таким образом, чтобы выбранный образ постоянно воспроизводился, пока не будет выбран другой.
3. Запишите функцию для каждого образа в `laser.c` (раздел 7.2.6).
4. Измените программное обеспечение для цифрового вольтметра, чтобы осуществить работу с 10-разрядным кодом оцифровки модулем ATD.
5. Разработайте и реализуйте метеостанцию с тремя каналами измерения: окружающей температуры, относительной влажности и барометрического давления. Отобразите каждую из измеренных величин через интервал в 3 секунды на ЖК индикаторе. Разработайте структуру программы, блок-схемы алгоритмов, код программы и методику проверки для метеостанции.

Глава 8

ОПЕРАЦИОННЫЕ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

ПОСЛЕ ИЗУЧЕНИЯ ГЛАВЫ ВЫ СМОЖЕТЕ:

ЦЕЛИ

- Подробно описать понятия и термины, касающиеся операционных систем, реального времени ОСРВ (RTS – Real Time Sperating system).
- Понять принципы работы ОСРВ.
- Различать жесткие, твердые и мягкие ОСРВ.
- Описать основные свойства записи, списков связей, стеков и очередей.
- Дать пределение динамическому распределению памяти и описать связанные с ним преимущества и недостатки.
- Описать методы управления задачами с помощью блока управления задачами.
- Объяснить важность системных таблиц, блоков управления устройствами и диспетчера при реализации ОСРВ.
- Объяснить различия между алгоритмами планирования ОСРВ.
- Рассказать о проблемах, связанных с ОСРВ.

8.1. Рассказ: официант – «живая» операционная система реального времени 518
8.2. Что является целью ОСРВ? 519
8.3. Обзор комцепций 522
8.4. Основные понятия 545
8.5. Типы операционных систем реального времени 560
8.6. Проблемы ОСРВ 565
8.7. Выполнение операционной системы реального времени 569
8.8. Пример применения: ОСРВ циклического опроса 569
8.9. Другая прикладка программа: цикл опроса с прерыванием 584
8.10. Сложное прикладное устройство: имитатор ОСРВ 585
8.11. Заключение по главе 8 591
8.12. Что еще почитать? 591
8.13. Вопросы и задания 592

Операционные системы реального времени достаточно давно и успешно используются в промышленных программируемых контроллерах и во встраиваемых приложениях на основе 32-разрядных МК. Однако в настоящее время, в связи со значительного роста функциональности и быстро действии 16-разрядных МК наблюдается тенденция внедрения ОСРВ в системы на основе этой элементной базы. Поэтому авторы предлогают читателю познакомится с основными идеями, на основе которого строится ОСРВ.

8.1. Рассказ: официант – «живая» операционная система реального времени

В этой главе мы представим вам понятия, связанные с операционными системами реального времени. Начнем с притчи, чтобы показать некоторые ключевые моменты и концепции ОСРВ.

Я (Стивен Ф.Баррет) испытываю глубокое и подлинное уважение к профессии официанта. Я восхищаюсь умениями, связанными с этим трудным, часто неблагодарным ремеслом. Я был непосредственным свидетелем всех сложностей этой профессии, когда был старшеклассником средней школы. По вечерам и выходным я подрабатывал в качестве уборщика/посудомойки/помощника повара в офицерском обеденном клубе на авиабазе Минот, в Северной Дакоте. Вот тогда я с трепетом наблюдал за работой официантов. Так или иначе они были способны с успехом запоминать и выполнять требования людей за множеством столиков одновременно. Порой казалось, что они способны делать все сразу. Если случалось что-то необычное, например на обеде появлялся генерал со своей свитой, или ребенок опрокидывал на стол свой стакан молока, они сходу реагировали на эти непредвиденные события. И снова, даже при этих дополнительных покушениях на свое драгоценное время, они умудрялись каким-то образом следить за всеми событиями сразу. Они должны были одновременно следить за состоянием множества заказов на различных столиках и наличием блюд на кухне (например, знать, сколько еще осталось ежедневных удешевленных обедов). Для этого они должны были постоянно держать тесную связь с поварами.

А когда случались небольшие паузы, официанты успевали готовиться к работе следующего дня, протирая серебро или сворачивая салфетки и т.д. Да, при такой интенсивной работе эти талантливые люди, должно быть, очень крепко спали по ночам.

Через несколько лет мне пришлось работать в пиццерии в Беллевю, штат Небраска, где мне была предоставлена возможность сменить мое умение в приготовлении пиццы, на работу официанта в ресторане. Я продержался официантом два дня. А потом попросил администратора, разрешить мне возвратиться к моей кулинарной деятельности. Мне было гораздо легче работать поваром, чем официантом.

Так вот, официант – это классический пример операционной системы, реального времени (ОСРВ) – системы, которая должна реагировать на множество событий,

пользуясь ограниченными ресурсами. Во встроенной системе управления, основанной на ОСРВ, мы имеем только одно устройство, последовательный процессор, которое должно обнаруживать множество событий, сортировать их по приоритетам, и реагировать на них соответствующим образом. При этом процессор не может обслуживать до полного завершения только событие с самым высоким приоритетом, игнорируя все остальные. Он должен каким-то образом реагировать на событие с самым высоким приоритетом в течение некоторого времени, а затем перейти к обработке другого события со следующим уровнем приоритета. Нетрудно представить себе, что ждет официанта, который всю свою энергию и внимание будет уделять только одному столику, игнорируя все остальные.

Процессоры, переключаясь с одной задачи на другую, должны помнить такие ключевые детали происходящих событий, как значения основных регистров, программных счетчиков, и так далее. Назовем эту информацию контекстом задачи. Точно так же, как официант переключает свое внимание «от столика к столику», так и процессоры должны помнить состояние каждого своего «столика».

Как мы уже сказали, когда происходит незапланированное событие с более высоким приоритетом, процессор (или официант) должен своевременно ответить на это новое событие, не игнорируя при этом полностью все остальные. Кроме того, должна иметься связь между отдельными задачами, которая называется межзадачной связью.

Наша цель в этой главе состоит в том, чтобы познакомить вас с понятиями и сложностями разработки ОСРВ системы. Будем считать, что вы не имеете никакой предварительной подготовки в этой области. Мы не предлагаем вам написать собственную ОСРВ , отказавшись от покупки готовой версии. Наша цель состоит в том, чтобы рассказать о понятиях и проблемах ОСРВ. А выбор между разработкой собственной системы и покупкой коммерческой версии мы оставляем за вами.

8.2. Что является целью ОСРВ?

В этом разделе мы обсуждаем основные концепции ОСРВ. Как только вы получите общее представление о том, что же такое ОСРВ, мы начнем обзор основных концепций, касающихся их разработки..

Так, что же такое операционная система реального времени? Согласно большинству фундаментальных определений, ОСРВ – это операционная система процессора, которая должна своевременно обрабатывать несколько событий при ограниченных ресурсах. Во встроенной системе управления целый ряд событий обрабатывается одним устройством – последовательным процессором. Операционная система должна обрабатывать многие, часто одновременные события, каждое из которых требует для своей для этого драгоценного времени. Так как мы используем одно устройство, последовательный процессор, он может выполнять в каждый момент только один шаг программы. Мы исследуем методы, которые позволяют так определить систему приоритетов событий, чтобы сначала были обработаны события с самым высоким приоритетом, но при этом другие события с более низким приоритетом не игнорировались.

Все события обрабатываются как бы одновременно (хотя мы должны снова подчеркнуть, что это не соответствует действительности , так как мы используем для обработки только одно устройство – последовательный процессор).

События, которыми управляет ОСРВ могут быть периодическими, асинхронными или возникающими в произвольное время. Например, если мы разрабатываеи ОСРВ для управления всей вашей домашней работой, некоторые события, такие, например, как корректировка температуры в помещениях, будут происходить периодически, в то время как другие, такие как срабатывание охранной или пожарной сигнализации, не могут быть запланированы и происходят в любое время.

Мы исследуем здесь широкий диапазон систем ОСРВ – от простых систем опроса (поллинга) до сложных, с несколькими различными прерываниями и рассмотрим также смешанные системы, которые являются комбинацией обоих типов систем. При простом опросе, операционная система регулярно проходит через ряд задач. Например, операционная система, контролируя местную защиту, могла бы последовательно циклически опрашивать состояние каждого датчика защиты в помещениях вашей фирмы. При этом система защиты как бы задает каждому датчику вопрос «А здесь все в порядке?» Система же, использующая несколько прерываний, вместо этого реагировала бы на события в тот момент, когда они случаются. Например, при активации датчика защиты, он вызывал бы тревогу в операционной системе через прерывание, показывая, что произошло некоторое важное событие, требующее реакции. Различие между двумя операционными системами заключается в том, что при поллинге, операционная система постоянно опрашивает состояния, а система, управляемая прерыванием, приводится в готовность, когда происходит некоторое ключевое событие. Важно подчеркнуть, что ни одна методика не лучше другой. Каждая имеет собственные преимущества и недостатки. В действительности важно, чтобы особенности каждой операционной системы соответствовали ее конкретному применению.

Пример: Имеется пример, иллюстрирующий, что операционная система может выполнять критическую функцию, но при этом использовать метод поллинга. Первый автор книги работал в старших классах и во время обучения в колледже в электронной фирме. Фирма разрабатывала систему защиты от хищения телевизоров для мотелей или гостиниц. Эта система должна была опрашивать телевизоры в каждом номере, чтобы установить их наличие. Система непрерывно и последовательно опрашивала все телевизоры в гостинице или мотеле. Если телевизор не отвечал, в главном офисе гостиницы звучал сигнал тревоги и высвечивался номер помещения на семисегментном дисплее. И снова, продолжался опрос, поскольку все действия имели равный приоритет. Это была простая методика, которую к тому же довольно легко было реализовать.

А теперь вернемся к теории ОСРВ. Любое действие операционной системы в ОСРВ называется *задачей*. Так как система предназначена для обработки целого ряда задач, она называется *многозадачной системой*. Считается, что все задачи выполняются одновременно. То есть отдельная задача выполняется до определенной точки перехода внутри задачи или до определенного момента, связанного с функциями задачи. Затем операционная система передает управление другой задаче, ожидающей выполнения. Управление возвращается первой задаче спустя некоторое время, и она продолжает выполнять соответствующие действия с того места, где была прервана. Так как операционная система передает управление от задачи к задаче, важно, чтобы состояние всех ключевых регистров, связанных с задачей сохранялось во время выполнения других задач. Ключевые значения регистров называются *контекстом задачи*. Такая работа ничем не отличается от деятельности официанта в приведенном ранее рассказе. Когда официант переключает внимание с одного столика на другой, ему, или ей необходимо запомнить состояние предыдущего столика – его контекст.

Операционные системы реального времени могут быть классифицированы по степени риска, связанного с невыполнением задачи за установленное ограниченное время обработки. Если система не завершает назначенную задачу за ограниченное время, то происходит сбой в работе ОСРВ. Даже удачное завершение задачи с запаздыванием должно считаться сбоем. Например, если мы рассматриваем ОСРВ, корректирующую направление полета ракеты, то будет считаться сбоем, если вычисления не будут закончены достаточно быстро, чтобы воздействовать на ход ракеты. По критичности ограничений, налагаемых на время отклика, системы ОСРВ можно разделить на следующие категории [8]:

- Жесткая система реального времени: система, где выход момента окочания вычислений за установленные временные рамки ведет к сбою системы;
- Твердая система реального времени: система, в которой небольшой выход за установленный предельный интервала может допускаться;
- Мягкая система реального времени: система, эффективность которой ухудшается при выходе за пределы временных ограничений, но система продолжает функционировать.

ОСРВ выполняет все действия, связанные с задачей, включая следующее:

- Управление задачей, включая планирование и диспетчеризацию;
- Связь между задачами (межзадачная связь);
- Управление системой памяти;
- Управление системой ввода – вывода (I/O);
- Синхронизация;
- Управление обработкой ошибок;
- Управление сообщениями.

При рассмотрении встроенных систем ОСРВ, мы обычно касаемся только небольшой части операционной системы, которая называется ядром и обеспечивает ключевые функции планирования задачи, диспетчеризации и межзадачной связи. Далее в настоящей главе мы рассмотрим эти понятия.

Вопросы для самопроверки

Вопрос: Что такое задача?

Ответ: Каждое действие ОСРВ определено как задача.

Вопрос: Что называется контекстом?

Ответ: Текущее значение ключевых регистров, связанных с задачей.

Вопрос: Что называется межзадачной связью?

Ответ: Связь между различными задачами.

Вопрос: Каково различие между жесткой, твердой и мягкой ОСРВ?

Ответ: В жесткой системе, выход времени выполнения задачи за установленный предел ведет к сбою системы; в твердой системе может допускаться незначительный выход за установленные временные рамки; в мягкой системе, выход за установленные временные рамки снижает эффективность системы, но приводит к сбоям.

Вопрос: Что такое ядро?

Ответ: Ядро – очень небольшая часть операционной системы, которая обеспечивает ключевые функции планирования задачи, диспетчеризации, и межзадачной связи.

Прежде чем перейтик дальнейшему изучению ОСРВ, мы должны сделать дать некоторых понятия, касающиеся ОСРВ.

8.3. Обзор концепций

В этом разделе мы проведем обзор некоторых важных тем, касающихся ОСРВ. Некоторые авторы определенно (и безапеляционно) заявляют, что программы ОСРВ должны быть написаны на ассемблере, чтобы обеспечить максимальное быстродействие системы. Не оспаривая такой точки зрения, мы тем не менее, представляем наши примеры на языке С, чтобы яснее осветить понятия и важные детали ОСРВ. Рассмотрим связь некоторых понятий языка С с основными принципами структур данных. ОСРВ состоит из этих совместно работающих основных структур данных, позволяющих выполнить требования к системе.

Мы советуем вам возвратиться к предыдущим разделам и вспомнить о следующих понятиях: об указателях (глава 3), глобальных и локальных переменных (глава 3) и свойствах стеков и в прерываний 68HC12 (глава 4). Повторив эти разделы, вы можете вернуться к рассмотрению динамического распределения памяти.

8.3.1. Требования к динамическому распределению RAM

В главе 4 мы обсуждали систему памяти, встроенную в отладочную плату (EVB) контроллера B32, который предназначен прежде всего для однокристальных применений. Он включает в себя 32 Кб ПЗУ типа FLASH, 1 Кб статической оперативной памяти RAM и 768 байт стираемого по байту ПЗУ типа EEPROM для сохранения данных системы. Карта памяти для B32 EVB показана на рис.8.1. Обратите внимание, что большая часть объема принадлежит флеш-памяти EEPROM, в то время как оперативная память (RAM) занимает только 1 Кб. Фактически же только 512 байтов, доступны для использования (от $ 0800 до $9FFF).

$0000 $01FF	Регистры ЦП
$0800 $0BFF	1 Кб RAM «на чипе» • Код/данные пользователя ($0800-$09FF) • Резервирована для D-Bug12 ($OA0O-$OBFF)
$0D00 $0FFF	768 байт EEPROM «на чипе» • Код/данные пользователя
$8000 $FFFF	FLASH EEPROM 32 Кб «на чипе» • код D-Bug12 ($8000-$F67F) • Область, доступная пользователю ($F6C0-$F6FF) • Настройка функций D-Bug12 ($F680-$F6BF) • Код запуска D-Bug12 ($F700-$F77F) • Таблица вектора прерывания ($F780-$F7FF) • Расширение загрузчика ($F800-$FBBF) • EEPROM загрузчика ($FC00-$FFBF) • Векторы сброса и прерывания ($FFC0-$FFFF)

Рис. 8.1. *Карта памяти микроконтроллера B32*

Для чего же эта RAM используется? Прежде всего она используется для локальных переменных каждой функции. Переменные, помещенные в стек, доступны только при вызове функции. При выходе из функции переменные удаляются из стека, освобождая пространство памяти. Ясно, что можно достаточно быстро исчерпать объем стека, если ваш встроенный контроллер использует рекурсивную подпрограмму (вызываемую, например, для получения ряда Фибоначчи или операции вычисления факториала) или функцию с высокой степенью вложения. То есть при обращении к функции, которая снова вызывает функцию, и т.д. В этих ситуациях активны как все функции, так и связанные с ними локальные переменные.

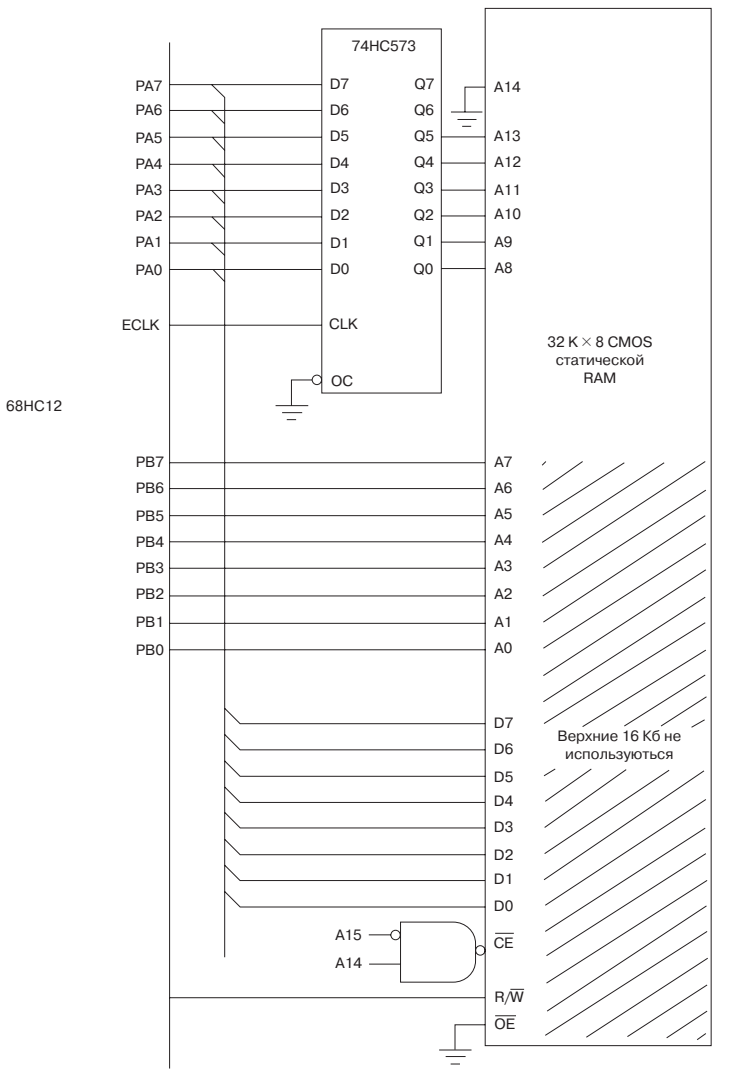


Рис. 8.2. *Подключение внешней памяти к микроконтроллеру B32*

Ключевой инструмент ОСРВ – использование таких абстрактных типов данных, как записи, списки с указателями и очереди. Обсудим их вкратце. Эти типы данных обычно используют динамическое распределение памяти RAM. Где мы можем взять RAM для этих типов данных? Если мы используем 512 байт как для абстрактных типов данных, так и для стека мы можем потенциально сталкиваться с «зависанием» структуры данных при переполнении стека или наоборот. Это может привести к катастрофическому сбою системы. Мы должны предотвратить это любой ценой. В идеале, мы должны выделить дополнительную память RAM в карте памяти B32. Было бы также полезно физически отделить эту память от памяти RAM на плате B32. Это позволило бы нам иметь отдельные пространства RAM для стека и динамической памяти. Но это не всегда возможно. Если же и стек и динамическая память постоянно находятся в одном и том же пространстве памяти, необходимо гарантировать, что не происходит подмены информации в процессе выполнения программы. Контроллер 68HC12 не обеспечивает автоматического контроля этой ситуации и исключение ее при програмировании системы входит в вашу задачу.

В разделе 4.7.1 мы уже обсуждали систему памяти, размещенной на плате B32. На рис 8.2 приведено схемное решение системы, с учетом которого мы можем использовать дополнительное пространство RAM. Настоятельно рекомендуем вам ознакомится с концепций расширения памяти у Pack и Barrett [2002, Ch. 8]. Здесь же мы только приводим схемное решение, не обсуждая проблем подключения памяти, электрической связи с помощью интерфейса и синхронизации. Хотя все они чрезвычайно важны, но при обсуждении динамического распределения памяти, без их рассмотрения можно обойтись.

Прежде чем определить конкретные границы этого дополнительного пространства RAM, обеспечим их удобный расчет. Мы хотим определить область 16 Кб для статической оперативной памяти (SRAM) в карте памяти B32 EVB. Разместим эту память в ячейках от $ 4000 до $7FFF. Если бы это удалось, то все, к чему мы имели доступ на участке памяти – это 32 Кб x 8 SRAM. Если мы заземлим старшую линию адреса этой памяти A[14], то старшие 16 Кб чипа памяти будут недоступны. Как мы уже упоминали в главе 4, PORTA обеспечивает мультиплексную линию данных D[7:0], и старшие разряды адреса A[15:8] в расширенных режимах работы МК. PORTB обеспечивает младшие разряды адреса A[7:0]. Мы используем адресную линию A[15:14] со схемой И-НЕ, чтобы создать активный низкий сигнал для SRAM памяти чипа.

Альтернативой расширению памяти является использование варианта HCS12 с большей дополнительной областью RAM. Обратитесь к различным доступным вариантам, описанным в разделах 1.3, 1.4 и 4.8. Эти варианты включают в себя RAM объемом от 4 до 12 Кб.

Перед продолжением нашего обсуждения, относящегося к динамическому распределению памяти, рассмотрим еще несколько проблем, воспользовавшись рис. 8.2.

Вопросы для самопроверки

Вопрос: Какова цель применения схемы И-НЕ?

Ответ: Логический элемент И-НЕ переходит на низкий уровень и генерирует сигнал разрешения (CE) для SRAM памяти, когда A15 находится на низком , а A14 – на высоком уровне.

Вопрос: Какова цель применения микросхемы 74HC573?

Ответ: Эта микросхема действует как защелка при демультиплексировании линий адреса / данных от порта PA.

Вопрос: Каков промежуток адресов памяти RAM?
Ответ: От $4000 до $7FFF или 16 Кб RAM.
Вопрос: Каков размер SRAM памяти?
Ответ: Этот размер составляет 2^{15} или 32К адресов с размещением одного байта в каждом адресе. Нам удается использовать только нижние 16 Кб памяти.
Вопрос: Какой вид будет иметь карта памяти после введения этих новых компонентов памяти.
Ответ: Кара памяти приведена на рис. 8.3.

8.3.2. Динамическое распределение памяти

В компиляторе C для эффективного управления динамической памятью используются указатели. Они позволяют объявить большое число переменных различных типов и размеров. Когда память динамически распределена для переменных в RAM, указатель может привести вас к ресурсам памяти для переменной.

Динамическое распределение памяти осуществляется с помощью команды распределения памяти `malloc()`. Эта команда содержится в файле заголовка stdlib.h, который является частью любого C компилятора. Команда `malloc()` обычно используется вместе с функцией `sizeof()`. Эта комбинация функций чрезвычайно полезна при динамическом распределении памяти. Общая форма этой комбинации функций:

```
Ptr = (variable_type) * malloc (sizeof (variable_type));
```

$0000 $01FF	Регистры ЦП
$0800 $0BFF	1 Кб RAM «на чипе» • Код/данные пользователя ($0800-$09FF) • Резервирована для D-Bug12 ($OA0O-$OBFF)
$0D00 $0FFF	768 байт EEPROM «на чипе» • Код/данные пользователя
$8000 $FFFF	FLASH EEPROM 32 Кб «на чипе» • код D-Bug12 ($8000-$F67F) • Область, доступная пользователю ($F6C0-$F6FF) • Настройка функций D-Bug12 ($F680-$F6BF) • Код запуска D-Bug12 ($F700-$F77F) • Таблица вектора прерывания ($F780-$F7FF) • Расширение загрузчика ($F800-$FBFF) • EEPROM загрузчика ($FC00-$FFBF) • Векторы сброса и прерывания ($FFC0-$FFFF)

Рис. 8.3. *Карта памяти B32 расширенная внешней флеш-памятью и RAM*

Большинство структур данных объявляется и распределяется с помощью этой методики. Когда переменная больше не нужнаа, пространство памяти использовуемое для нее, возвращается системе с помощью функции `free()`. Это наилучший способ динамического распределения памяти. Мы создаем переменные на ходу (в процессе выполнения программы), когда мы нуждаемся в них, и избавляемся от переменных, когда они больше не нужны.

При таком распределении и освобождении памяти, происходящем в процессе выполнения программы, необходима эффективная система управления памятью. Динамическая память – это часть памяти RAM, используемая для динамического распределения памяти. Как мы упомянули прежде, важно сохранять стек и динамическую память, отделенными от друг друга.

Теперь, когда мы имеем два отдельных пространства RAM в нашей карте памяти, мы можем легко выделить одну из них для стека, а другую – для динамической памяти. Мы рекомендуем использование меньшего пространства для стека и большего для динамической памяти. В данном компиляторе имеются параметры настройки, конфигурируемые пользователем и позволяющие установить адрес начала для стека и области динамической памяти.

Получив способ динамического распределения памяти, мы можем теперь создавать структуры данных для использования в наших ОСРВ. В следующем разделе мы проведем общий обзор структур данных, используемых в ОСРВ: структур (или записей), списков с указателями, очередей, и круговых очередей. Как только мы познакомимся с каждым из этих основных типов, мы сможем объединять их в более сложные структуры данных, используемые для работы ОСРВ.

8.3.3. Структуры данных

В этом разделе мы проведем общий обзор структур данных, используемых в операционных системах, работающих в режиме реального времени. Мы выполним обзор каждой структуры данных отдельно и затем объединим их, чтобы выполнять различные операции внутри ОСРВ.

Структура. Мы будем использовать взаимозаменяемые термины *запись* и *структура*. Структуры позволяют программистам при разработке совокупности данных для спецификации, использовать другие основные типы данных. Это позво-

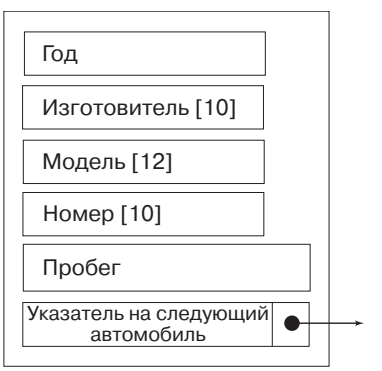


Рис. 8.4. *Запись для автомобиля. Запись содержит поля данных, совокупность которых описывает автомобиль*

ляет им следить за сложной информацией, которая может содержать различные типы данных. Например, если вы разрабатываете систему описи для автомобильного торгового агента, вы могли бы разработать следующую структуру основных данных для конкретного автомобиля, таких, как год выпуска, изготовитель, модель, номер идентификации транспортного средства (VIN), и прогон, как показано на рис.8.4.

Каждый из типов данных в записи называется полем. Мы объявляем запись или структуру, используя следующий синтаксис:

```
struct car
{
int  year;                    /* год производства* /
char make[10];               /*BWM, Hummer, Saturn                  */
char model[12];              /*купе, обратимый, SUV, пикап          */
char VTN[10];                /* комбинация цифр, букв               */
float mileage;               /*показания счеитчика: от 0 до 500,000+*/
struct car *next;    /*указатель на следующий автомобиль в списке*/
};
/*typedef обеспечивает компилятор an alternate???*/
typedef    struct car  ELEMENT;    /*для типа переменной */
typedef    ELEMENT  *car_temp_ptr; /*определяет указатель на автомобиль*/
```

Как можно видеть, все поля записи описывают различные параметры данного автомобиля. Мы привели сокращенный список параметров. Структура может содержать столько полей, сколько необходимо. Мы выбрали для каждого поля тот тип данных, который наиболе подходит для описания каждого параметра автомобиля.

Структуры создаются с использованием динамических методов распределения памяти. Как мы уже упоминали, команда `malloc()` – предоставляет методику динамического распределения памяти, определяя соответствующий объем памяти для структуры данных. Например, чтобы создать новую запись для автомобиля, можно использовать следующий код:

```
Car_temp_jptr new_car_entry;
New_car_entry = (car_temp_ptr) malloc(sizeof(ELEMENT));
```

В главе 3, мы описали, как обращаться к различным полям внутри записи. В этом примере, мы инициализируем недавно созданную автомобильную запись с информацией о кокретном автомобиле. Обратите внимание, что мы использовали оператор –>, чтобы обратиться к специфическому полю внутри структуры.

```
/* инициализация новых полей для структуры car */
new_car_entry->year = 1981; /* год производства                    */
strcpy(new_car_entry->make,"Chevy");   /*BWM, Hummer, Saturn */
strcpy (new__car_entry->model, "Camaro"); /*купе, обратимый, SUV, пикап */
strcpy(new_car_entry->VIN, " 12Z3 67"); /* комбинация цифр, букв*/
new_car_entry->mileage = 37456; /*показания спидометра: от 0 до 500,000+ */
new_car_entry->next - NULL;    /* определяет указатель на автомобиль */
```

Чтобы распечатать различные поля записи, мы могли бы использовать следующий код:

```
printf (" \nyear: %4d ", new_car __ вход -> year); / *year mfg */
printf (" \nmake: %s ", new_car_entry -> make); /*car делает */
```

```
printf (" \nmodel: %s ", new_car __ entry -> mobel >; /*model*/
printf (" \nVIN: является ", new_car_entry -> VIN); /*VIN */
printf (" \nMileage: %6.of ", new_car_entry -> mileage);
/*odometer reading*/
```

Поместим все эти части вместе с примером. Мы убедительно просим вас компилировать и выполнить этот код.

```
#include <stdio.h>          /*стандартные входные/выходные функции*/
#include <stdlib.h>         /*библиотека и распределение памяти */
void main(void)
/*определение структуры */
struct car
int  year; /*год выпуска */
char make[10]; /*BWM, Hummer, Saturn*/
char model[12]; /*купе, обратимый, SUV, пикап */
char VIN[10]; /*комбинация цифр и букв*/
float mileage; /*показания одометра: от 0 до  500 000+   */
struct car *next; /*указатель на следующий автомобиль в списке */
typedef   struct car  ELEMENT; typedef   ELEMENT  *car_temp_ptr;
car_temp__ptr new__car_entry; /*ввод записи для автомобиля*/
new_car_entry = (car_temp_ptr) malloc(sizeof(ELEMENT));
/*инициализация новых полей для автомобиля  */
new_car_entry-?year = 1981; /*год изготовления*/
strcpy(new_car__entry->make, "Chevy");   /*BWM, Hummer, Saturn    */
strcpy(new_car_entry->model,"Camaro");
/* купе, обратимый, SUV, пикап */
strcpy(new_car_entry->VIN, "12Z3 67");
/* комбинация цифр и букв */
new_car_entry->mileage - 37456;/*показания одометра: 0 to 500,000+*/
new_car_entry->next - NULL;/*указатель на следующий автомобиль в списке*/

printf(*\nyear: %4d", new_car_entry->year);   /*год выпуска */
printf (" \nmake: %s", new__car_entry->make> ;  /*производитель */
Printf (" \model: % s ", new_car_entry ->model); /*модель */
Printf (" \nVIN: % s ", new_car_entry ->VTN); /*номер */
Printf (" \nMileage: % 6. Из ', new_car_entry - > rnileage];
/*показания одометра*/
}
```

Когда эта программа будет откомпилирована и выполнена, на экране вашего ПК появится следующее сообщение:

```
year: 1981
make: Chevy
model: Camaro
VIN: 12Z367
Mileage: 37456
```

Пока, наверное, использование динамического распределения памяти не кажется вам очень мощным инструментом. В следующем разделе вы сможете оценить силу динамического распределения памяти, когда мы скомпонуем записи и создадим список, что позволит нам осуществлять изменения в процессе выполнения программы. То есть мы сможем добавлять новые элементы в список, переносить элементы из од-

ного списка в другой, удалять элементы из списка, и т.д. Но не будем здесь углубляться в лес деталей, чтобы сразу в нем не заблудиться. Лучше сосредоточим свое внимание на дороге через этот лес. Не будем забывать, что мы изучаем здесь основные структуры данных, чтобы узнать, как они могут использоваться в построении и реализации операционной системы в режиме реального времени.

Список с указателями. Список с указателями является мощной структурой данных, которая может быть создана, вставлена в программу или удалена из нее динамически в процессе выполнения программы. Список связей состоит из узлов, содержащих две части: часть данных и часть поля связи. Часть данных хранится информация об узле (или пункте) списка. Например, если мы должны были создать список автомобилей, готовых к продаже, частью данных для узла будет структура (или запись) car, которую мы разработали в предыдущем разделе. Полем связи был бы указатель (адрес памяти) следующую запись в списке. Начало списка названо *головой* (head). Конец списка называется *хвостом* (tail) и обозначается символом нуля (?) в поле связи. Это построение списка иллюстрируется на рис.8.5. Здесь приведено объявление различных списков, позволяющих автомобильным дилерам проследить за состоянием парка автомобилей.

```
car_temp_ptr head_ptr;       /*начало списка состояния автомобилей    */
car_temp_ptr in_stock_list;  / *автомобили в наличии                  */
car_temp_ptr repair_list;
/*автомобили в ремоньных мастерских - не предназначены для продажи  */
car_temp_ptr paint_shop_list;
/*автомобили в мастерских покраски - - не предназначены для продажи */
car_temp_ptr sold_list;
/*проданные автомобили - не предназначены для продажи */
```

Мир автомобильных продаж очень динамичен. Автомобильные дилеры постоянно продают автомобили, занимаются их обменом, размещают автомобили в мастерских для ремонта, или даже производят перекрашивание автомобилей. Если нам необходимо создать список для каждого из этих действий, мы будем постоянно добавлять и удалять элементы в каждом из списков. Эти действия иллюстрируются на рис.8.5в и 8.5г.

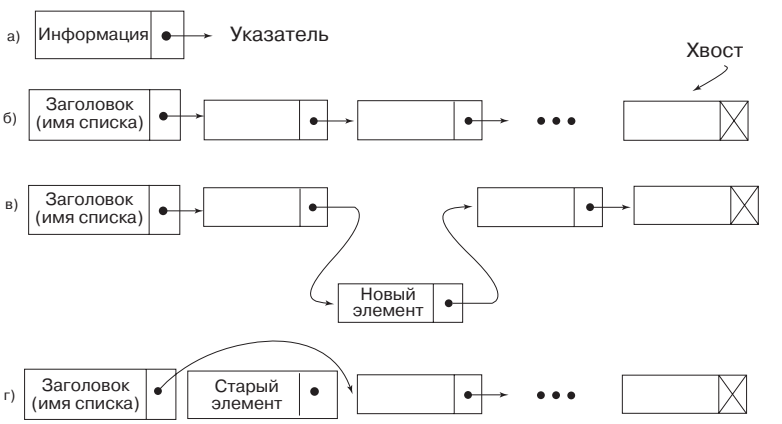


Рис. 8.5. *Операции со списком а) список состоит из узла с данными и указателя, б) начало списка называется головой, конец списка – хвостом, в) вставка узла элемента в список, г) удаление элемента*

Чтобы вставить новый автомобиль в список, мы были бы должны найти соответствующее место для автомобиля в существующем списке. Например, если мы размещаем автомобили в алфавитном порядок, мы должны просматривать список, пока мы не найдем, соответствующее место чтобы вставить автомобиль в список, как показано на рис. 8.6. Как только соответствующая точка ввода найдена, указатель предшествующего узла должен быть изменен, чтобы указать на новый автомобиль, и указатель нового автомобиля должен быть изменен, чтобы указать на следующий автомобиль в списке.

Когда автомобиль продан, он больше не доступен для продажи. Он должно быть затем быть удален из списка «для продажи». Для этого связь предшествующего автомобиля должна теперь указывать на последующий автомобиль. Автомобиль, который был продан, будет теперь действительно исключен из списка «для продажи» и может быть добавлен в список «проданные». Если бы у нас не было списка «проданные», мы могли бы освободить динамическую память,выделенную для этого автомобиля. Это осуществляется с помощью комады `free (argument)`.

Если вы ищите в списке определенный автомобиль, вы должны будете следовать по цепочке указателей, пока желательный автомобиль не будет найден. Мы выполнили бы поиск, исследуя содержание определенных полей каждой записи в списке с указателями.

На рис. 8.7 показаны общие функции, связанные со списком с указателями. Код для каждой из этих функций приведен ниже:

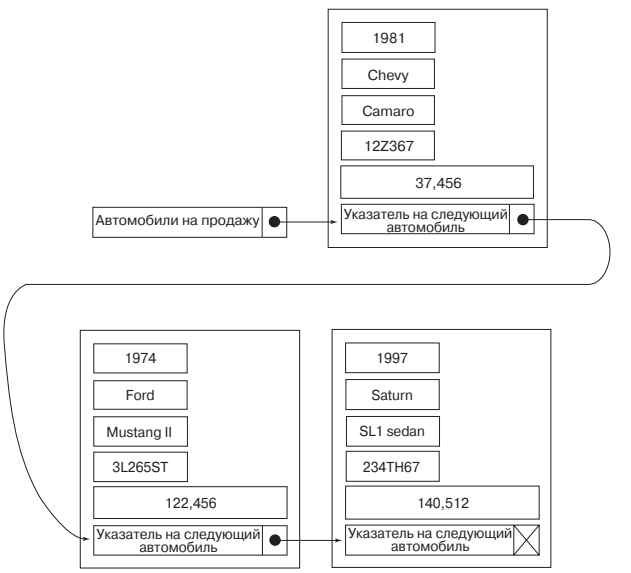


Рис. 8.6. *Список с указателями для текущего учета автомобилей*

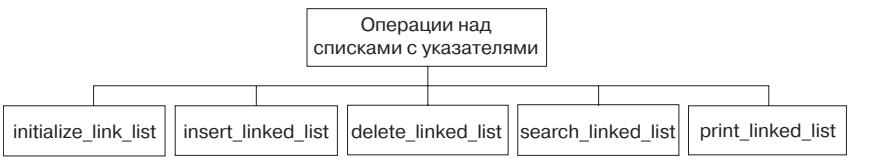


Рис. 8.7. *Общие функции, связанные со списками с указателями*

```
/*********************************************************************
/*имя файла: linklist.c */
/*********************************************************************

/*включенные файлы*/
*include <stdio.h>  /*стандартная библиотека Вводов - вывода */
*include <stdlib.h> /*стандартная библиотека  динамического */
                    /*распределения*/

/*global variables - глобальные переменные. Объявление этих переменных */
/*помещают в файл заголовка (header file). Они приведены здесь,       */
/* чтобы иллюстрировать последовательность построения  программы.     */


/*определение структуры "автомобиль"*/
struct car
{
int  year;                  /* год производства                      */
char make[10];              /*BWM, Hummer, Saturn                    */
char model[12];             /*купе, обратимый, SUV, пикап            */
char VTN[10];               /*комбинация цифр, букв                  */
float mileage;              /*показания спидометра: от 0 до 500,000+*/
struct car *next;    /*указатель на следующий автомобиль в списке   */
};

/*определение  указателя на автомобиль */
typedef   struct car  ELEMENT;
typedef   ELEMENT  *car_temp_ptr;

/*функции прототипов*/
void initialize_link_list(void);
void print_link_list (car__temp_ptr);
void insert_link_list(car_temp_ptr);
void delete_link_list(car_temp_ptr);
void search_link_list(car_temp_ptr);

/*переменные*/
/ " Создают списки, чтобы следить за состоянием автомобильного сервиса*/

Car_temp_ptr in_stock_list;  /* автомобили в продаже                 */
Car_temp_ptr repair_list; /* автомобили в ремонт - не подлежат продаже*/
Car_temp_ptr paint_shop_list;/*автомобили в покраске - не подлежат продаже*/
Car_temp_ptr sold_list;    /*проданные автомобили -- не подлежат продаже*/
Car_temp_ptr new_car_entry; /*новый автомобиль для введения в список*/
int TRUE=l,PALSE=0;        /*логические флаги                       */

void main(void)
{

                           /*заполняет пустой список переменными NULL */
in_stock_list = NULL;      /* автомобили в продаже                   */
repair_list = NULL;  /* автомобили в ремонте - не подлежат продаже    */
paint__shop__list = NULL; /* автомобили в покраске - не подлежат продаже*/
sold_list - NULL;    /*проданные автомобили -- не подлежат продаже  */
```

```
new_car_entry = NULL;

initialize_link_list();      /*составление списка для продажи       */
print_link_list(in_stock_list); /*print the list                    */
insert_link_list (in_stock__list); /*вставить новый автомобиль в список*/
print_link_list(in_stock_list); /*распечатать список                */
delete_link_list(in_stock_list); /*удалить автомобиль из списка     */


print_link_list(in_stock_list); /*распечатать список                 */
search_link_list(in_stock_list); /*поиск определенного пункта в списке */

/*********************************************************************/
/*void initialize_link_list (car__temp_ptr): инициализирует автомобиль */
/* для списка продаж используя список.Отметим, что этот список       */
/* с указателями был объявлен как глобальная переменная.             */
/*********************************************************************

void initialize_link_list(void)
{
car_ternp_ptr new_car_entryl, new_car_entry2;
/*создает вход в список автомобилей */

new_car_entry = (car_temp_ptr) malloc(sizeof(ELEMENT));
/*инициализирует новые поля для ввода автомобиля в список*/
new_car_entry->year - 1981;            /*год выпуска */
strcpy(new_car__entry->make, "Chevy"); /*BWM, Hummer, Saturn        */
strcpy(new_car_entry->model,"Camaro"); /*купе, обратимый, SUV, пикап*/
strcpy(new_car_entry->VIN, "12Z3 67"); /*комбинация цифр и букв     */
new_car_entry->mileage - 37456; /*показания одометра: от 0 до 500 000+*/
new_car_entry->next - NULL; /*указатель на следующий автомобиль в списке*/

in_stock_list - new_car_entry;

new_car_entryl = (car_temp_ptr) malloc(sizeof(ELEMENT));
/*инициализирует новые поля для ввода автомобиля в список*/
new_car_entryl->year = 1974;           /*год выпуска*/
strcpy(new_car_entryl->make,"Ford");   /*BWM, Hummer, Saturn        */
strcpy(new_car_entryl->model,"Mustangll")/*купе, обратимый, SUV, пикап*/
strcpy (new_car_entryl->VIN, "3L265ST" ) ; /*комбинация цифр и букв */
new_car_entryl->mileage = 122456; /*показания одометра: от 0 до 500 000+ */
new_car_entryl->next = NULL; /*указатель на следующий автомобиль в списке */

new_car_entry2 = (car__temp_ptr) malloc(sizeof (ELEMENT));
/*инициализирует новые поля для ввода автомобиля в список*/
new_car__entry2->year - 1997;          /*год выпуска*/
strcpy(new_car_entry2->make,"Saturn"); /*BWM, Hummer, Saturn        */
strcpy(new_car_entry2->model,"SL1");   /*купе, обратимый, SUV, пикап */
strcpy(new_car_entry2->VIN, "234TH67");/*комбинация цифр и букв     */
new_car_entry2->mileage = 140512;/*показания одометра: от 0 до  500 000+ */
new_car_entry2->next - NULL; /*указатель на следующий автомобиль в списке*/
new_car_entryl->next = new_car_entry2;
}

/*********************************************************************/
```

```
/*print_link_list: печатает поля выделенного списка с указателями     */
/*********************************************************************/

void print_link_list(car_temp_ptr print_list)
{
car_temp_ptr temp__ptr;      /*объявляет текущий указатель           */

printf("\nCars available in stock for sale:");

/*продвижение по списку  */
for (ternp_ptr-print_list; temp_ptr ! =NULL; temp_ptr-temp_ptr->next){
  printff"\n\nyear: %4d, temp_ptr->year);      /*год выпуска*/
  printf("\nmake: %s", temp_ptr->make);        /*изготовитель*/
  printf("\nmodel: %s", temp_ptr->model);      /*модель*/
  printf("\nVIN: %S", temp_ptr->VIN);  /*номер*/
  printf("\nMileage: %6.Of", temp_ptr->mileage); /*показания одометра*/
  }
}

/*********************************************************************/
/*insert_link_list (in_stock_list) - вставляют новый автомобиль в     */
/* отмеченный список в алфавитном порядке                             */
/*********************************************************************/

void insert_link_list(car_temp_ptr in_stock_list)
{
car_temp_ptr new_car_entry, list, ptr;
int place_found;

list = in_stock_list;
                                        /*создает ввод автомобиля    */
new_car_entry - (car_temp_ptr) malloc(sizeof(ELEMENT));
              /*инициализирует новые поля для ввода автомобиля в спи-
сок */
new_car_entry->year - 2002;             /*год выпуска */


 strcpy(new_car_entry->make,"Hummer"); /*BWM, Hummer, Saturn*/
strcpy(new_car_entry->model/"H2");       /*купе, обратимый, SUV, пикап */
strcpy (new_car_entry->VTIM, "73H2L7");/*комбинация цифр и букв*/
new_car_entry->mileage = 13; /*показания одометра: от 0 до  500 000+
*/
new_car__entry->next = NULL; /*указатель на следующий автомобиль в списке */

if (list==NULL)
{                                       /*вставка в пустой список    */
list=new_car_entry;
)
else
{                            /* вставка в первый элемент списка */
if(strcmp(new_car_entry->make, list->make) <1)
{
new_car_entry->next=list; list - new_car_entry;
}
else                 /*вставка в непустой список                     */
```

```
{
ptr = list;          /*определение позиции вставки                    */

place__found = FALSE;
while((ptr->next != NULL) && (!place_found))
{
if ( strcmp (new_jcar_entry->make, ptr->next->make) > = 1)
                                        /*сравнение                    */
{
ptr=ptr->next;                          /*продвижение по списку        */

}
else                                    /*вставка после указателя      */
{
place_found = TRUE;
}
}/*конец цикла while*/
                              /*переадресует уазатель, чтобы           */
                              /*закончить ввод в список                */
new_car_entry->next = ptr->next;
ptr->next - new_car_entry;
}/*конец else*/
}/*конец else*/
}/*конец insert_link_list*/
/*********************************************************************/
/*delete_link_list (car_ternp_ptr): */удаление отмеченных элементов   */
/*из списка                                                           */
/*********************************************************************/

void delete_link_list(car_temp_ptr in_stock_list)
{
car_temp_ptr current,backup,temp;        /*текущий указатель списка */
char delete_make[10];
/*определить поле  make для удаления */
printf("\n\nDelete car from for sale list.");
printf("\nEnter make of car for deletion from list.");
scanf{"%s", delete_make);
                              /*инициировать указатели для поиска    */
current = in_stock_list;
backup=NULL;
               /*поиск записи, содержащих заданное значение make     */
while(strcmp(current->make, delete_make) !=0)(
  backup - current;
  current = current -> next;
  }
                     /*Был удален автомобиль из первого элемента?    */
if fbackup=-NULL){   /*удалить автомобиль из первого элемента        */
in_stock_list - in_stock_list->next;
}
else{                                  /*удалить элемент из списка  */
backup->next - current -> next;
}
free(current);       /*перераспределить динамическую память*/
}
/*********************************************************************/
```

```
/*********************************************************************/
/*void search_link_list (car__ternp_ptr) - найти запись с определенным  */
/* значением поля make.  Распчатать автомобили этого изготовителя.   */
/*********************************************************************/

void search_link_list(car_temp_ptr search_list)
{
char searcn_make [10] ;
car_temp_ptr temp_ptr;       /*объявить текущий указатель            */
                             /*определить изготовителя для поиска    */
printf("\n\nSearch for car in stock.");
printf("\nEnter make of car to search for in list. ");
scanf("%s", search_make);
                                /*движение по списку                 */
for(temp_ptr-search_list;temp_ptr!-NULL;temp_ptr-temp_ptr->next)
  {
  if(strcmp(temp_ptr->make, search_make) == 0)
     {
     printf("\n\nyear: %4d", temp_ptr->year);  /*год изготовления   */
     printf("\nraake: %s",  temp_ptr->make);   /*изготовитель       */



    printf("\nmodel: %s", temp_ptr->model);   /*модель */
    printf("\nVIN: %s",   temp_ptr->VIN);     /*номер */
    printf (" \nMileage : %6 . Of ", temp_ptr->rnileage);
    /*считать показания одометра*/
    }
  }
}
/*********************************************************************/
/*********************************************************************/
```

После выполнения программы на дисплее появится следующее сообщение.

```
year: 1981
make: Chevy
model: Camaro
VIN: 12Z367
mileage: 37456

year: 1974
make: Ford
model: Mustangll
VIN: 3L265ST
mileage: 122456

year: 1997
make: Saturn
model: SL1
VIN: 234TH67
Mileage: 140512

year: 1981
```

```
make: Chevy
model: Camaro
VIN: 12Z367
mileage: 37456

year: 1974
make:Ford
model: Mustangll
VIN: 3L265ST
mileage: 122 456

year: 2002
make: Hummer
model: H2
VIN: 73H2L7
Mileage: 13

year: 1997
make: Saturn
model: SL1
VIN: 234TH67
Mileage: 140512

Удалены следующие автомобили из списка для продаж.
Введено поле  make для удаления из списка - Hummer

year: 1981
make: Chevy
model: Camaro
VIN: 12Z367
mileage: 37456

year: 1974 make:
Ford model: Mustangll
VIN: 3L265ST
mileage: 122456

year: 1997
 make: Saturn
model: SL1
VIN: 234TH67
Mileage: 140512

Найден автомобиль в списке продаж.
Введенное поле  make для поиска - Saturn

year: 1997
make: Saturn
model: SL1
VIN: 234TH67
Mileage: 140512
```

Этот очень простой пример был придуман, чтобы показать основные действия, связанные обработкой списка с указателями. Мы намеренно выбрали скелетную программу, чтобы концентрироваться на работе списков с указателями. Если бы мы разра-

Рис. 8.8. *Структура данных очередь – «первым вошел-первым вышел»*

батывали фактическую программу описи, основанную на списках с указателями, мы разработали бы дружественное меню, которое позволило бы вызывать функции в любом порядке. Кроме того, мы сформировали бы исходный список загрузив данные из файла. Мы обеспечили бы также возможность сохранять данные списка в файле.

Очередь. Очередь представляет собой особым образом сконфигурированный список с указателями. Она называется также буфером первым-вошел-первым-вышел (first-infirst out – FIFO). Элементы добавляются в хвост очереди и извлекаются с головы, как показано на рис.8.8. Вернемся к нашей автомобильной коммерческой аналогии. Как только вы приобретаете автомобиль, вы должны зарегистрировать его в департаменте транспортных средств. Я был там недавно с моим сыном, которому нужны было заменить водительские права. После нашего прихода мы встали в хвост очереди и стали ждать, пока нас обслужат. После короткого ожидания, мы достигли головы очереди и смогли получить требуемые права. Нас обслуживали в порядке прохождения очереди. Во время пика занятости, например, в утренние часы, в конце месяца – очередь в департменте может быть очень длинной. В более свободное время она может быть совсем короткой. В обоих случаях мы видим, что очередь не имеет фиксированного числа элементов. Длина очереди или число элементов в ней, является переменной величиной и изменяется, в зависимости от действия программы.

Круговая очередь. В круговой очереди, которая имеет ту же самую базисную структуру, что и простая очередь, указатель последней записи в очереди не пустой, он указывает на первую запись. Круговая очередь показана на рис.8.9. Как мы скоро увидим, это эффективная структура данных для переключения с задачи на задачу.

Стек. Стек – это структура данных последним вошел – первым вышел (last-in-first-out – LIFO), показанная на рис.8.10. Она также может быть создана с помощью методов списка с указателями. Во встроенных контроллерных системах на базе 68HC12, стек – определенная пользователем часть RAM, в которой в течение нормального выполнения программы временно хранятся переменные, например содержимое какого-либо регистра. Верхняя часть стека в 68HC12 обычно определяется последней позицией RAM плюс один. Указатель вершины стека для 68HC12 содержит адрес последнего используемого расположения стека. Когда элемент помещают в стек, указатель вершины стека уменьшается на 1 (проводится операция декремента). Когда элемент извлекается из стека, указатель вершины стека увеличивается на 1 (операция инкремента). Когда программирование проводится на языке C, положение вершины стека является опцией компилятора. Если встроенная контроллерная система содержитт только один стек, лучше всего просто использовать свойства, встроенные в процессор. Однако, как мы скоро увидим, в ОСРВ можем потребоваться несколько стеков, по одному для каждой задачи. В этом случае разработчику системы необходимо, чтобы обеспечить работу нескольких стеков. Если используются динамические методы распределения памяти, то несколько стеков создаются и используются в динамической памяти. Мы оставим разработку структуры данных стека, использующей динами-

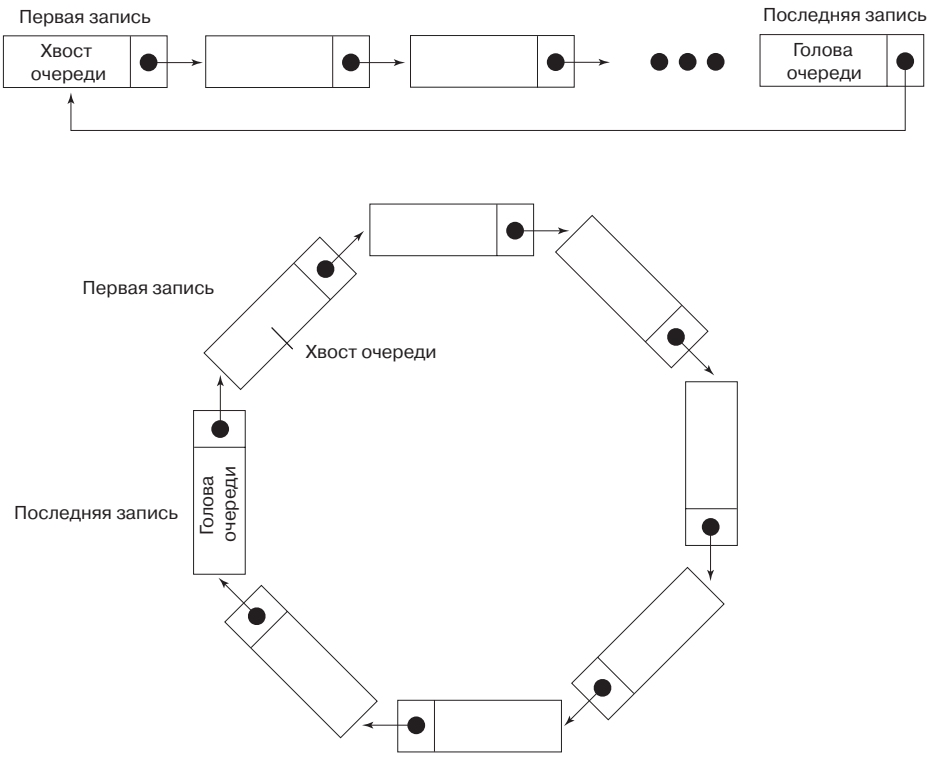


Рис. 8.9. *Круговая очередь*

Рис. 8.10. *Стек*

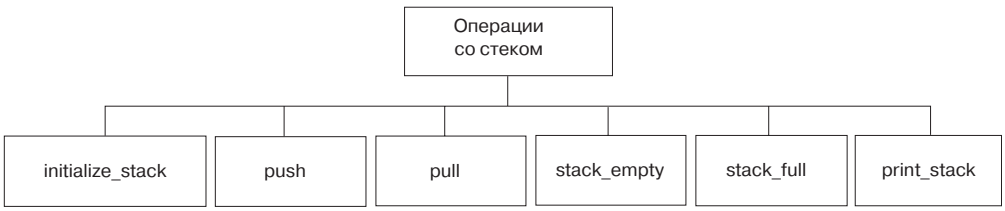


Рис. 8.11. *Операции со стеком*

ческие методы распределения памяти в качестве вашей домашней работы (задача 1 для самостоятельного исследования). Вместо этого, мы разработаем структуру стека с фиксированный размером массива. Стек с фиксированным массивом используется, когда размер стека известен и неизменен. Как мы скоро увидим, стеки с успехом могут используются в блоках управления задачами.

Структура данных стека, разработаны ли они с использованием методов динамическиого распределения или фиксированного массива, имеет следующие функции:

- `initialize_stack`: инициализация стека;
- `push`: помесить элемент в стек;
- `pull`: извлечь элемент из стека;
- `stack_empty`: выяснить, не пуст ли стек. Это необходимо сделать до использования функции `Pull`;
- `stack_full`: выяснить, не полон ли стек. Это необходимо сделать до использования функции `Push`;
- `Print_stack`: распечатать содержимое стека.

Основные операции стека показаны на рис.8.11.

Следующий программный код реализует стек с фиксированным массивом.

```
/*****************************************************************/
/* имя файла: stack.c                                            */
/*****************************************************************/
/
/*включенные файлы*/
#include <stdio.h>  /*стандартная библиотека I/O                  */
#include <stdlib.h> /*стандартная библиотека динамического распределения*/

/*global variables -объявляет глобальные переменные в header  file. */
/* Они приведены здесь, чтобы иллюстрировать общее построение программы.*/

/*определение структуры*/
struct stack_struct
{
int  stack_top;             /*следующая используемая позиция в стеке*/
int  stack_item[10]; /*элемент, сохраненный в стеке с фиксипрованым
размером */
}


typedef struct stack_struct stack;

               /*массив для распознавания имен различных переменных*/
```

```
typedef stack *stack_ptr;    /*определение указателей на стек */
/*функции-прототипы*/
void initialize_stack(stack);
void push(stack *, int); /*Используется метод передачи параметра по ссылке*/
int  pull(stack *);  /* когда содержимое стека должно измениться */
int  stack_empty(stack);
int  stack_full(stack);
void print_stack(stack);

/*переменные*/
int YES=1,NO=0;              /*логические флаги */
stack   stackl;              /*объявление стека  */
char *filename;
FILE *outputfile;

void main(void)
{
int response;
int stack_item;
/*печать ре5
Зультата в файл/
printf("\n\nEnter a filename for output.");
scanf("%s", filename);
outputfile = fopen(filename, "a");

initialize_stack(stackl);
response = stack_empty(stackl);         /*вызов по значению */
response - stack_full(stackl);
print_stack(stackl) ;
push(&stackl, 11);                      /*вызов по ссылке */
pushf&stackl, 12);
push(&stackl, 13) ;
push{&stackl, 14) ;
print_stack(stackl);
pull(ustackl);
pullf&stackl);
pull(&stackl);
pull(&stackl);
pull(ustackl);
fclose(outputfile);                     /*закрыть выходной файл */
}
/*****************************************************************/
/*initialize_stack: установить указатель вершины стека в 0            */
/*****************************************************************/

void initialize_stack(stack a_stack)
{

a_stack.stack_top=0;                    /*установить указатель стека в 0*/
}
/*****************************************************************/
/*stack_empty: возвращает ДА если стек пуст, и НЕТ в противном случае */
/*****************************************************************/

int stack_empty(stack a_stack)
```

```
{
fprintf(outputfile, "\n\nStack top: %d", a_stack.stack_top);

if(a_stack.stack_top == 0)    /*проверить не пуст ли стек*/
  {
  fprintf(outputfile, "\nStack Empty!");
  return YES;
}
     else
{
fprintf(outputfile, "\nStack is not empty.");
return NO;
}
    }
/*********************************************************************/
/*stack_full: возвращает ДА если стек полон, и НЕТ в противном случае */
/*********************************************************************/

int stack_full(stack a_stack)
{
if(a_stack.stack_top == 10)  /*проверить не заполнен ли стек */
  {                          /*произвольный выбор предела стека */

  fprintf(outputfile, "\n\nStack Full!");
  return YES;
  }
else
  {
  fprintf(outputfile, "\n\nStack is not full.");
  return NO;
  }
}
/*********************************************************************/
/*print_stack: печать текущего элемента (на вершине стека)      */
/*********************************************************************/

void print_stack(stack a_stack)
{
int  i;
if(!(stack_empty(a_stack)))/*проверить не пуст ли стек перед печатью*/
  {                         /*перейти коснованию стека перед печатью */
  for(i-a_stack.stack_top;i>=0;i=i-l)
  fprintf(outputfile, "\nStack item: %d", a_stack.stack_item[i]);
  }
else
  fprintf(outputfile,"\nCannot print - stack is empty!"); }

/*********************************************************************/
/*push(stack *, int): запись элемента в стек                         */
/*********************************************************************/

void push(stack *a_stack, int item)
{
fprintf(outputfile, "\n\nBefore push - stack pointer: %d",
                                          a_stack->stack__top);
```

```
if(!(stack_full(*a_stack)))  /*проверка заполнения стека*/
                             /* перед записью элемента*/
{
a_stack->stack_itern[a_stack->stack_top] = item;
fprintf(outputfile, "\nstack item after push: %d",
a_stack->stack_item[a_stack->stack_top]);
a_stack->stack_top - a_stack->stack_top + 1;
fprintf(outputfile, "\nstacktop after push: %d",
a_stack->stack_top);
}
else
fprintf(outputfile, "\nCannot push - stack is full!");
}
/*********************************************************************/
/*pull(stack *): извлечение элемента из стека
/*********************************************************************/

int pull(stack *a_stack)
{
int item;


fprintf(outputfile,"\n\nBefore pull - stack pointer: %d",
                                    a_stack->stack_top);
if(!(stack_empty(*a_stack))) /*проверка не пуст ли стек */
                             /*перед извлечением элемента*/
{
item = a_stack->stack_itern [a_stack->stack_top-l ] ;
fprintf(outputfile, "\nstack item pulled: %d",  item);
a_stack->stack_top = a_stack->stack_top - 1;
fprintf(outputfile,"\nstacktop after pull: %d",
a_stack->stack_top); return item; } else
fprintf(outputfile, "\nCannot pull - stack is empty!");
}
/*********************************************************************/
```

Мы показали работу этого примера на рис.8.12. После выполнения этой программы будет выдан следующий код:

```
Stack top: 0
Stack Empty!

Stack is not full.

Stack top: 0
Stack Empty!
Cannot print - stack is empty!

Before push - stack pointer: 0

Stack is not full.
stack item after push: 11
stacktop after push: 1

Before push - stack pointer: 1
```

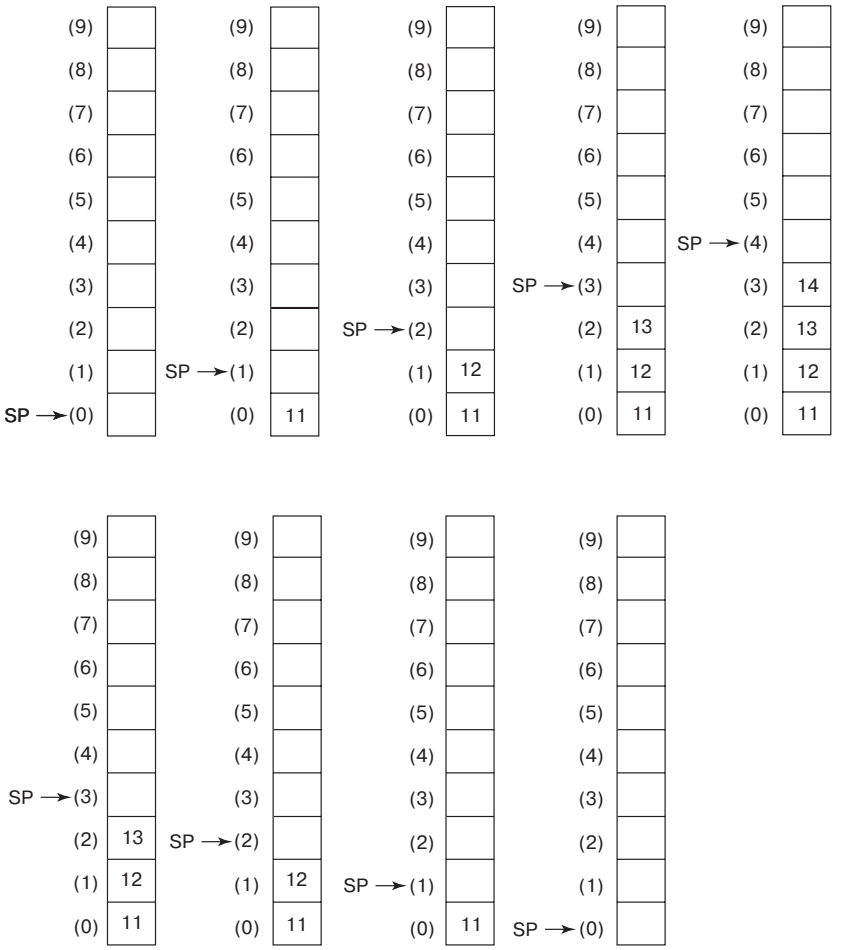


Рис. 8.12. *Запись в стек и извлечение из стека*

```
Stack is not full.
stack item after push: 12
stacktop after push: 2

Before push - stack pointer: 2

Stack is not full.
stack item after push: 13
stacktop after push: 3


Before push - stack pointer: 3

Stack is not full.
stack item after push: 14
```

```
stacktop after push: 4

Stack top: 4
Stack is not empty.
Stack item: 0
Stack item: 14

Stack item: 13
Stack item: 12
Stack item: 11

Before pull - stack pointer: 4

Stack top: 4
Stack is not empty
stack item pulled: 14
stacktop after pull: 3

Before pull - stack pointer: 3
Stack top: 3
Stack is not empty.
stack item pulled: 13

stacktop after pull: 2

Before pull - stack pointer: 2

Stack top: 2
Stack is not empty,
stack item pulled: 12
stacktop after pull: 1

Before pull - stack pointer: 1
Stack top: 1
Stack is not empty.
stack item pulled: 11
stacktop after pull: 0

Before pull - stack pointer: 0

Stack top: 0
Stack Empty!
Cannot pull - stack is empty!
```

Несколько стеков. Обычно система микропроцессора содержит один стек. Этот стек объявляется внутри RAM, и процессор имеет несколько функций для объявления положения стека (LDS), записи данных (PUSH), извлечение данных из стека (PULL) и т.д. Кроме того, как мы уже рассказывали в главе 4, в процессор встроен целый ряд аппаратных функций, связанных стеком, таких, как сохранение данных программного счетчика и ключевых регистров. В операционной системе реального времени нам нужен будет стек для каждой задачи, в котором мы будем сохранять контекст. Следовательно, мы должны иметь несколько стеков для работы с системами ОСРВ. В этих случаях, мы используем понятия о стеке, рассмотренные в этом

разделе. Мы могли бы легко объявлять дополнительные стеки, использовав приведенный выше код. Кроме того, таким же образом может работать любой из стеков, которые мы объявим.

На этом мы завершаем обзор основных конструкций, которые используются для реализации операционной системы в режиме реального времени. Мы теперь собираемся сместить акценты и обсудить дополнительные концепции ОСРВ в следующем разделе. Мы расстаемся с конструкциями и концепциями, чтобы описать, как программировать различные ОСРВ.

8.4. Основные понятия

Ранее в этой главе мы сказали, что ОСРВ – компьютерная операционная система, которая должна своевременно обрабатывать несколько событий при ограниченных ресурсах процессора. Наше исследование ОСРВ начинается с определения понятия задачи. Это потребует радикального изменения нашего понимания программ (сдвига парадигмы). При наличии в системе только одного последовательного процессора, мы можем рассматривать программу как последовательность шагов, которые процессор выполняет один за другим по определенному алгоритму. В ОСРВ, наша программа состоит из независимых, асинхронных (могущих появиться в любое время) взаимодействующих задач. И все они будут конкурировать за драгоценное (и ограниченное) время обработки. Наша программа состоит из механизмов, позволяющих следить за состоянием каждой задачи, планировать задачи для выполнения, и удостовериться, что каждая задача получает необходимую долю процессорного времени.

Мы начнем этот раздел, с получения хорошего описания того, что мы понимаем под задачей и как мы представляем ее в программе. Затем мы исследуем, как следить за состоянием каждой задачи и модифицировать его, используя блок управления задачами (task control block – TCB). Мы исследуем также, как отслеживается состояние другой системой информации, с помощью управляющих блоков устройства. Мы увидим, как диспетчер следит за состоянием всех задач и определяет, какая из задач является очередной. В заключение, мы также исследуем различные алгоритмы планирования, которые могут использоваться в ОСРВ.

8.4.1. Что такое задача?

Задача – это независимое, асинхронное действие, которое выполняется системой ОСРВ. Поскольку задачи асинхронны, мы не можем точно предугадать момент, когда они будут выполняться программой. Каждая задача может рассматриваться как маленькая, независимая программа, которая выполняет специфическое действие. Так как мы имеем несколько задач, конкурирующих за использование одного и того же процессора, задача должна иметь возможность сохранить контекст (ключевые значения регистров, счетчик программы, и т.д.). Эта информация резервируется на интервале выполнения другой задачи. Следовательно, каждая задача должна иметь свой стек для сохранения контекста. Даже если выполнение задачи прервано другой задачей, в конечном счете, его планируется завершить позднее.

В нескольких следующих разделах мы исследуем возможные состояния задач и способы, с помощью которых вся информация о задачах обрабатывается блоком управления задачами. До перехода к этому материалу рассмотрим предварительно задачи, связанные со знакомым уже нам роботом, проходящим через лабиринт.

Пример: В главе 7 был рассмотрен проект автономного робота, проходящего через неизвестный лабиринт. Этот робот, обнаруживая границы лабиринта с помощью инфракрасных локаторов, принимал решения, двигаться ли вперед или повернуть в необходимом направлении, чтобы пройти через лабиринт. При проходе через лабиринт робот должен был избежать земляных мин (магнитов в полу лабиринта). Как мы говорили, робот должен находить мины с помощью датчика Холла. Если робот обнаруживал магнит, он должен был остановиться, отъехать назад, и объехать мину. Робот был также оборудован ЖК дисплея (ЖКД), показывающим его текущее состояние в процессе выполнения программы.

Создадим список функций, которые должна выполнять операционная система робота, чтобы успешно выполнять все перечисленные задачи:

- Функции инициализации ЖКД, ATD-преобразователя и системы широтно-импульсной модуляции (ШИМ);
- ATD-преобразование выходных сигналов ИК датчиков;
- Сравнение выходных сигналов ИК датчика с пороговыми уровнями обнаружения стенки;
- Алгоритм поворотов робота, позволяющий правильно изменить движение робота в ответ на выходные сигналы ИК датчиков;
- Функции, позволяющие осуществить поворот робота направо, налево и продолжить движение вперед;
- Метод обработки выходного сигнала датчика Холла;
- Функции, необходимые для выполнения объезда мины – остановка, задний ход, и объезд;
- Функции, обеспечивающие работу ЖКД.

Эти функции показаны в структуре программы (рис.8.13).

При реализации ОСРВ робота эти функции становятся задачами. Как мы уже сказали, задачами называются независимые, асинхронные и взаимодействующие процессы, соревнующиеся за предоставление процессорного времени. Исследуем сценарий работы системы управления роботом, чтобы иллюстрировать это.

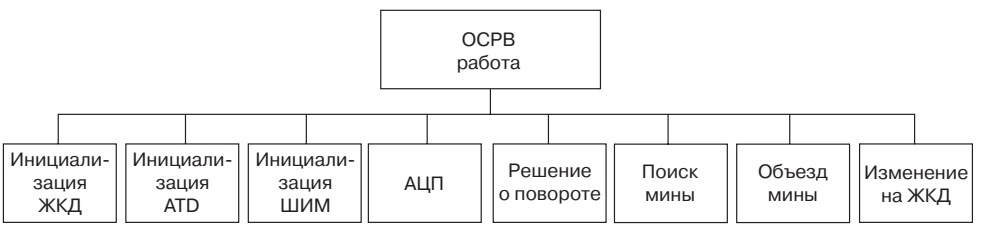


Рис. 8.13. *Структура программы, управляющей роботом, проходящим лабиринт*

Сценарий. Робот помещается в начальную точку неизвестного лабиринта, содержащего магнитные мины. Операционная система инициализирует ЖКД, ATD-конвертер и систему ШИМ. В главе 4, мы обсуждали требования инициализации для каждой из этих систем робота.

Как только инициализация закончена, робот начинает проход через неизвестный лабиринт, обрабатывая сигналы ИК датчиков и датчика Холла. Что получается, если робот получит сигналы о приближении к стенке одновременно с сигналом об обнаружении мины? Робот не сможет обрабатывать оба эти два события одновременно, поскольку располагает только одним процессором. Оба события являются критическими, хотя и не в равной степени. Если мы обрабатываем сначала информацию о стенках, робот избегает столкновения, но рискует подорваться на мине, если обработка информации о стенках не закончится достаточно быстро. С другой стороны, если мы сначала обрабатываем информацию о минах, считая эту задачу более приоритетной, мы подвергаемся риску возможного столкновения со стенками. К тому же оба события взаимосвязаны. Мы не хотим подорваться на мине, обрабатывая информацию о стенках, но и не хотим наткнуться на стенки объезжая мины.

В следующих разделах мы обсудим, как решить рассматриваемые проблемы, используя ОСРВ и, прежде всего, познакомимся с концепциями управления задачами.

8.4.2. Управление задачами

В этом разделе мы рассмотрим, как ОСРВ взаимодействует с отдельной задачей. Сначала мы рассмотрим различные состояния, в которых может находиться задача. Затем исследуем, как задачи переходят из одного состояния в другое, и рассмотрим программную функцию, позволяющую моделировать состояния задач и их связь друг с другом. Это обсуждение будет частью полного обсуждения блока управления задачами.

Состояния задачи. В любой момент времени задача может быть только в одном определенном состоянии. Различные состояния задачи показаны на рис. 8.14. ОСРВ должна отслеживать и изменять состояние каждой задачи. Не забудем при этом, что все задачи независимы, асинхронны и претедуют на процессорное время. ОСРВ должна обеспечить эффективное планирование этого дефицитного ресурса.

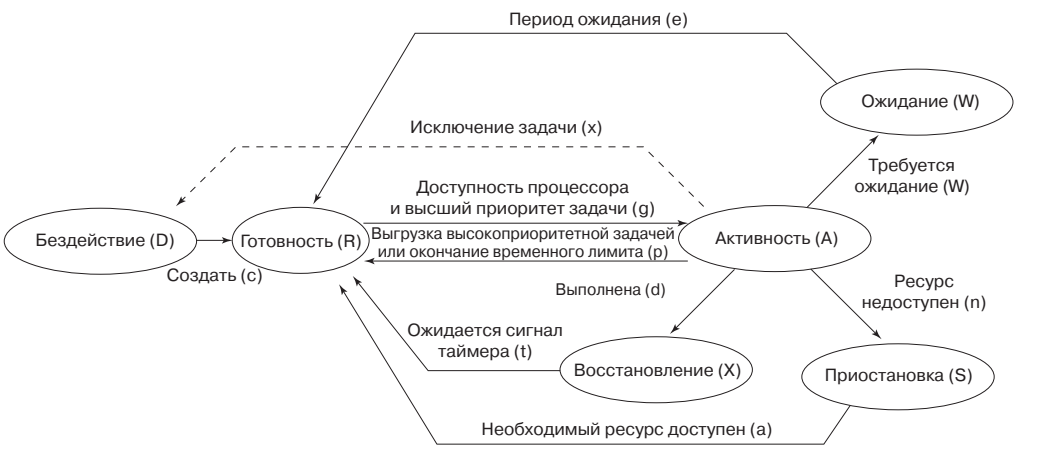


Рис. 8.14. *Состояния задач*

Задача находится в одном из следующих состояний:

- Бездействия (Dormant – D): задача не нуждается в обработке и не требует процессорного времени. Она рассматривается как удаленная или неактивная задача, и по сигналу операционной системы переходит в состояние готовности.
- Готовности (Ready – R): задача полностью готова к переходу в активное состояние; однако, в настоящее время процессор занят другой задачей. Задача может переходить в состояние готовности из состояний бездействия или активности. Она переходит от активного состояния в состояние готовности, когда процессор обрабатывает другую более приоритетную задачу. Зарезервированная задача с более низким приоритетом (находящаяся в состоянии готовности) повторно переходит в активное состояние, как только становится доступным процессорное время и операционной системой дается разрешение на выполнение.
- Активности (Executing/active/running – A): задача управляется процессором, выполняя часть своей программы. Так как наша система содержит только один процессор, только одна задача может быть в активном состоянии в любое данное время. Задача остается в активном состоянии, пока не происходит одно из трех событий:
  1) завершаются необходимые для выполнения задачи действия;
  2) она выгружается задачей с более высоким приоритетом;
  3) она возвращает управление операционной системе.

  Во всех этих случаях задача переходит от активного состояния в состояние готовности. Эти варианты станут более ясными, когда мы обсудим различные типы систем ОСРВ в разделе 8.5. Из активного состояния задача может также переходить в состояния ожидания.
- Ожидание (Wait – W): выполнение задачи было отсрочено. Она остается в ждущем состоянии на заданном отрезке времени и затем переходит в состояние готовности, ожидая обработки. Задача переводится в состояние ожидания временно, чтобы выделить время для обработки задач с более низким приоритетом.
- Приостановка (suspended – S): задача ждет некоторого ресурса. Как только ресурс становится доступным, задача переходит в состояние готовности и ждет процессорного времени.
- Восстановление (rescheduling – X): Это состояние вводится каждый раз, когда задача выполнена, но не может сразу же перейти в состояние готовности. В этом случае, задача остается в состоянии восстановления, пока не закончится необходимый интервал восстановления (RSI). Как только это происходит, задача снова переходит в состояние готовности.

Приведенная диаграмма состояний задач может быть выполнена с помощью программного кода, использующего команду switch, которая позволяет нам точно управлять переходами в зависимости от состояния.

```
/*****************************************************************/
char task_state_diagram(char present_state, char action)
{
char  next_state;

switch(present_state){
case 'D': /*если состояние бездействия (D) и выбрана опция create(c), то */
```

```
            /* задача переходит в состояние готовности, в противном  */
            /* случае  она остается в состоянии бездействия (D)      */
            if(action == 'c')
               next_state - 'R';
            else
               next_state - 'D';
            break;

case 'R':   /*если задача в состоянии готовности (R), процессор доступен */
            /* и задача имеет наивысший приоритет,то она переходит в   */
            /* активное состояние (A). Это состояние обозначается     */
            /*присвоением переменной  action значения (g).            */

            if(action == 'g')
               next_state = 'A';
            else
               next_state = 'R';
            break;

case 'A': /*из состояния активности(A)задача переходит в состояние  */
            /*определяемое значением  переменной action.              */

            if(action == 'p')
                              /*время ожидания или приостановка задачи*/
               next_state = 'R';        /*возврат в состояние готовности*/
            else if (action == 'w' )
                                       /*переход в состояние ожидания*/


               next_state = 'W';        /*состояние ожидания          */
            else if (action == 'n')             /*ресурс недоступен  */
               next_state = 'S';
                              /*задача переходит в состояние приостановки*/
            else if (action == 'd')
                              /*завершается выполнение задачи*/
               next__state = "X" ;
                              /*если требуется время для              */
                              /* восстановления                       */
            else if (action == 'x')      /*задача исключается         */
               next_state = 'D' ; /*возврат в состояние бездействия   */
            else next_state = 'A'; /*остается в состоянии активности */
            break;

case 'X':     /*из состояния восстановления (X) переход в состояние  */
              /*готовности (R)по сигналу таймера восстановления (t). */

            if(action=='t')    /*ожидается сигнал таймера восстановления*/
            next_state='R';    /*задача переходит в состояние готовности*/
            else
            next_state='X' ;
            break;

case 'W' : /*из состояния ожидания (W) переход в состояние готовности (R)*/
            /*по сигналу таймера ожидания  (e). */
```

```
            if(action == 'e')          /*ожидание сигнала таймера*/
               next_state = 'R';       /*возврат в состояние готовности*/
            else
               next_state = 'W';
            break;

case 'S': /*из состояния приостановки (S) переход в состояние готовности (R)*/
          /*при появлении ожидаемого ресурса (a)                          */

            if(action == 'a')          /*необходимый ресурс доступен*/
              next_state - 'R';        /*переход в состояние готовности*/
            else
               next_state - 'S';
            break;

return next_state;
}

}
/*****************************************************************/
```

Блок управления задачами. Как мы уже упоминали, задачи представляют собой независимые, асинхронные и взаимодействующие процессы, для взаимосвязанного выполнения которых необходим один и тот же процессор. Давайте вспомним о нашем бесстрашном официанте из предшествующего раздела. Официант (процессор) должен удовлетворять запросы нескольких заказчиков (задач), используя при этом ограниченные ресурсы. Так как официант (процессор) переключается с обслуживания одного столика заказчиков (от одной задачи) к другому, ему, или ей необходимо помнить состояние (контекст) каждого из столиков (задач), чтобы обеспечить качественное обслуживание.

В этом разделе мы рассмотрим, как ОСРВ «запоминает» и отслеживает состояние каждой задачи, чтобы позволить каждой задаче выполнить свой процесс. Вы, наверное, думаете: «А в чем сложности? Позвольте процессу, который стал активным завершиться». Но как мы иллюстрировали в примере с роботом, это невозможно. Так, в системе с большим количеством конкурирующих задач, некоторые задачи низким приоритетом никогда не выполнялись бы процессором. Вообразите, что случилось бы, если наш официант (процессор) посвятил все свое время одному столику (процесс) до полного его обслуживания, не обращая внимания на все остальные столики (задачи).

Мы не знаем точно, как хороший официант следит за многими столиками одновременно. А вот ОСРВ для отслеживания состояния каждой задачи обычно использует блок управления задачами (TCB). Каждая задача в ОСРВ имеет собственную связь с TCB, которая обеспечивает текущую информацию о задаче. Эта информация используется и модифицируется ядром ОСРВ, чтобы эффективно отслеживать, планировать и выполнять весь набор задач данной системы. Мы должны подчеркнуть, что TCB изменяется только операционной системой. Задача не имеет прямого контакта с TCB, хотя этот блок содержит наиболее обновленную информацию о задаче.

Структура TCB показана на рис.8.15. Когда ОСРВ переключается с одной задачи на другую, вся ключевая информация о текущей задаче должна быть надежно сохранена перед переключением к следующей задаче. Таким образом, когда задача позже получает процессорное время, она может продолжить процесс с того места,

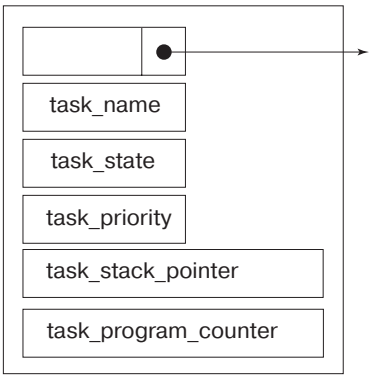


Рис. 8.15. *Блок управления задачами*

где он был остановлен, при выгрузке. Как мы говорили ранее, эта ключевая информация задачи называется контекстом.

Прервемся на минуту, и подумаем, какая информация была бы важна для задачи. Рассмотрим также, как мы могли бы выполнить программное обеспечение TCB. Как минимум TCB должен содержать имя задачи, текущее состояние, приоритет, текущий контекст (ключевые значения регистров), и адрес, по которому можно найти этот контекст для обработки задачи.

Сначала, попробуем справится с разработкой программы для TCB. Данные, отражающие отдельные свойства каждой задачи имеют различные типы. Ранее в этой главе мы обсуждали запись или структуру – абстрактный тип данных, хорошо подходящий для программной реализации TCB. Как было упомянуто, структура представлят собой определяемую пользователем совокупность различных, но связанных между собой типов данных. Мы можем объединить эти различные типы данных в структуру и использовать ее для создания программы TCB. В главе мы уже разработали структуру для автомобиля и следили за самой разнообразной информацией о конкретных автомобилях, заключенной в полях структуры. Мы сделаем теперь то же самое для TCB. (На самом деле, мы попросим, чтобы это сделали вы в качестве задания 4 для самостоятельной работы). Используйте автомобильную структуру в качестве примера, и разработайте подобную структуру для TCB.

Рассмотрим подробнее отдельные поля для TCB и определим типы данных для каждого поля.

- ***Имя задачи состоит из символьного массива.*** Для примера робота мы ограничим имя задачи 20 символами. Это позволит нам сохранить полное имя для каждой задачи, не пользуясь маловразумительными сокращениями. Мы позволяем себе такую расточительность в использовании памяти ради удобочитаемости.
- ***Текущее состояние задачи:*** как следует из предыдущего раздела, задача в каждый момент времени может находиться в одном из шести состояний - бездействия (D), готовности (R), активности (A), ожидания (W), приостановки (S) и восстановления (X). Мы обозначили каждое состояние задачи одним символом. Мы можем, следовательно, сохранять текущее состояние задачи в символьной переменной внутри нашей структуры TCB.

- ***Приоритет задачи:*** Для относительного ранжирования задач внутри системы, разработчиком системы назначается приоритет задачи. В нашем примере с роботом, мы использует фиксированные приоритетные значения. Но при необходимости можно выполнить и ОСРВ системы, в которых приоритет задачи может изменяться в процессе выполнения программы. Приоритет задачи представляется натуральными числами. В следующем примере мы покажем, как назначать приоритет задачи.
- ***Контекст задачи:*** Содержимое всех ключевых регистров связанных с задачей, составляет контекст задачи. Система прерывания, встроенная в микропроцессор 68HC12 использует стек, чтобы сохранить весь контекст, когда прерывание происходит. Мы также используем стек, чтобы сохранить контекст задачи. Каждая задача имеет собственный стек, следовательно, ОСРВ имеет целый ряд одновременно существующих стеков. Как программист системы, вы должны гарантировать, что эти стеки не будут смешиваться друг с другом в пространстве памяти. Для простоты, мы используем фиксированную структуру стека, разработанную ранее в этой главе для контекстной памяти TCB. Так как все ключевые регистры и ячейки памяти в микроконтроллере 68HC12 имеют объем в 8 или 16 бит, мы используем фиксированный массив целых чисел для стеков TCB.
- ***Состояние активности задачи:*** состояние активности задачи представляет собой следующий шаг программы, который должен выполняться, как только задача станет активной. После инициализации, задача начинается с начала соответствующего ей программного кода. Однако до своего полного завершения задача может неоднократно выгружаться задачами с более высоким приоритетом. Следовательно, TCB должен помнить следующий шаг, с которого должно продолжиться выполнение задачи.
- ***Указатель задачи:*** Так как мы будем связывать эти структуры TCB с помощью указателей, нам необходим указатель на следующий в списке TCB.

Вы уже, наверное, усвоили понятиям задачи и блока управления задачей. Однако мы еще не касались ряда сложных проблем. Вам, вероятно, не очень ясна идея выхода из программы до ее завершения, даже если она обладает наивысшим приоритетом. Мы исследуем эту тему в следующем разделе. Вы можете воображать осложнения, которые получатся, если мы начнем ATD- преобразование, а оно выгрузится событием с более высоким приоритетным прежде, чем будет закончено. И представьте себе, какая путаница возникнет, если это событие с более высоким приоритетом также должно будет использовать ATD- преобразование. Мы видим, что разработчик операционной системы должен определить, когда можно безопасно прервать управление задачи. Перед исследованием этих проблем, возвратимся к нашему примеру робота и посмотрим, как назначаются приоритеты задач.

Сценарий: В нашем примере с роботом у нас было много задач по обеспечению его работы. Теперь мы должны назначить численное значение приоритета для каждой задачи. Мы будем использовать более низкое численное значение для задач с более высоким приоритетом. Например, задаче с самым высоким приоритетом сопоставим значение 1. Мы используем наше понимание сценария работы робота, чтобы назначить приоритеты задач. Так как наш робот дезактивируется, когда приближается к мине, мы присваиваем задаче обнаружения мин приоритет 1.

Следующий самый высокий приоритет, равный 2 присвоим операции объезда мины. Задаче ATD-преобразования присвоим приоритет 3, так как она обеспечивает информацию о близости стенок лабиринта. Задаче выбора поворота присвоим следующий приоритет 4, так как она обрабатывает информацию, необходимую, чтобы избежать столкновения со стенками. Наконец, задаче модификации ЖКД назначаем приоритет 5. Это самый низкий приоритет для задач, рассмотренных к настоящему времени; однако, приоритет всех прочих задач еще ниже. Остающиеся задачи имеют еще более низкий приоритет. Они активны на начальных этапах работы нашей операционной системы, а затем входят в состояние бездействия. Мы, следовательно, назначаем им самый низкий приоритет, давая им приоритеты 6 (инициализация ЖКД), 7 (инициализация ATD), и 8 (инициализация ШИМ).

Фрагментация задач. Как мы уже выяснили в этом разделе, желательно разделить программный код задачи на непрерываемые части или фазы, определив удобные точки выхода. Это важно, поскольку в ОСРВ с несколькими задачами с одинаковым приоритетом, задача редко будет способна завершить связанные с ней действия от начала до конца. Чаще всего задача завершит некоторую часть действий и затем будет приостановлена задачей или задачами с более высоким приоритетом. Факт перехода задачи с более высоким приоритетом в состояние готовности еще не означает, что процессорное время немедленно передается ей. Мы должны организовать переход от одной задачи к другой, прервав выполнение кода в заранее определенном удобном для этого месте. Поскольку мы записываем контекст прерываемой задачи, мы должны сохранить его в памяти. Например, мы можем подразделять связанную с задачей функцию на три части. Первый раз, когда задача становится активной, процессор выполняет первую треть кода до удобной отметки прерывания (определенной вами, как программистом). Когда код достигает этой отметки прерывания, контекст сохраняется в TCB, и процессор прерывает управление задачей. Затем выполняется задача с более высоким приоритетом. Когда прерванная задача снова получает для своего выполнения драгоценное процессорное время, она начинается с того места, где была прервана и продолжает обработку второй части своего кода. Этот процесс продолжается, пока задача не завершает все предусмотренные действия.

Проиллюстрируем такую работу примером.

Пример: Предположим, что робот, имеющий пять ИК локаторов (рис.8.16) выполняет функцию названную `process.turn`, которая инициализирует систему ATD контроллера 68HC12, начиная последовательность преобразований, необходимую, чтобы записать аналоговые сигналы от пяти датчиков (с номерами от 0 до 4), которые связаны с каналами ATD от 7 до 3, соответственно. Выход датчика Холла, установленного в нижней части робота, чтобы обнаруживать магнитные мины, связан с каналом 2 ATD. Обратите внимание: этот пример придуман, чтобы показать, как следует подразделять код, чтобы обеспечить удобные точки прерывания.

Код `process.turn`, обеспечивающий процесс поворота, приведен ниже.

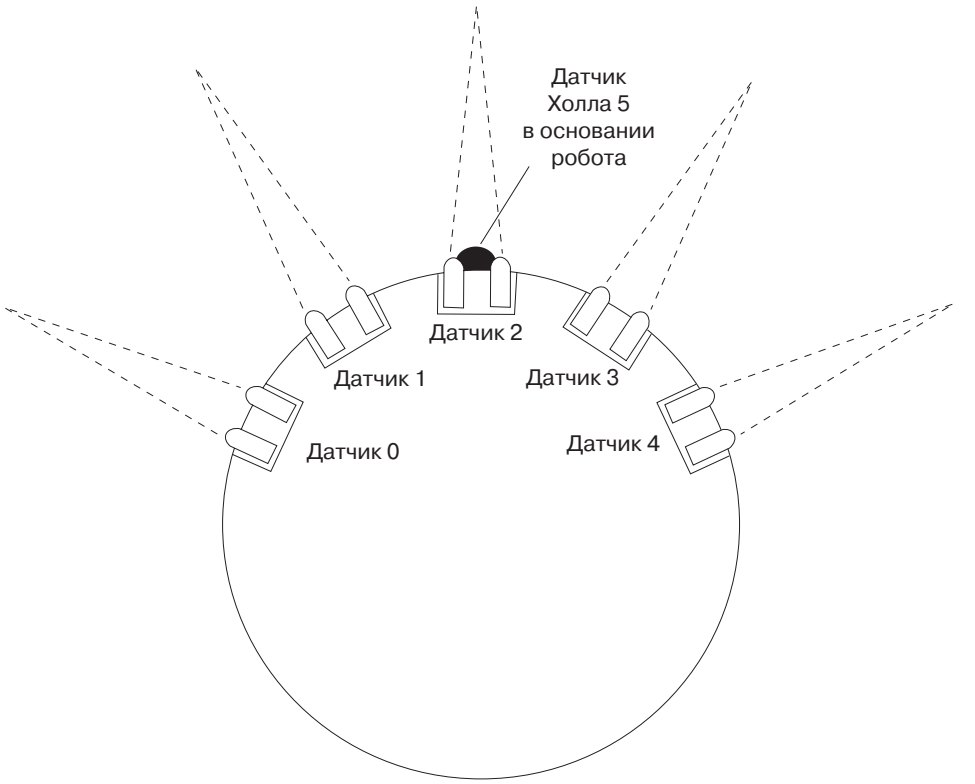


Рис. 8.16. *Робот с пятью ИК локаторами и датчиком Холла. ИК-датчик обнаруживает присутствие стенок лабиринта, в то время как датчик Холла обнаруживает присутствие магнитных мин.*

```
void process_turn()
{
                                          /*Инициализация системы ATD  */
ATDCTL2 = 0x80; /*установка флага ADPU, чтобы подать питание на систему ATD*/
ATDCTL3 = 0x00; /*игнорировать «замораживание» системы                  */

ATDCTL4 = 0x7F;      /*Снижение частоты таймера  P до 125 кГц        */
                     /*выборка, время преобразования = 32 ATD цикла */
                     /* 1 ввыборка за каждые 256 мкс                 */
for(i=0; i<67; i++){          /* ожидание 100 мкс при 8 МГц ECLK*/
  ;
  }
                     /*Инициализация ATD-преобразования              */
ATDCTL5 = 0x50;      /*Начать многоканальное ATD-преобоазование      */
                     /* для 8 каналов                                */

while((ATDSTAT & 0x8000) ==0){ /* проверить окончание преобразования по*/
                            /*состоянию флага SCF                    */
  ;
  }
```

```
/* сохранить результаты  ATD-преобразования*/
в глобальном массиве char*/
sens[0] = ADR7H;                        /*крайний левый датчик */
sens[1] = ADR6H;                        /*средний левый датчик */
sens[2] = ADR5H;                        /*центральный датчик */
sens[3] = ADR4H;                        /*средний правый датчик */
sens[4] = ADR3H;                        /*крайний правый датчик */
sens[5] = ADR2H;                        /*Датчик Холла*/



/*анализ информации датчиков для решения о повороте. Примечание: пороги для*/
/*датчика Холла(hes_threshold) и для ИК-датчиков (opto_threshold)являются*/
/* глобальными переменными и определены экспериментально*/

if(sens[5] < hes_threshold){ /*сигнал с датчика Холла, объезд*/
  pwm_motors(back_up);       /* робот дает задний ход*/
                             /*действия, следующие после того  */
                             /* как робот отъехал назад         */
if(sens[0] > opto_threshold)
  pwm_motors(right_turn);
else
  pwm_motors(left_turn);

for(i=0; i<0xFFFF; i++){     /*задержка перед вращением двигателя   */
  for(j=0; j<15; j++){
    ;
    }
  }
}

                             /*если обнаружен тупик - задний ход*/
else if((sens[2]>opto_threshold)&&(sens[0]>opto_threshold)
&&(sens[4]>opto_threshold)){
pwm_motors (back__up) ; }

                             /*если стенки спереди и слева,  */
                             /*поворот робота направо         */
else if((sens[0]>opto_threshold)&&(sens[2]>opto_threshold)){
pwm_motors (right__turn) ; }

                             /*если стенки спереди и справа,  */
                             /*поворот робота налево          */
else if((sens[2]>opto_threshold)&&(sens[4]>opto_threshold)){
pwm_motors(left_turn);}

                             /*если стенка перед средним правым  */
                             /* датчиком, то полуповорот направо */

else if(sens[1]>opto_threshold){
pwm_motors (half_right) ; }

                             /*если стенка перед средним левым  */
                             /* датчиком, то полуповорот налево */
else if(sens[3]>opto_threshold){
pwm_motors(half_left);}
```

```
/*если сигналов от датчиков нет, продолжить движение вперед*/

else{
  pwm_motors(forward);}
}
```

Если мы хотим подразделить этот код на три части обрабатываемые ОСРВ без прерывания, мы можем вставить точки прерывания после последовательности инициализации ATD и после последовательности записи данных с ATD. Это позволит функции без проблем прерывать и восстанавливать управление процессором. Чтобы выполнять эти изменения, мы должны ввести переменную, которую мы назовем `code_section`. Эта переменная позволит нам проследить, какая из трех частей кода должна быть выполнена при очередной активности задачи.

```
Int process_turn(int code_section)
{

switch(code_section){

  case 0:
                          /*Инициализация системы ATD */
    ATDCTL2 = 0x80;  /*включение ATD */
    ATDCTL3 = 0x00; /*игнорировать доступ при отладке системы      */
    ATDCTL4 = 0x7F; /*Снижение частоты таймера  P до 125 кГц       */
                    /*выборка, время преобразования = 32 ATD цикла */
                    /* 1 ввыборка за каждые 256 мкс                */

    for(i=0; i<67; i++)
      {             /* ожидание 100 мкс при 8 МГц ECLK*/
      ;
      }

      code_section = 1; /*update code__section variable
      break;

  case 1:
                          /*Инициализация ATD-преобразования     */
    ATDCTL5 = 0x50;       /*Начать многоканальное ATD-преобоазование*/
                          /* для 8 каналов                        */
    while((ATDSTAT & 0x8000] = = 0)
     {
                          /* проверить окончание преобразования по*/
                          /*состоянию флага SCF                   */
      ;
      }

                          /* сохранить результаты  ATD-преобразования*/
                                  /* в глобальном массиве char */
    sens[0] = ADR7H;              /*крайний левый датчик */
    sens[1] = ADR6H;              /*средний левый датчик */
    sens[2] = ADR5H;              /*центральный датчик */
    sens[3] = ADR4H;              /*средний правый датчик */
    sens[4] = ADR3H;              /*крайний правый датчик */
    sens[5] = ADR2H;              /*Датчик Холла
```

```
     code_section = 2;                  /*update code_section variable
     break;

case 2:

/*анализ информации датчиков для решения о повороте. Примечание: пороги для*/
/*датчика Холла(hes_threshold) и для ИК-датчиков (opto_threshold)являются*/
/* глобальными переменными и определены экспериментально*/

if(sens[5] < hes_threshold){ /*сигнал с датчика Холла, объезд*/
  pwm_motors(back_up);       /* робот дает задний ход*/
                             /*действия, следующие после того  */
                             /* как робот отъехал назад         */
  if(sens[0] > opto_threshold]
    pwrn_motors(right_turn);
  else
    pwm_motors(left_turn);


for(i=0; i<0xFFFF; i++){     /*задержка перед вращением двигателя   */
  for(j=0; j<15; j++){
      ;
      }
   }
}

/*если обнаружен тупик - задний ход*/
else if((sens[2]>opto_threshold)&&(sens[0]>opto_threshold)
&&(sens[4]>opto_threshold)){
pwm_motors (back__up) ; }

                             /*если стенки спереди и слева,  */
                             /*поворот робота направо         */
else if((sens[0]>opto_threshold)&&(sens[2]>opto_threshold)){
pwm_motors (right__turn) ; }

                             /*если стенки спереди и справа,  */
                             /*поворот робота налево          */
else if((sens[2]>opto_threshold)&&(sens[4]>opto_threshold)){
pwm_motors(left_turn);}

                             /*если стенка перед средним правым  */
                             /* датчиком, то полуповорот направо */


else if(sens[1]>opto_threshold){
pwm_motors (half_right) ; }

                             /*если стенка перед средним левым  */
                             /* датчиком, то полуповорот налево */
else if(sens[3]>opto_threshold){
pwm_motors(half_left);}
                             /*если сигналов от датчиков нет         */
                             /*продолжить движение вперед            */
else{
```

```
  pwm_motors(forward);}

code_section = 0 ;           /* изменить переменную code_section    */
break;

}/*конец switch*/
return code section;

}
```

Когда задача, связанная с функцией `process_turn`, переходит из состояния готовности в активное состояние, ОСРВ вызывает функцию с параметром 0. Функция `process_turn` затем выполняется до первой отметки прерывания в коде. Достигнув этой отметки, функция возвращает управление ОСРВ, которая модифицирует TCB, связанный с процессом и продолжает выполнение второй части кода, когда задача в очередной раз переходит в активное состояние. Затем задача снова возвращается в состояние готовности и ждет, когда ОСРВ выделит ей процессорное время. Повторим снова, что причина, по которой мы делим код на логические части, состоит в том, чтобы позволить задаче работать до завершения определенной части и затем позволить другой задаче выполнить часть связанного с ней кода, и т.д. Это дает возможность выполнять несколько появившихся задач практически одновременно, хотя в любой момент времени процессор выполняет только одну задачу.

В этом примере мы использовали глобальные переменные, чтобы сохранять информацию между последовательными обращениями к функции `process_turn`. Использование глобальных переменных может быть не самое лучшее использование дефицитных ресурсов памяти RAM. Далее в этой главе мы рассмотрим альтернативные методы движения информации между задачами.

Как можно предположить из этого примера, ядро ОСРВ должно быть довольно сложным, чтобы позволить следить за множеством задач, эффективно планировать использование процессорного времени и полностью выполнять функции, необходимые для конкретного применения. В следующем разделе мы исследуем составляющие части ядра ОСРВ и их взаимодействие друг с другом, позволяющие удовлетворить требования применения.

8.4.3. Компоненты многозадачных систем

Чтобы выполнить многозадачную операционную систему, разработчик системы (которым являетесь вы) должен рассматривать управление прикладным устройством как группу независимых, но взаимодействующих задач. Каждая задача выполняет ряд связанных с ней действий. Цель многозадачной системы состоит в том, чтобы выполнить требования к прикладному устройству, изменяя задачи, планируя и реализуя их осуществление и решая конфликты, возникающие между различными задачами, из-за использования только одного процессора. Как вы могли уже себе представить, многозадачная система может быть очень сложной. Однако мы разбиваем систему на составные части и исследуем каждую из них отдельно.

Операционная система должна следить за состоянием каждой из управляемых ей задач. Обратимся снова к примеру с автомобильными дилерами, в котором мы использовали списки с указателями, чтобы учесть текущее состояние каждого автомобиля: включение его в партию, предназначенную для продажи, продажу, ремонт, перекраску и т.д.; мы удаляли его из одного списка и добавляли в другой при изменении состояния. ОСРВ также использует подобные методы, чтобы проследить за состоянием каждой задачи.

Обычно ОСРВ позволяет частично выполнять задачу на некотором ограниченном временном промежутке. Когда время, выделенное для задачи истекает, ОСРВ сохраняет текущий контекст задачи в связанном с ней блоке управления TCB. Затем ОСРВ решает, какой из задач предоставить очередную часть процессорного времени. Для принятия этого решения может использоваться ряд алгоритмов планирования. Но какой бы алгоритм планирования не был выбран, он должен удовлетворять основному требованию: каждой задаче должна быть выделена оптимальная доля процессорного времени.

Прежде, чем задача сможет перейти в состояние готовности, ОСРВ должен гарантировать, что ей будут доступны все необходимые ресурсы. Под ресурсами понимаются специфические данные, аппаратные подсистемы и т. д. Например, если задача должна использовать одну из подсистем микроконтроллера 68HC12, ОСРВ должны гарантировать, что этот ресурс доступен для задачи, переходящей в состояние готовности. Задача же, которая ожидает необходимого ресурса, должна быть переведена в состояние ожидания.

Процессор должен также иметь возможность приостановить задачу на некоторое время, чтобы предоставить процессорное время задачам с более низким приоритетом. Если мы не сделаем этого, выполняя все время задачи с самым высоким приоритетом, то ряд задач просто никогда не получит процессорного времени. Чтобы предотвратить эту ситуацию, активные задачи с высоким приоритетом должны временно переводиться в ждущее состояние, чтобы позволить частично выполнить низкоприоритетные задачи, находящиеся в состоянии готовности. Операционная система должна также иметь механизмы, позволяющие выполнять критические задачи с наивысшим приоритетом по мере их появления. Это подразумевает использование прерываний в обработке ОСРВ.

Для выполнения этих действий ядро ОСРВ использует ряд инструментов, включая системные таблицы, отслеживающие состояние задач и диспетчеры/ планировщики, исследующие данные системы, чтобы определить, какая из задач должна выполняться в любой момент. ОСРВ также обеспечивает возможность осуществления межзадачных связей, которую мы рассмотрим далее в этой главе.

Системные таблицы. Операционная система используют ряд таблиц /блоков, чтобы проследить состояние задач, устройств ввода – вывода и услуг системы. Мы уже обсудили в общих чертах блок управления задачами TCB в разделе 8.4. Кроме TCB, операционная система также поддерживает управляющий блок устройства (DCB), чтобы отслеживать состояние связанных с системой устройств. Это позволяет операционной системе гарантировать, что все требуемые задачей ресурсы доступны для использования, до того как будет дано разрешение на переход задачи в состояние готовности. В зависимости от числа устройств и ресурсов в системе, DCB может быть введен с помощью простого двумерного массива, который может динамически изменяться при изменении состояния устройства.

Пример: Когда мы наблюдали по телевидению феноменальную игру в гольф Лесного Тигра (Matt Christopher, прим. переводчика), то были заинтригованы тем, как же обслуживающий персонал поля для гольфа способен проследить за состоянием большого числа игроков в гольф (задач) на турнире, чтобы предоставлять им лунки (ресурсы). В процессе передачи, мы увидели большое табло, на котором отслеживалось состояние игроков и лунок. Обслуживающий персонал поля для гольфа мог сообщить состояние данного игрока или лунки. Табло состояния постоянно изменялось в течение турнира. ОСРВ использует тот те же методы (TCB и DCB) чтобы проследить состояние задач, ресурсов и услуг в процессе выполнения программы.

Диспетчер / планировщик. Диспетчер / планировщик – другая ключевая часть ядра ОСРВ. Первичная его функция состоит в том, чтобы определить, какая из задач является очередной. Планировщик может использовать ряд алгоритмов, чтобы принять это решение. В следующем разделе обсуждаются различные алгоритмы и свойственные им недостатки и преимущества.

8.5. Типы операционных систем реального времени

Обычно, операционные системы реального времени характеризуются типом алгоритма, используемого звеном диспетчера/планировщика, имеющимся в ядре ОСРВ. Уже название каждого типа ОСРВ дает достаточно хорошее описание работы его алгоритма. Проведем сравнительный обзор различных типов ОСРВ. Мы должны подчеркнуть, что ни один из рассматриваемых алгоритмов планирования ОСРВ не лучше любого другого. Выявление алгоритма, наиболее подходящего для конкретного применения является задачей программиста, создающего систему ОСРВ.

8.5.1. Системы с циклическим опросом

Система с циклическим опросом является наиболее простым ядром планирования ОСРВ, как для разработки, так и для исполнения. Как подразумевается самим названием, оно использует механизм последовательного опроса, чтобы определить, требует ли данная задача процессорного времени. Когда задача его требует, оно предоставляется ей от начала до конца выполнения. Как только задача заканчивается, операционная система продолжает опрос других задач, требующих процессорного времени.

Системы с циклическим опросом используются при управлении многозадачными системами с равным приоритетом для всех задач. Она должна также использоваться для систем с задачами, которые вряд ли потребуют процессорного времени одновременно. Так как одиночная задача выполняется до завершения, межзадачная связь в таких операционных системах обычно не требуется.

Основное преимущество системы с циклическим опросом – простота. С этим типом системы просто определить время срабатывания специфической задачи, систему просто написать и отладить. Основной недостаток такой системы – также

простота. Система с циклическим опросом не может обрабатывать пакет событий – то есть несколько задач, требующих процессорного времени одновременно. Следует подчеркнуть, что система с циклическим опросом – не лучший выбор сценария работы операционной системы реального времени. Она имеет существенные ограничения, однако идеально подходит для некоторых приложений, в чем мы убедимся на следующем примере.

Пример: В главе 2 мы обсуждали встроенную систему управления, предназначенную для обработки входных сигналов дистанционного управления или группового ввода для стерео усилителей. В этом специфическом примере, операционная система инициализировала систему усилителя и затем непрерывно опрашивала входные сигналы от пульта дистанционного управления и переключателей на лицевой панели. Интерфейс дистанционного управления связан с портом A, а переключатели на лицевой панели – портом B. Для этого применения, опрос был наилучшим метод для реализации операционной системы. Он был выбран в качестве предпочтительного, потому что 1) как только усилитель инициализирован, операционная система, выполняет только опрос сигналов на входах, 2) входы от дистанционного управления и переключателей лицевой панели имеют равный приоритет и 3) время реакции на изменение входных сигналов не критично.

8.5.2. Циклический опрос с прерываниями

Что же будет, если наше прикладная программа в основном хорошо соответствует системе с опросом, но содержит несколько задач, которые требуют непосредственного доступа к процессору? Необходимо ли отказываться от простой системы с опросом в пользу намного более сложного алгоритма планирования ОСРВ? Ответ будет отрицательным. Как мы указывали в главе 4, микроконтроллер 68HC12 имеет мощную, гибкую систему прерывания с приоритетным управлением. Чтобы обрабатывать простые текущие задачи может применяться алгоритм опроса, в то же время для внеочередной обработки задач требующих немедленного вмешательства может использоваться система прерываний.

Эти типы алгоритмов планирования известны также как системы с передним планом и фоном. Алгоритм опроса рассматривается как фоновая часть операционной системы, в то время как прерывание отвечает за приоритетную часть системы. Обычно, эти системы и разрабатываются по частям. Сначала разрабатывается, выполняется и отлаживается фоновая часть, а затем к ней добавляется часть приоритетного прерывания.

Пример: Обратимся опять к нашему контроллеру стереоусилителя. Транзисторные усилители, используемые в нашем стереопроекте достаточно дороги. В хорошем проекте транзисторы не перегреваются при обычной работе усилителя, чтобы предотвратить их повреждение. Предположим, что для защиты от перегрева проектировщик предусмотрел датчики температуры для каждого транзистора, чтобы постоянно контролировать их текущую температуру. Если температура любого транзистора достигает недопустимого уровня, проектировщик хотел бы, чтобы операционная система включала дополнительные вентиляторы охлаждения. Эта моди-

фикация проекта не требует, чтобы мы вообще отказались от схемы опроса. Лучше всего использовать схему опроса в фоновом режиме, чтобы обработать обычное рабочее управление стереоусилителем. В приоритетной части системы, датчики, контролирующие температуру транзисторов, могли бы быть подключены с помощью интерфейса к системе прерывания. Прерывание было бы активизировано при достижении температурой транзистора(ов) неприемлемо высокого уровня. В этом случае процессор должен был бы формировать сигнал, на включение дополнительного вентилятора. Мы исследуем этот проект в применениях, рассмотренных в разделе 8.9 и задании для самостоятельного исследования 2, приведенного в конце главы.

8.5.3. Карусельные системы

В карусельных ОСРВ системах, ядро последовательно переходит от обработки одной активной задачи к другой. Когда оно достигает последней задачи в системе, то начинает снова с первой задачи. В циклической системе, задачам часто позволяют работать до завершения прежде перейти к следующей задаче. Если это недопустимо для каждого прохода через все задачи, может использоваться метод квантования времени. При квантовании времени она каждую задачу цикла выделяется фиксированным временем доступа к процессору. Если за этот период задача не заканчивается, ее контекст сохраняется в связанном с ней ТСВ, она переводится в состояние готовности, а активной становится следующая задача в циклической последовательности. Карусельная операционная система может быть легко выполнена при использовании структуры данных с круговой очередью. В этом случае, каждый ТСВ мог бы быть связан с позицией в очереди. Операционная система последовательно переходила бы от одной задачи к другой внутри очереди. А если желательно было бы использовать квантование времени, то это легко было бы реализовать с помощью прерываний в режиме реального времени (RTI) имеющихся в 68HC12.

Пример: В предыдущей профессиональной жизни я (Стивен Ф. Баррет) был наводчиком в военно-воздушных силах. Я вместе с моим партнером отвечал за текущий контроль состояния своих ракет. Мы находились на расстоянии нескольких километров от ракет. Состояние ракеты проверялось по компьютеру в нашем центре управления. Компьютер проводил последовательный циклический опрос состояния каждой из наших ракет. Я не участвовал в проектировании компьютерной операционной системы центра управления; однако, я думаю, что использовалась карусельная система опроса, поскольку состояние каждой ракеты имело равную важность.

8.5.4. Смешанные системы

Как мы уже видели, карусельная система опроса может также быть дополнена возможностями приоритетного прерывания. Такой тип циклической системы с прерываниями называется смешанной системой. В этом случае, циклический алгоритм формирует фоновую часть из операционной системы, в то время как приоритетная часть системы формируется программой прерываний.

Пример: В сценарии управления ракетами, мы были бы заинтересованы в том, чтобы получить немедленное сообщение о катастрофических событиях, угрожающих одной из управляемых нами ракет, таких как затопление пусковой шахты, пожар в бункере ракеты, нарушение защиты, и т.д. В этих случаях операционная система могла бы быть дополнена частью программы с прерываниями, чтобы мгновенно перейти от обычного циклического опроса, к обработке событий с высоким приоритетом, а после ее окончания вернуться к обычному карусельному опросу.

8.5.5. Системы с управлением по прерыванию

В главе 4 мы обсуждали мощную и гибкую систему прерывания микроконтроллера 68HC12 с приоритетным управлением. Эту систему прерывания мы можем использовать теперь при разработке ОСРВ. При этом сначала основная программа проводит инициализацию, позволяющую конфигурировать систему, а затем система переходит к бесконечному циклу. Внутри цикла, процессор просто ждет событий, вызывающих прерывание. Когда приходят отдельные запросы на прерывания, процессор выполняет программу обработки прерывания, связанную с каждой из задач (с каждым прерыванием). Если несколько запросов на прерывания приходят одновременно, используются механизмы приоритетов, встроенные в 68HC12, и определяющие, какая из задач должна быть выполнена сначала. Механизмы стеков в системах прерывания 68HC12 гарантирует, что контексты задач будут правильно сохраняться и восстанавливаться. Операционную систему с управлением по прерыванию такого типа относительно просто написать. К числу ее преимуществ относится малое время реакции на событие, вызывающее прерывание. Систему необходимо спроектировать таким образом, чтобы она точно оценивала приоритет каждой задачи, от которой приходит запрос на прерывание.

Пример: В этой главе мы уже обсуждали робот, движущийся в лабиринте и обнаруживающий магнитные мины. Операционная система для этого робота могла бы быть выполнена, используя методы прерывания. В этом случае задачи инициализации (для ЖКД, ATD и ШИМ) были бы реализованы при инициировании операционной системы и затем переведены в бездействующее состояние. Операционная система затем перешла бы к непрерывному циклу, ожидая запросов на прерывание от различных задач. Каждая из задач, связанных с обнаружением мин, с обнаружением стенок лабиринта и управлением соответствующими поворотами, с модификацией информации, выводимой на ЖКД, посылала бы отдельный запрос на прерывание и вызывала связанную с ним программу обработки прерывания (ISR). Если, например, робот обнаруживает мину, это событие вызовет прерывание, и затем соответствующую ISR, связанную с обходом мины. Если несколько запросов на прерывания приходят однровременно; аппаратные средства приоритета прерывания 68HC12 обеспечивают первоочередное обслуживание прерывания с самым высоким приоритетом. Если запрос на прерывание с более высоким приоритетом приходит во время обслуживания низкоприоритетного прерывания, то последнее прерывается, контроллер 68HC12 конфигурируется для обслуживания прерывания с высоким приоритетом и лишь затем возвращается к обслуживанию низкоприоритетного прерывания.

8.5.6. Кооперативная многозадачность

При кооперативном многозадачном режиме ОСРВ для задачи с самым высоким приоритетом из находящихся в состоянии готовности выделяется определенное количество процессорного времени. Затем она возвращает управление операционной системе в отметке, удобной для прерывания связанных с ней действий. До передачи управления, эта задача модифицирует связанный с ней ТСВ в соответствии с текущим контекстом. Затем задача снова переводится в сосстояние готовности. Операционная система затем позволяет следующей залдаче с самым высоким приоритеом, находящейся в состоянии готовности перейти в активное состояние. Задача, вводящая активное состояние восстанавливает контекст, извлекая информацию из своего ТСВ. Эта задача затем выполняет свою часть программы, пока не достигает точки прерывания. Затем она, в свою очередь, модифицирует свой ТСВ в соответствии с текущим контекстом, возвращает управление операционной системе и процесс продолжается. Важно подчеркнуть, что в кооперативной многозадачной системе, задача возвращает управление операционной системе. Если задача не будет передавать управление, то задачи с более низким приоритетом никогда не смогут получить процессорного времени. Вы можете заметить, насколько важна межзадачная связь при таком типе операционной системы. Она обычно осуществляется путем использования глобальных переменных.

Операционные системы этого типа можно выполнить, используя для задач системы ряд списков с указателями. При этом списки с указателями существуют для каждого из различных состояний, в которых могут находиться задачи. Эта концепция иллюстрируется структурой, показанной на рис. 8.17. В процессе работы системы, задачи перемещаются из списка в список. Ранее в этой главе мы перемещали из одного списка с указателями в другой запись, идентифицирующую автомобиль, в ответ на его обслуживание различными автомобильными дилерами. Точно так же в зависимости от действий нашей прикладной программы перемещается из списка в список и задача, состояние которой отражается в форме записи с указателем.

8.5.7. Многозадачные системы с преимущественным приоритетом

Многозадачная система с преимущественным приоритетом (PPMS) очень похожа на кооперативную многозадачную систему, главное различие между ними заключается в том, какой из компонентов системы отвечает за предоставление процессорного времени. В кооперативной многозадачной системе, задача возвращает управление процессором операционной системе. В PPMS, сам процессор решает, какая из задач станет активной и будет ли выгружатьсязадача с низким приоритетом. С PPMS, каждой задаче внутри системы назначен приоритет. Операционная система исследует список с указателями, содержащий все задачи, находящиеся в состоянии готовности и переводит в активное состояние задачу с самым высоким приоритеом. Когда готова задача с более высоким приоритетом, операционная система затем выгружает низкоприоритетную задачу. Последняя м изменяет свой ТСВ и затем переходит в состояние готовности. Активной становится задача с высоким приоритетом. При этом типе ОСРВ, приоритеты задач могут быть постоянными или динамически изменяться в процессе выполнения программы.

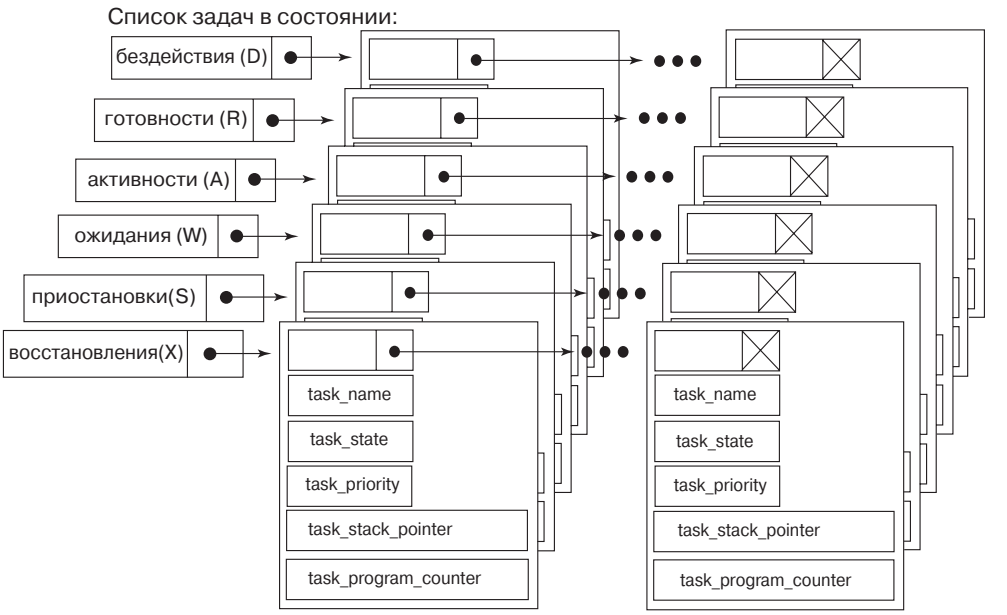


Рис. 8.17. *Система списов с указателями для задач*

Завершая наше обсуждение различных типов ОСРВ, мы должны подчеркнуть, что ни один из способов построения ОСРВ не лучше, чем другие. Проектировщик системы, должен выбрать наиболее подходящую для конкретной разработки конфигурацию ОСРВ в соответствии с требованиями приладного устройства. В следующем разделе мы обсудим методы, применяемые при разработке ОСРВ.

8.6. Проблемы ОСРВ

После обсуждения теории и принципов работы ОСРВ в предыдущих разделах, вы, наверное, оценили эти системы по достоинству. Система ОСРВ решает много проблем но создает также ряд трудностей. В этом разделе мы исследуем некоторые из проблем, с которыми вы можете столкнуться во встроенных системах ОСРВ. Мы также обсуждаем методы, с помощью которых можно избежать этих ситуаций, либо смягчить или исправить их. Мы исследуем проблемы конкуренции, повторной входимости, межзадачной связи и безопасности, и коснемся, наконец, главного вопроса: создавать ли вам собственную ОСРВ или покупать стандартную.

8.6.1. Конкуренция

Другой рассказ

Я провожу много времени в поездках. Однажды я проезжал от Ларами до Форта Коллинз, где должен был подготовить конференцию. На прекрасном и живописном шоссе U.S. 287, мне встретился протяженный участок ремонтируемой дороги. Эта магистраль – асфальтированная дорога с двумя встречными полосами.

Одна полоса магистрали была закрыта на ремонт, по другой можно было проехать. Регулировщики с флажками стояли на обоих концах свободной полосы, чтобы поочередно пропускать группу автомобилей, едущих с юга, а затем – группу, едущих с севера. Регулировщики поддерживали связь друг с другом по радио. Чтобы поддерживать движение и избежать столкновений, один регулировщик должен был дать знак остановки приближающемуся потоку, в то время как другой должен был дать знак встречному потоку «осторожно продолжайте движение».

Этот рассказ иллюстрирует проблему конкуренции, которая присутствует в некоторых типах ОСРВ. В этом случае мы имеем критический ресурс (одну дорогу для встречных потоков), который одновременно требуется для использования двумя различными задачами (пропуском северного и южного потоков). Задача двух регулировщиков состоит в том, чтобы предотвратить одновременный доступ двух задач к критическому ресурсу, не допустив столкновения. В этом разделе мы исследуем, как распределяется критический ресурс в ОСРВ и как можно при этом избежать столкновений.

Как мы уже установили, ОСРВ должна позволить многим задачам, конкурирующим за один и тот же ресурс – процессорное время, завершить свои действия. Мы исследовали достаточно много алгоритмов планирования, позволяющих реализовать эту задачу. Некоторые алгоритмы разрешали межзадачные переключения, только в четко определенные моменты (карусельный и кооперативный многозадачные режимы), другие же – в частности, использующие механизмы прерывания, или приоритетного планирование – могли осуществлять межзадачное переключение без предупреждения.

Проблема конкуренции появляется, когда задача с низким приоритетом выполняет некоторую критическую часть своего кода или использует критический ресурс. Если она будет прервана, то не сможет корректно завершить свою программу, даже если впоследствии и получит необходимое процессорное время. Например, представим себе, что задача использует ATD микроконтроллера 68HC12, выполняя аналого-цифровое преобразование. Если задача начнет преобразование и будет прервана до его окончания, то произойдут ошибки. Эта ситуация может стать еще более сложной, если ставшая активной высокоприоритетная задача также должна использовать модуль ATD.

Обычно, мы называем такую часть программного кода или ресурса критической. Правильное решение проблемы конкуренции должно в первую очередь предотвратить подобные случаи. Чтобы предотвратить одновременный доступ к критическим ресурсам может использоваться ряд методов, включая следующие:

- *Запрет на прерывания:* Прерывания блокируются, когда задача выполняет критическую часть программного кода. В микроконтроллере 68HC12 это выполняется с помощью команды SEI (установить маскирование прерывания). Как только критическая часть кода закончена, прерывания могут быть снова разрешены командой CLI (очистить маскирование прерывания).
- *Использование семафоров или блокировок;* семафор или блокировка может использоваться, чтобы указать, что критический ресурс не доступен для использования, потому что это используется в настоящее время другой задачей. Регулировщики с флажками из нашего рассказа как раз и использовали семафоры (знаки «остановиться» и «продолжать осторожное движение») чтобы запрещать или разрешать доступ к критическому ресурсу. Прежде, чем часть программы сможет использовать критическую часть кода, она должна выяснить, доступен ли ресурс.

8.6.2. Повторная входимость

Тесно связана с концепцией конкуренции проблема повторных входов. Функция считается повторно используемой, если она всегда работает правильно и сохраняет данные даже после прерывания и перезапуска. И снова возникают проблемы, когда задача с низким приоритетом прерывается прежде, чем завершает свою работу функция. Если более высокоприоритетная задача использует ту же самую функцию, функция перезапустится прежде, чем она закончит задачу с более низким приоритетом.

Следующие методы могут использоваться, чтобы создать повторно используемую функцию:

- *Запрет на прерывания:* Прерывания заблокированы при выполнении критических частей некоторой функции.
- *Использование локальных переменных:* Вспомним, что локальные переменные хранятся в стеках. Если высокоприоритетная задача выгружает задачу с низким приоритетом, переменные безопасно сохранены в стеке.
- *Использование регистров микропроцессора:* регистры микропроцессора могут использоваться, чтобы сохранить критические переменные внутри повторно используемой функции. Если функция прервана, переменные автоматически сохраняются в стеке микропроцессора.
- *Комбинация методов:* чтобы создать повторно используемую функцию, может использоваться комбинация этих методов.

8.6.3. Межзадачные связи

Межзадачная связь – ключевое требование для ОСРВ, использующих прерывания. Эта – проблема не критична для кооперативных многозадачных ОСРВ поскольку сами задачи определяют, когда можно вернуть управление операционной системе. Следовательно, действия задачи могут гарантировать, что все данные были правильно модифицированы перед отказом задачи от управления.

Проще всего использовать для пересылки данных между задачами глобальные переменные. Как мы видели в предыдущем обсуждении, эта простая методика может нарушаться, если низкоприоритетная задача выгружается задачей с высоким приоритетом прежде, чем получит возможность, чтобы модифицировать эти переменные.

Чтобы предотвращать такие случаи, мы можем использовать почтовый ящик – взаимно согласованное расположение в памяти, которую задачи могут совместно использовать. Почтовый ящик может содержать одну переменную, группу переменных или даже структуру данных. Чтобы гарантировать, что, данные в почтовом ящике являются текущими, задача, может выполнять почтовую операцию – операцию, которая может записывать в почтовый ящик информацию, защищенную шифром, и другая задача может затем считать содержимое почтового ящика (выполнить операцию pend), если имеет доступ к шифру. Шифром в каждый момент обладает только одна задача, и только она может работать с почтовым ящиком в это время. Это очень похоже на обсужденную ранее методику семафоров.

8.6.4. Безопасность, проверка и безотказная работа

Одной из наиболее критичных проблем, усложняющих работу ОСРВ, является проблема безопасности. Мы уже обсудили часть стандартов, касающихся измерений и безопасности. В зависимости от того, где используется ОСРВ (связь, медицинское изделие, коммерческое изделие, авиация, и т.д.), имеются различные требования безопасности, которые должны быть выполнены. Безопасность операционной системы должна быть подтверждена документацией и испытаниями. Кроме того, в случае сбоя, система должна переходить в безопасный режим – предотвращающий травматизм персонала и повреждение оборудования.

Пример: Если мы должны были использовать ОСРВ, чтобы управлять светофором на оживленном перекрестке, безопасным режимом при любых сбоях системы будет красный свет для обоих направлений движения. Нетрудно представить, что выбор для безопасного режима зеленого света сразу приведет к катастрофическим последствиям.

Проблема безопасности – ключевой фактор при решении вопроса: написать ли вам собственную операционную систему или приобрести стандартную.

8.6.5. Главный вопрос

При нашем обсуждении операционных систем в режиме реального времени, мы обходили ключевой вопрос, который вы могли бы задать: следует написать свою собственную систему или приобрести стандартную? Мы полагаем, что это решение этой проблемы остается за вами – проектировщиком системы. Чтобы помочь вам принять это решение, мы рекомендуем учесть следующие три фактора:

- ***Применение:*** Для чего система будет использоваться? Если это используется в системе, где безопасность – критичной проблемой, разумно использовать систему, сертифицированную для безопасного использования. Это не подразумевает высокой стоимости. Пакеты ОСРВ для безопасной работы могут быть и дешевыми [Labrosse 2002].
- ***Критичность:*** уважаемый коллега по вычислительной технике сознательно выбрал, собственную разработку, а не применение стандартного пакета ОСРВ при разработке критичного применения ОСРВ, усложненного требованиями к ядерной безопасности. Он мотивировал это тем, что он хотел написать каждую строку этой программы и полностью понимать работу системы. Он чувствовал, что в коммерческом пакете некоторые подробности системы будут от него скрыты.
- ***Стоимость:*** Не следует думать, что разработка собственного ОСРВ может принести вам большую экономию, поскольку стоимость программ в 5 000 долларов может показаться слишком высокой. Однако при создании собственной программы ваши трудозатраты могут стоить не меньше.

Какой бы путь вы не выбрали, вам будут полезны некоторые рекомендации следующего раздела, которым необходимо следовать при разработке ОСРВ.

8.7. Выполнение операционной системы реального времени

Мы приводим здесь краткое описание шагов, необходимых для создания реализации выполнения операционной системы в режиме реального времени ОСРВ:

1. Полностью понять все системные требования. Определить, действительно ли вам требуется ОСРВ. Если вы разрабатываете простой алгоритм управления, то вас может удовлетворить обычная последовательная программа; ОСРВ необходима только тогда, когда ваша прикладная программа должна управлять целым рядом событий.
2. Определить соответствующий алгоритм планирования, который целесообразно использовать для управления задачами. Не забудьте, каждый из них не лучше другого. Вы, как разработчик системы, задание должны найти алгоритм планирования, наиболее удовлетворяющий применению.
3. Разделить вашу систему на независимые задачи.
4. Выполнить операционную систему, основная функция которой заключается в планировании и управлении задачами.
5. Написать код задач. Убедитесь, что задачи не конкурируют за одни и те же участки памяти и что они не зависят друг от друга. Убедитесь также, что реализация задачи соответствует принятому алгоритму планирования.
6. Если вы разрабатываете систему с передним планом и фоновыми задачами, разработайте сначала фоновую, а затем приоритетную часть.
7. И что важнее всего, проверяйте, проверяйте и проверяйте! Используйте звуковые методики испытаний, которые были обсуждены в главе 2.

8.8. Пример применения: ОСРВ циклического опроса

В этом разделе мы обсудим различные операционные системы в режиме реального времени. Мы начнем с базовой системы циклического опроса, а затем рассмотрим систему циклического опроса с прерываниями. Затем мы опишем аппаратный имитатор, предназначенный для разработки и проверки более сложных алгоритмов планирования.

8.8.1. Краткий обзор проекта

Мы исследуем здесь базовую систему циклического опроса, используемую для управления стереоусилителем. Мы рассматривали уже такой усилитель в главе 2, когда обсуждали проектирование систем и затем в данной главе в качестве примера системы циклического опроса. Рис. 8.18 представлен общий вид системы усилителя. Шесть переключателей на передней панели блока или на пульте дистанционного управления используются, чтобы выбрать один из шести источников звукового сигнала. На усилитель в любой момент времени подается сигнал только с одного источника.

8.8.2. Пример кода

Приведенный код, который используется, чтобы управлять усилителем, состоит из кода инициализации и цикла опроса. Цикл опроса непрерывно проверяет изменение в состоянии переключателей на лицевой панели блока (PORTB) или дистанционном управлении (PORTA). Алгоритм управления UML приведен на рис.8.19.

```
//file name:   ampl2.c
//function:   program provides   control   of   amplifier
//target   controller:   Motorola  68HC912B32   evaluation board   {EVB]
//      -    32    K  Flash EEPROM available    at    $8000
//      -    Compiler  options:
//      -    Program Memory:    0x8000
//      -    DataMemory:         0x0800
//      -    Stack    Pointer:  0x09FF
//
// Эта программа обеспечивает управление звуковым усилителем.
// Усилитель может принимать звуковой сигнал от ряда
// источников. Пользователь может выбирать источник сигнала
// для усиления с помощью переключателей на лицевой панели
// (связанных с портом B), либо  переключателей на пульте
// дистанционного управления (связанных с портом A). Процессор
// управляет светодиодами на передней панели(связанными с портом P)
// и показывающими активный источник сигнала и включает реле(связанные
// с портом T), подсоединяющие один из источников сигнала к усилителю
//
```

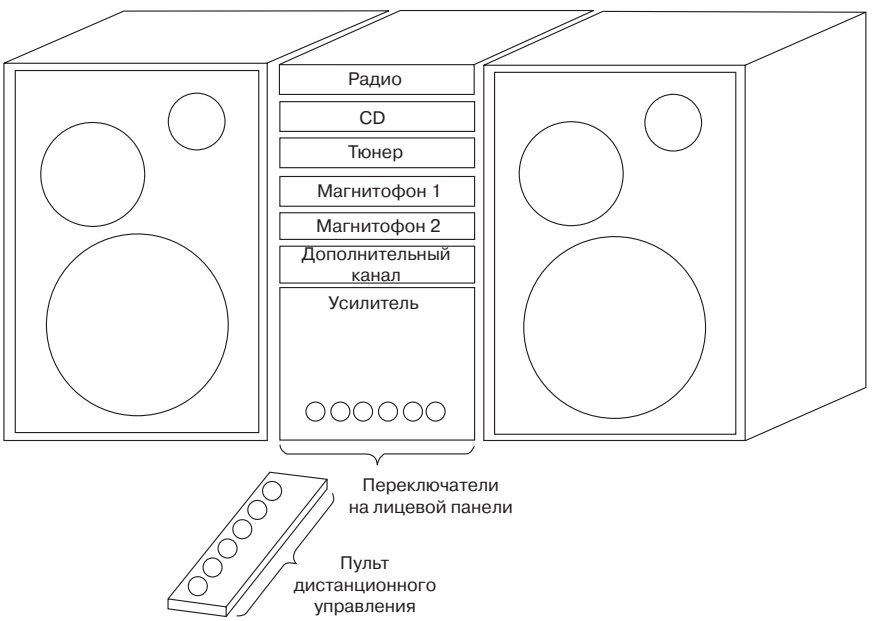


Рис. 8.18. *Краткий обзор усилителя*

```
// Функции портов ввода
//
// Порт A, входной - вводит сигналы от пульта дистанционного управления,
// требует импульсов высокого логического уровня длительностью в 100 мс
```

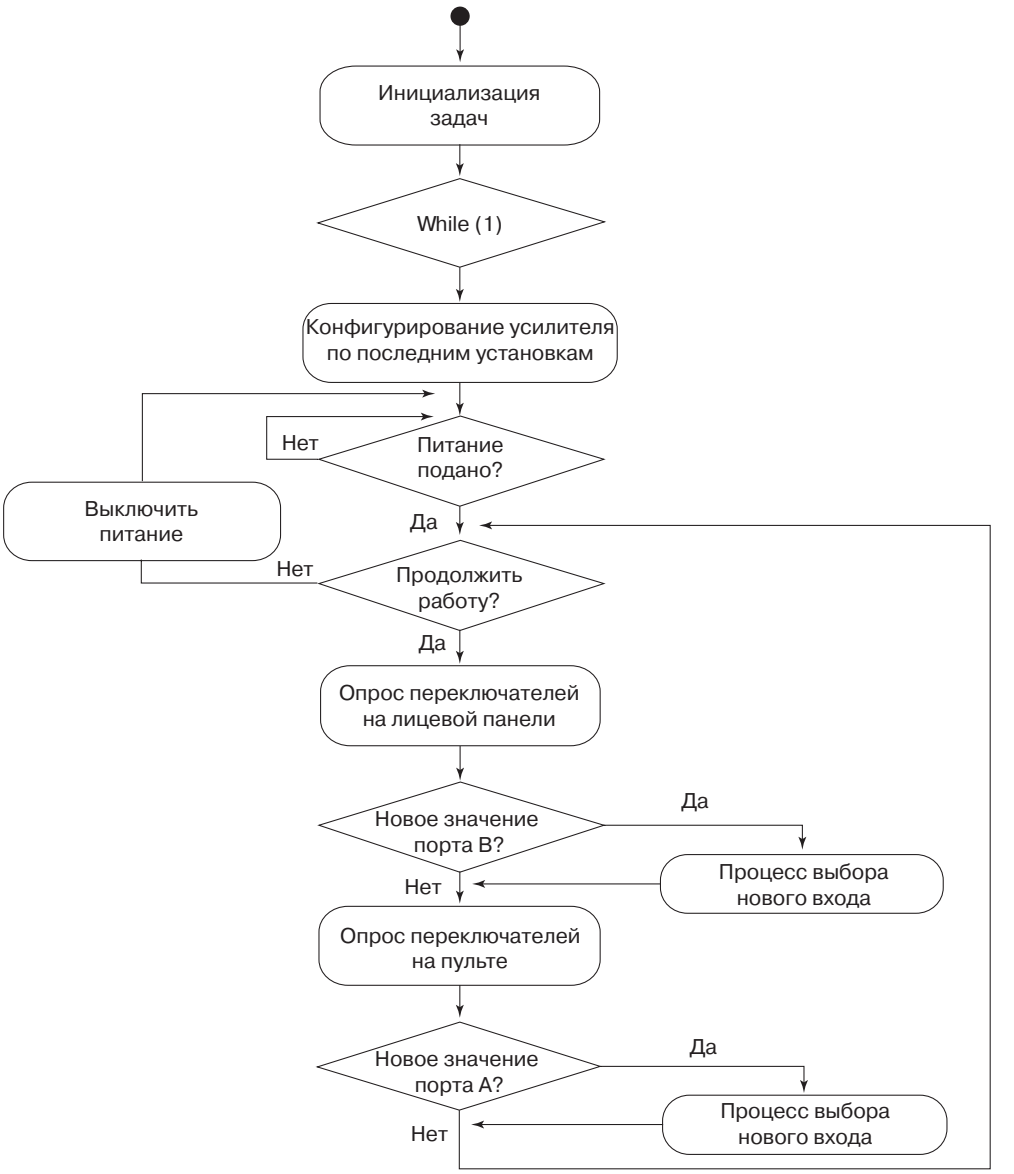


Рис. 8.19. *Алгоритм программы UML для усилителя*

```
// PA7 выкл. звука  от пульта дист. управления высокий - импульс 100 мс
// PA6 Дополнителльный канал (ДК)  от пульта дист. управления высокий
// импульс 100 мс
// PA5 магнитофон # 2 от пульта дист. управления высокий - импульс 100 мс
// PA4 магнитофон # 1 от пульта дист. управления высокий - импульс 100 мс
// PA3 тюнер от пульта дист. управления высокий - импульс 100 мс
// PA2 CD от пульта дист. управления высокий - импульс 100 мс
// PA1 пианино от пульта дист. управления высокий - импульс 100 мс
// PA0 предусилитель от пульта дист. управления высокий - импульс 100 мс

// Порт B входной - от переключателей на  лицевой панели блока

// PB0 предусилитель от переключателя на лицевой панели, вжатый перек-
// лючатель = вкл
// PB1 пианино от переключателя на лицевой панели, вжатый переключатель = вкл
// PB2 CD от переключателя на лицевой панели, вжатый переключатель = вкл
// PB3 тюнер от переключателя на лицевой панели, вжатый переключатель = вкл
// PB4 магнитофон # 1  от переключателя на лицевой панели, вжатый пе-
// реключатель = вкл
// PB5 магнитофон # 2  от переключателя на лицевой панели, вжатый пе-
// реключатель = вкл
// PB6 ДК от переключателя на лицевой панели, вжатый переключатель = вкл
// PB7 выкл. звука  от переключателя на лицевой панели, вжатый перек-
// лючатель = вкл
//
//Порт P выходной  - светодиоды на лицевой панели

//PP0 сигнал на силовое реле и на светодиоды и сигнал низкого уровня
//для //светодиодов в буфер
//PP1 светодиод пианино выходной низкопотенциальный сигнал - 10 мА
//PP2 светодиод CD выходной низкопотенциальный сигнал - 10 мА
//PP3 светодиод тюнер выходной низкопотенциальный сигнал - 10 мА
//PP4 светодиод магнитофон # 1 выходной низкопотенциальный сигнал - 10 мА
//PP5 светодиод магнитофон # 2 выходной низкопотенциальный сигнал - 10 мА
//PP6 светодиод ДК выходной низкопотенциальный сигнал - 10 мА
//PP7 светодиод выкл. звука, сигнал на силовое реле
//
//Порт T выходной - драйверы реле

//PT0 реле RESET выход на реле RESET высокий  уровень - импульс 5 мс
//PTl реле пианино выход на реле пианино высокий  уровень - импульс 5 мс
//PT2 реле CD  выход на реле CD высокий  уровень - импульс 5 мс
//PT3 реле тюнера выход на реле тюнера высокий  уровень - импульс 5 мс
//PT4 реле магнитофон # 1 выход на реле магнитофон # 1 высокий  уро-
//вень - импульс 5 мс
//PT4 реле магнитофон # 2 выход на реле магнитофон # 2 высокий  уро-
//вень - импульс 5 мс
//PT6 реле ДК выход на реле ДК   высокий  уровень //- импульс 5 мс
//PT7 высокий  уровень - импульс 10 мс для подачи питания на оптроны
//светодиодов и усилитель
//Подача питания  (от сети или от источника 5 В):

//Крнфигурация портов:
//1. Порт A: конфигурирован как входной, отжатый переключатель - запрет
//2. Порт B: конфигурирован как входной, отжатый переключатель - разрешение
```

```
//3. Порт P: конфигурирован как выходной, все линии в 1
//4. Порт T: конфигурирован как выходной, все линии в 0
//5. Установка "RELAY-RESET" (PTO) импульсом высокого состояния 5 мс
//6. Установка "RELAY-CD" (PT2) импульсом высокого состояния 5 мс
//7.  Установка "WHICH-INPUT" позиция сохранения = "CD"
//8. Цикл PPl-PP6 (устанавливаются в низкое состояние) светодиоды
//   показывают , что контроллер  работает
//9.  Переход к последовательности  "PREAMP ON"
//
//Логика работы :
//Последовательность "PREAMP ON"
//1. Ожидание установки "S-PREAMP-PWR" (PB0) или "R-PREAMP-PWR" (PAO)
//2. Установка "LED-MUTE-RELAY" (PP7)
//3. Установка "LED-PWR-RELAY" (PP0)
//4. Считывание  позиции в "WHICH-INPUT"
//5. Установка "LED-xxxxx" = позиция "WHICH-INPUT"
//6. Установка PT7(1) импульсом  10 мс
//7. DE-Assert "LED-MUTE-RELAY" (PP7) через ~3 с.
//8. переход к режиму "SCAN"
//
//Последовательность "SCAN"
//1.Ожидание входного сигнала от (PB0-PB7) или от (PA0-PA7)
//2. IF = "x-PREAMP-PWR" - переход к последовательности "PREAMP OFF"
//3. IF = "x-MUTE" GOTO - переход к последовательности "MUTE"
//4. IF = любой входной сигнал от  (PB1-PB6) или (PAl-PAб)- переход к
"CHANGE"
// последовательность "INPUT"
//
// последовательность "CHANGE INPUT":
//1. Включить "LED-MUTE-RELAY" (PP7)
//2. Включить "RELAY-RESET" (PT0) импульсом высокого уровня 5 мс
//3. Включить "RELAY-xxxxx" (PT1-PT6)  (в соответствии с выбором
//  "WHICH-INPUT" импульсом высокого уровня 5 мс)
//4. Включить  "LED-xxxxx" (PP1-PP6) (в соответствии с выбором
//  "WHICH-INPUT")
//5. Очистить  Old/Input сохранить новое значение "WHICH-INPUT"
//6. DE-переключение "MUTE-RELAY" (PP7) примерно через 3 с.
//7. Перейти к последовательности  "SCAN"
//
// последовательность "MUTE":
//1. Переключить "LED-MUTE-RELAY" (PP7)
//2. Перейти к последовательности  "SCAN"
//
//Последовательность "PREAMP OFF":
//1. Включить "LED-MUTE-RELAY" {PP7)
//2. DE-переключение "LED-PWR-RELAY" (PPO)
//3. DE-переключение всех светодиодов (PPl-PP6)
//4. Включить  PT7(1) импульсом  10 мс
//5. DE-переключение "LED-MUTE-RELAY" (PP7) примерно через 3 с.
//6. Перейти к последовательности "PREAMP ON"
//
//авторы: Steven Barrett и Daniel Pack
//Дата разработки: 19 июня 2004
//Последняя редакция: 20 июня 2004
//*****************************************************************
```

```
//*****************************************************************
//включенные файлы
#include<912b32.h>  //B32 EVB header file
#include"func_def.h"        //функции-прототипы, глобальные переменные

//main program*****************************************************

// глобальные переменные
int       which_input;                  //вход усилителя
int       keep_going;                   //ввод переменных
int       mute;              //флаг управления выключением звука
unsigned char   old_PORTB = Qxff;       //текущие значения PORTB
unsigned char   old_PORTA = 0x00;       //текущие значения PORTA
unsigned char  new_PORTB, new_PORTA;    //новые значения PORTA, PORTB

void main(void)
{
asm(" .area vectors(abs)\n"
" .org 0xFFF8\n"             //инициализация вектора сброса для 68HC12 B32

" .word 0x8000, 0x8 00 0, 0x8000, 0x8000\n" " .text");
initialize_task();
                             //главный цикл
while(1){                    //ожидается сигнал на включение питания
  if ((PORTB==0xFE)||(PORTA==0X01))
                             //PORTB переключается в низкое, PORTA - в
                             // высокое состояние
{            //вы забыли включит питание! Запрос на операцию включения
keep_going = 1;              //цикл считывания переменных
PORTP=0x7E;                  //включение LED-MUTE-RELAY PP7(0)
                             //LED-PWR-RELAY PPO(0)  (0111_1110)
which_input_task();
activate_power_relay_task();
delay_3s();                  //задержка 3 с.

PORTP  = 0x80;               // DE-переключение PD7(1) - включение звука
while(keep_going)            //прохождение меню - главный цикл опроса
          {
          process_PORTB_input_task
          process_PORTA_input_task
          }

      }//end if - ожидание включения питания - питание не подано!
   }//end while(1)
}//конец главного цикла
//*****************************************************************
// определение функций
//*****************************************************************

initialize_task: начальные установки усилителя
//*****************************************************************

void initialize_task(void)
```

```
{
mute = on; //turn mute on
initialize_timer();          // инициализация таймера
initialize_ports();          // инициализация портов
initialize_pins();           // инициализация состояния отдельных выводов
which_input = 2 ;            //по умолчанию включается вход  CD(2)
                             //включение светодиодов на лицевой панели
PORTP = 0x81; //включение всех светодиодов PD1-PD6 низким активным
уровнем (1000_0001)
delay_3s();   //задержка 3 с
PORTP = Oxff; //выключение светодиодов
}
//*********************************************************************
//which_input_task: опрос входов, установка текущего состояния
//*********************************************************************

void which_input_task(void)
{

switch(which_input){ // подсвечивается светодиод для используемого
                     // входа (по умолчанию вход 2 - CD )

  case 1: //Пианино
    phono_task();
           break;

  case 2: //CD
    CD_task(>;
           break;

  case 3: //Тюнер
    tuner_task();
           break;

  case 4: //Магнитофон 1
    tapel_task();
          break;

  case 5: //Магнитофон 2
    tape2_task();
          break;

  case 6: //Дополнительный канал (ДК)
    aux_task();
          break;

  default:;
  }//конец switch
}
//*********************************************************************
//phono_task: конфигурируется вход от Радио
//*********************************************************************

void phono_task(void)
{
```

```
PORTT |= 0x02;  //устанавливается  PT1(1) (0000_0010)
delay_5ms();

PORTT &= -0x02; // выключается PT1(0)
PORTP  - 0x7E;  //гасятся все светодиоды
PORTP   &=   -0x02; //включается светодиод 1 (0)
}
//*********************************************************************
//CD_task: конфигурируется вход от CD
//*********************************************************************

void CD__task (void)
{
//CD
PORTT |= 0x04;       // устанавливается  PT2(1) (0000_0100)
delay_5ms();
PORTT &= -0x04;      // выключается PT2(0)
PORTP |= 0x7 E;      //гасятся все светодиоды
PORTP &= -0x04;      // включается светодиод 2 (0)
}
//*********************************************************************
//tuner_task: конфигурируется вход от тюнера
//*********************************************************************

void tuner_task(void)
{
//TUNER PORTT    |=   0x08; // устанавливается PT3(1) (0000_1000)
delay_5ms () ;
PORTT   & =   0x08;  // выключается PT3(0
PORTP    |=   0x7E;) //гасятся все светодиоды
PORTP   Sc~   -0x08; // включается светодиод 3 (0)
}
//*********************************************************************
//tapel_task: конфигурируется вход от магнитофона 1
//*********************************************************************

void tapel_task(void)
{
//TAPE#1
PORTT |= 0x10;       //assert PT4(1) (0001_0000)
delay_5ms();
PORTT &= -0x10;      // выключается PT4(0)
PORTP |= 0x7E;       //гасятся все светодиоды
PORTP &= -0x10;      // включается светодиод 4 (0)
}
//*********************************************************************
//tape2_task: конфигурируется вход от магнитофона 2
//*********************************************************************

void tape2__task(void) {
//TAPE#2
PORTT |= 0x2 0;   // устанавливается PT5(1> (0010_0000)
delay_5ms();
PORTT &= -0x20;   // выключается PT5(0)
PORTP |= 0x7E;    //гасятся все светодиоды
```

```
PORTP & = -0x2 0; // включается светодиод 5 (0)
}

//*****************************************************************
//aux_task: конфигурируется вход от дополнительного канала
//*****************************************************************
{
void aux_task(void)
//ДК
PORTT |= 0x40;       // устанавливается PT6(1) (O100_0000)
delay_5ms();
PORTT &= -0x40;      // выключается PT6(0)
PORTP |= 0x7E;       //гасятся все светодиоды
PORTP &= -0x40;      // включается светодиод 6(0)
}
//*****************************************************************
//activate_power_relay_task(): включается реле силового питания
//*****************************************************************

void activate_power_relay_task(void){
PORTT |= 0x8 0;      // устанавливается PT7(1) импульсом 10 мс
delay_5ms();
delay_5ms();
PORTT &= -0X80;      // выключается PT7
}
//*****************************************************************
//process_PORTB_input_task(): определяется выбранный вход от  PORTB
//*****************************************************************

void process_PORTB_input__task (void){
new_PORTB = PORTB; //read PORTB
if(new_PORTB != old_PORTB){  //считывание состояния порта  PORTB
  switch(new_P0RTB){         //PORTB устанавливается на низкий уровень

        case OxFE: //PB0 "S-PREAMP-PWR" (llll_1110)

                   if(process_valid_input_PORTB(new_PORTB))
                     {
                     preamp_off();
                     keep_going=0;
                     }
                   break;

        case 0xFD:                    //PB1 "S-PHONO" (llll_1101)
                    if(which_input !=1){
                    if(process_valid_input_PORTB(new_PORTB)
                      }
                      which_input = 1;
                      change_input();
                      }}
                    break;

        case OxFB:                     //PB2 "S-CD" (llll_1011)

                    if{which_input!=2) {
```

```
                    if(process_valid_input_PORTB(new_PORTB))
                      }
                      which_input = 2; change_input();
                      }}
                    break;

        case 0xF7: //PB3 "S-TUNER" (llll_0111)
                    if (which__input ! -3 ) {
                    if(process_valid_input_PORTB(new_PORTB))
                     {
                     which_input = 3;
                     change__input() ;
                      }}
                    break;

        case OxEF: //PB4 "S-TAPE#1" (1110_llll)
                    if (which__input ! -4) {
                    if(process_valid_input_PORTB(new_PORTB))
                      {
                      which_input = 4;
                      change__input () ;
                      }}
                    break;

        case OxDF: //PB5 "S-TAPE#2" (1101_llll)
                    if (which__input ! =5) {
                    if(process_valid_input_PORTB(new_PORTB))
                      {
                      which_input = 5;

                      change_input();
                      }}
                      break;

        case OxBF: //PB6 "S-AUX" (1011_llll)
                    if(which_input !=6){
                    if(process_valid_input_PORTB(new_PORTB))
                      {
                      which_input - 6;
                      change_input();
                      }}
                    break;

        case 0x7F: //PB7 "S-MUTE" (0111_llll)
                    if (process_valid_input_PORTB(new_PORTB))
                      {
                      mute_toggl^();
                      }
                    break;
        default:;  //all other cases
        }  //конец switch(new_PORTB)
  }       //конец if new_PORTB
  old_PORTB=new_PORTB; //update PORTB
}
```

```
//*****************************************************************
//process_PORTA_input_task():определяется выбранный вход от  PORTA
//*****************************************************************

void process_PORTA_input_task(void)
{
new_PORTA = PORTA; //Читать PORTA
if(new_PORTA != old_PORTA){  //выбор входа по состоянию порта PORTA
 switch(new_PORTA){          //PORTA переводится в высокое состояние
        case 0x01:           //PAO "R-PREAMP-PWR" (0000_0001)
                    if(process_valid_input_PORTA(new_PORTA))
                      {
                      preamp_off();
                      keep_going-0;
                      }
                    break;

        case 0x02: //PA1 R-PHONO" (0000_0010)
                    if(which_input !=1) {
                    if(process_valid_input_PORTA(new_PORTA))
                      {
                      which_inp"ut = 1;

                      change_input();
                      }}
                    break;

        case 0x04:          //PA2 "R-CD" (0000_0100)
                    if(which_input ! -2) {
                    if(process_valid_input_PORTA(new_PORTA))
                      {
                      which_input - 2;
                      change_input();
                      }}
                    break;
        case 0x08:       //PA3 "R-TUNER" (0000_1000)
                    if(which_input !=3){
                    if (process_valid_input_PORTA(new_PORTA))
                      {
                      which_input - 3;
                      change_input();
                      })
                    break;

        case 0x10:      //PA4 "R-TAPE#1" (0001_0000)
                    if(which_input !-4){
                    if fprocess_valid_input_PORTA(new_PORTA))
                      {
                      which_input = 4;
                      change_input();
                      }}
                    break;

        case 0x20: //PA5 "R-TAPE#2M (0010_0000)
                    if(which_input !=5){
```

```
                    if(process_valid_input_PORTA(new_PORTA))
                      {
                      which_input = 5;
                      change_input();
                      }}
                    break;

        case 0x40: //РАб "R-ДОПОЛНИТЕЛЬНЫЙ КАНАЛ" (0100_0000)
                    if(which_input !=6){
                    if(process_valid_input_PORTA(new_PORTA))
                      {
                      which_input - 6;
                      change_input(J;
                      }}
                    break;

        case 0x80:       //PA7 "R-MUTE" (1000_0000)
                    if (process_valid_input_PORTA(new_PORTA) )
                      {
                      mute_toggle();
                      }
                    break;
        default:; //all other cases
} //конец switch(new_P0RTA)
  }//конец if new_P0RTA
old__PORTA=new_PORTA;                    //изменяется состояние  PORTA
}

//*****************************************************************
//initialize_timer:установка частоты таймера обслуживающего счетчик
//*****************************************************************

void initialize_timer(void)
{
TMSK2 = 0x05; //установка на  250 КГц
TSCR =  0x80; //разрешение работы таймера
}

//*****************************************************************
//initialize_ports: начальная конфигунрация портов
//*****************************************************************

void initialize_ports(void)
{
DDRA=0x00;  //конфигурация PORTA в качестве входного
PORTA=0x00; //запрет на подключение подтягивающих резисторов в  PORTA
DDRB=0x00;  //конфигурация PORTB в качестве входного
PORTB=0xff; //разрешение  подключения подтягивающих резисторов в  PORTB
DDRT=0xff;  // конфигурация PORTT в качестве выходного
PORTT=0x00; // установка на низкий уровень
DDRP=0xff;  // конфигурация PORTD в качестве выходного
PORTP=0xff  // установка на высокий уровень
}

//*****************************************************************
```

```
//*****************************************************************
//initialize_pins: установка отдельных выводов
//*****************************************************************

(void initialize_poins(void)
{
PORTT=0x01;                  //сброс реле PT0(1) 5 мс импульс с
                             // активным уровнем (0000_0001)
//delay_5ms():

PORTT=0x00;
}

//*****************************************************************
//delay_5ms: Задержка на 5 мс сформированная из базе частоты таймера
//в 250 кГц
//*****************************************************************

void delay_5ms(void)
{
int i;

for(i=0; i<1250;
  asm("nop");                //требуется только один импульс таймера
}

//*****************************************************************
//delay_3s: Задержка на 3 с
//*****************************************************************

void delay_3s(void)
{
int i;

for(i=0;i<600;i++)
  delay_5ms();
}

//*****************************************************************
//change_input: изменение активного входа
//*****************************************************************

void change_input(void)
{
PORTP   &=   -0x80; //установка LED-MUTE-RELAY PP7(0) 1000_0000
PORTT    |-   0x01; //установка сброса  реле PT0(1) 5 мс
delay_5ms();
PORTT   &=   -0X01; //turn off PT0
switch(which_input)
  {
  case 1: //PHONO
           phono_task();
           break;

  case 2: //CD
```

```
          CD_task();
          break;

  case 3: //TUNER
          tuner_task();
          break;

  case 4: //TAPE#1
          tapel_task();
          break;

  case 5: //TAPE#2
          tape2_task();
          break;

  case 6: //AUX
          aux_task();
          break;
  default:;//все другие входы
  }//конец switch
  delay _3s () ;
  PORTP |= 0x80;            //сброс LED-MUTE-RELAY PP7(1)
 }

//*********************************************************************
//mute_toggle: включение и выключение звука
//*********************************************************************

void mute_toggle(void)
{
if (nmte ==off) {
  PORTP &= -0x80;           //установка  LED-MUTE-RELAY PP7(0)
  mute = on;}
else{
  PORTP |= 0x80;            // сброс LED-MUTE-RELAY PP7(1)
  mute = off;} }
}//end mute_toggle

//*********************************************************************
//preamp_off: turn amplifier off
//*********************************************************************

void preamp_off(void)
{
PORTP &= -0x80;      //установка LED-MUTE-RELAY PP7(0)
PORTP |= 0x01;       //сброс LED-PWR-RELAY PP0(1)
PORTP |= 0x7e;       //сброс  светодиодов PP1-PP6(1)(0111_1110)
                     //установка  PT7 импульсом 10 мс
PORTT |- 0x80;       //установка PT7(1) импульсом 10 мс
delay_5ms();
delay_5ms();
PORTT &= -0X80;      //сброс PT7
delay_3s();
PORTP  = 0x80;       //сброс PP7(1) LED-MUTE-RELAY
```

```
keep__going=0;
}

//*****************************************************************
//process_valid_input_PORTA: проверка состояния пульта дистанционного
//управления, длительностью не менее  50 мс
//*****************************************************************


int process_valid_input_PORTA(unsigned char portx)
{
int  valid_input;            //установить флаг ошибочного входа

unsigned int current_count;

valid_input = TRUE;
current_count = TCNT;        // задать текущее состояние
while(TCNT < (current_count+12500)){   //отследить активный вход за 50 мс
if(portx==PORTA) valid_input = TRUE; else valid_input = FALSE;
if f!valid_input) break; //цикл while
}//end while
return valid_input;
}

//*****************************************************************
//process_valid_input_PORTB: проверка состояния переключателей на
//лицевой панели,длительностью не менее 50 мс
//*****************************************************************

int process_valid_input_PORTB(unsigned char portx)
{
int   valid_input;           //установить флаг ошибочного входа
unsigned int current_count;

valid_input = TRUE;
current_count = TCNT;        // задать текущее состояние
while(TCNT < (current_count+12500)){   //отследить активный вход за 50 мс
if(portx==PORTB) valid_input = TRUE; else valid_input = FALSE;
if (!valid_input) break; //цикл while
)//конец while
return valid_input;
}

//*****************************************************************
//*****************************************************************
```

8.8.3. Испытание контроллера усилителя

Проверять программное обеспечение целесообразно при работе с имитатором усилителя, как мы уже упоминали в главе 2. Воспроизведем снова схему имитатора усилителя для удобства на рис. 8.20. На дешевомом имитаторе можно провести полное испытание алгоритма управления, проверив его работу при мыслимых сценариеях.

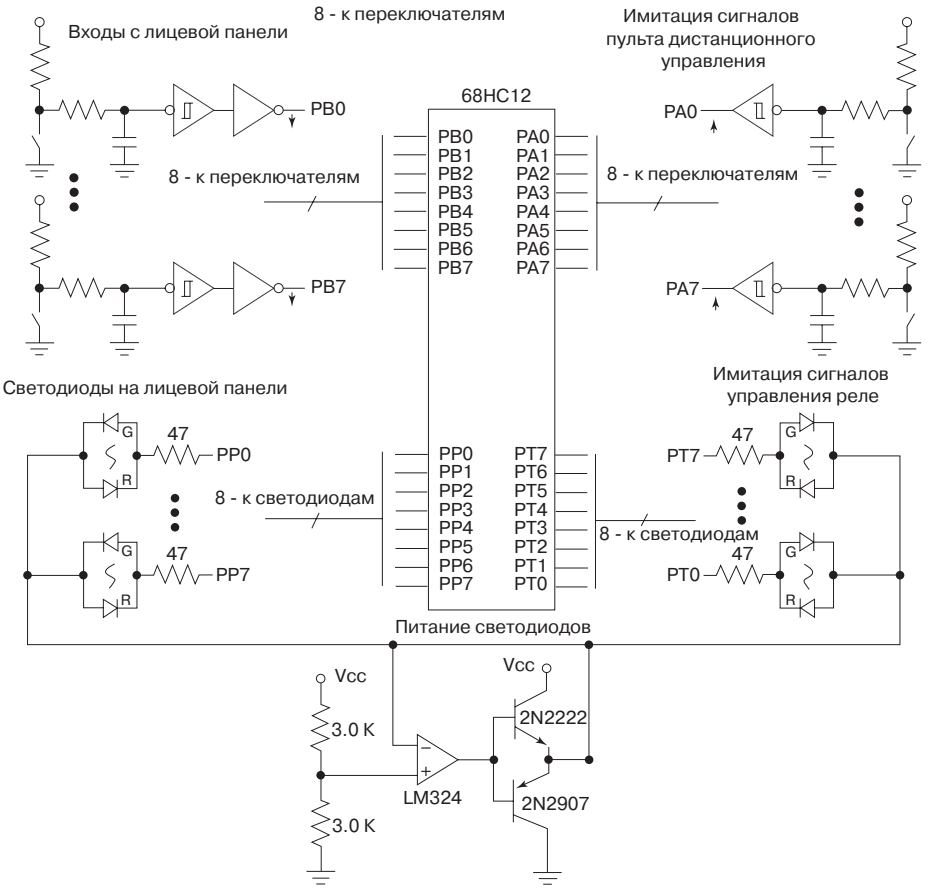


Рис. 8.20. *Имитатор усилителя*

8.9. Другая прикладная программа: цикл опроса с прерываниями

В этой программе мы разработаем приоритетную часть системы опроса с передним планом и фоном, позволяющей предотвратить перегрев транзисторов в стереоусилителе. Система приведена на рис. 8.21. Температура транзистора постоянно контролируется датчиком температуры LM34 (в пластмассовом корпусе), приклеенным к металлическом корпусу TO – 220 мощного транзистора. Напряжение на выходе датчика линейно связано с его температурой (коэффициент 10 мВ/ С).

Выход LM34 подан к один из входов аналогового компаратора, построенного на ОУ. На другой вход подается опорное напряжение, адающее порог температуры. Когда температура мощного транзистора достигает этого порога, на входе системы прерывания микроконтроллера появляется активный сигнал низкого уровня формирующий запрос на прерывание IRQ.

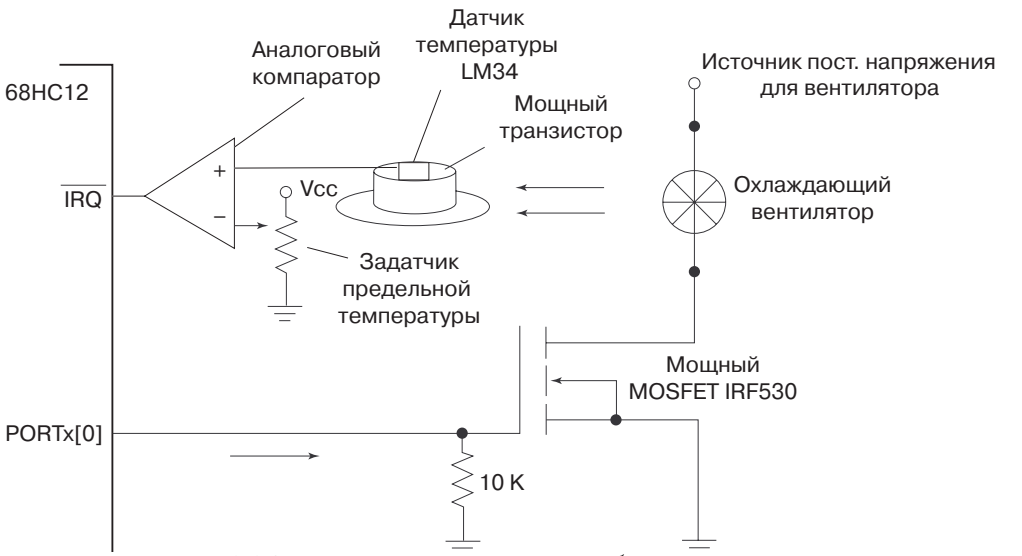


Рис. 8.21. *Транзисторные системы обнаружения перегрева*

При обработке запроса на прерывание, PORTX [0] формируется импульс с активным высоким уровнем, подключающий дополнительный вентилятор через мощный МОП-транзистор IRF530. Можно предумотреть определенную задержку в программе обработки прерывания и затем и затем выключить вентилятор. Когда температура транзистора упадет ниже установленного предела, вывод запроса на прерывание перейдет на высокий логический уровень, возвращая процессор к выполнению основной программы.

Если температура транзистора упадет до безопасного уровня процессор продолжит нормальную работу, (при этом на входе запроса на прерывание установится сигнал высокого уровня), а при превышении температуры снова инициализирует прерывание (установится непрерывный низкий уровень на входе запроса на прерывание). Мы предлагаем вам самостоятельно разработать программу для системы, использующей прерывания, в качестве заданиядля самостоятельной работы (задание 2).

8.10. Сложное прикладное устройство: имитатор ОСРВ

8.10.1. Краткий обзор проекта

Некоторые из продвинутых алгоритмов планирования в режиме реального времени достаточно сложно выполнить. Сложность частично заключаетсмя в том, что внутренняя операция этих программ не визуализирована. Имитатор ОСРВ, показанный на рис. 8.22 может использоваться как помощь в разработка ОСРВ.

Имитатор состоит из набора блоков для 16 задач: пронумерованных в шестнадцатеричной системе от задачи 0 до задачи F. Каждый из таких блоков выполнен на микросхеме регистра с защелкой 74HC573. Состояние пересылается на в регистр,

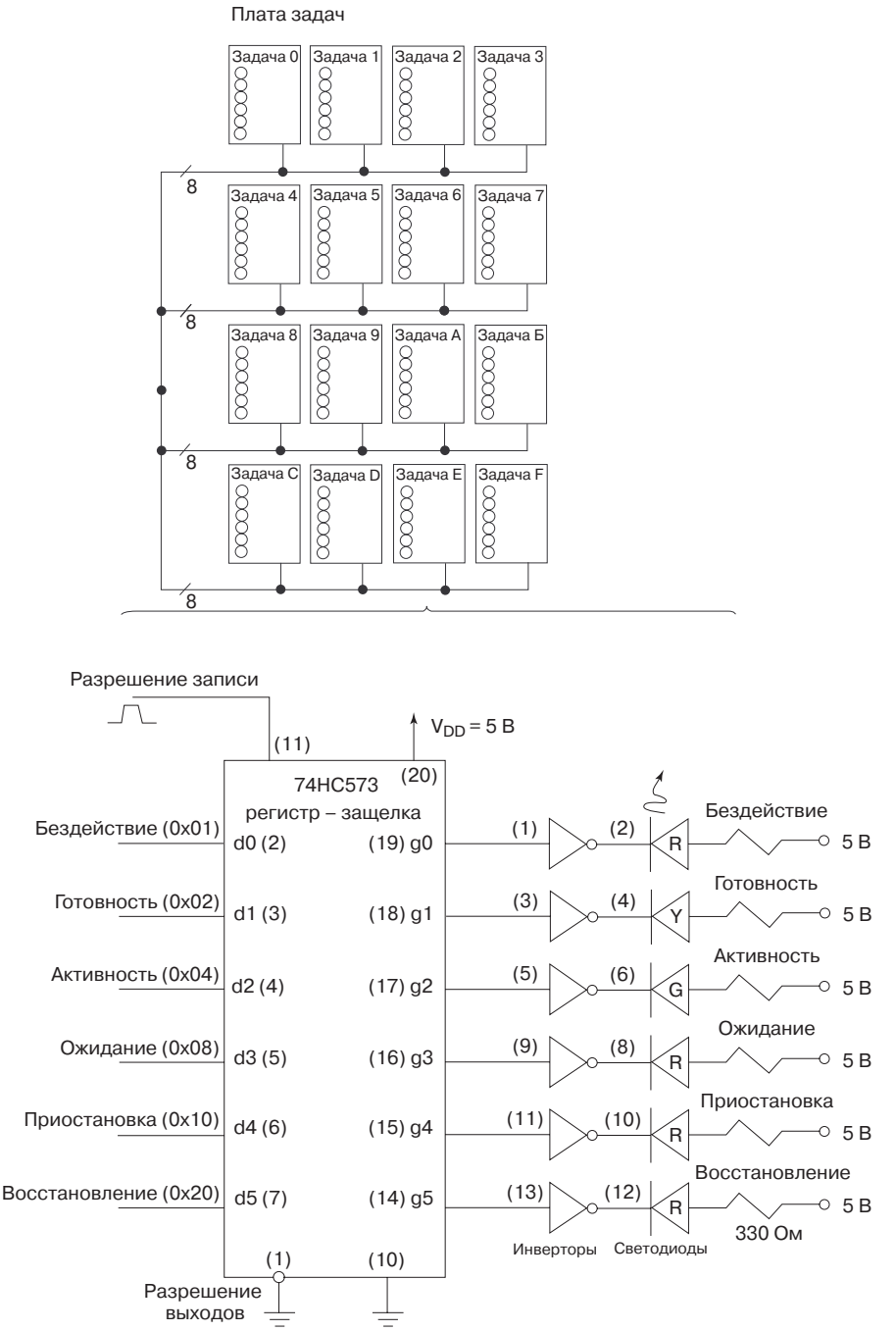


Рис. 8.22. *Имитатор ОСРВ*

где оно сохраняется вплоть до изменения. Состояние каждой задачи отображается набором светодиодов. Только один светодиод высвечивается в любой момент для каждой задачи. Мы использовали схему драйвера для светодиодов, рассмотренную в главе 5 для каждого из регистров 74HC573.

Схема симулятора приведена на рис.8.23. Демультиплексор 4 в 16 (74HC154) используется, чтобы выбрать отдельную задачу. Двоичный «адрес» задачи задается микропроцессором на входных линиях демультиплексора. В любой момент времени только один его выход находится в активном низком состоянии и, следовательно, выбирается только одна задача.

Обычно после инициировании все задачи находятся в состоянии бездействия. Чтобы выбрать различные состояния системы используется вспомогательная клавиатура, моделирующая работу системы ОСРВ. С вспомогательной клавиатураы набирается сначала конкретное состояние отображающее набор типов активности различных задач. Затем с той же клавиатуры набирается информация о том, активность какой из задач будет изменена.

Чтобы облегчить понимании концепций ОСРВ, обратимся снова к относительно гибким, быстро измененяющимся и просто визуализируемым сценариям с дилерским обслуживанием автомобиля, обслуживанием многих заказчиков официантом или даже к системе защиты в большой гостинице. Любые из этих примеров могут использоваться, чтобы иллюстрировать различные алгоритмы планирования.

Например, используемый автомобиль (задача) может иметь свойственный контекст, состоящий из года изготовления, модели, номера и показаний одометра. Автомобиль (задача) может находится в различных состояниях, идентичных состояниям задач (готовности, бездействия и т.д). После изготовления автомобиля, он может быть готов для продажи (готовность), проходить испытания (активность), недоступен из-за обслуживания (ожидание), продан (бездействие), и т.д. Состояние нескольких автомобилей (задач) может быть визуально отображено на имитаторе ОСРВ (до 16 задач). Новое состояние сценария может вводиться пользователем с шестнадцатеричной вспомогательной клавиатуры. Новое состояние можно показывать на ЖКД, а изменение списков с указателями получаемых при изменении состояния может быть отображено на экране ПК [Barrett 20041].

8.10.2. Типовой код

Детальное описание имитатора ОСРВ выходит за рамки данной книги. В этом разделе мы просто приведем код, позволяющий отображать и изменять состояние кажой из задач.

Функция update_task_status требует в качестве параметров идентификационного номера задачи и кода желательного изменения состояния задачи. Специфическая задача используется, чтобы конфигурировать `PORTA [3:0]` правильным четырех-разрядным двоичным кодом, который подается на микросхему 74HC154 декодера 4-16. Когда на демультиплексор подается код через выходной порта `PORTA[7]`, активируется триггер-защелка, соответствующий выбранной задаче. Это позволяет передать модифицированное состояние задачи, представленное на выходах порта T контроллера 68HC12 на светодиоды и, таким образом, показать состояние задачи на дисплее.

Алгоритм программы на UML для реализации этой функции представлен на рис. 8.24.

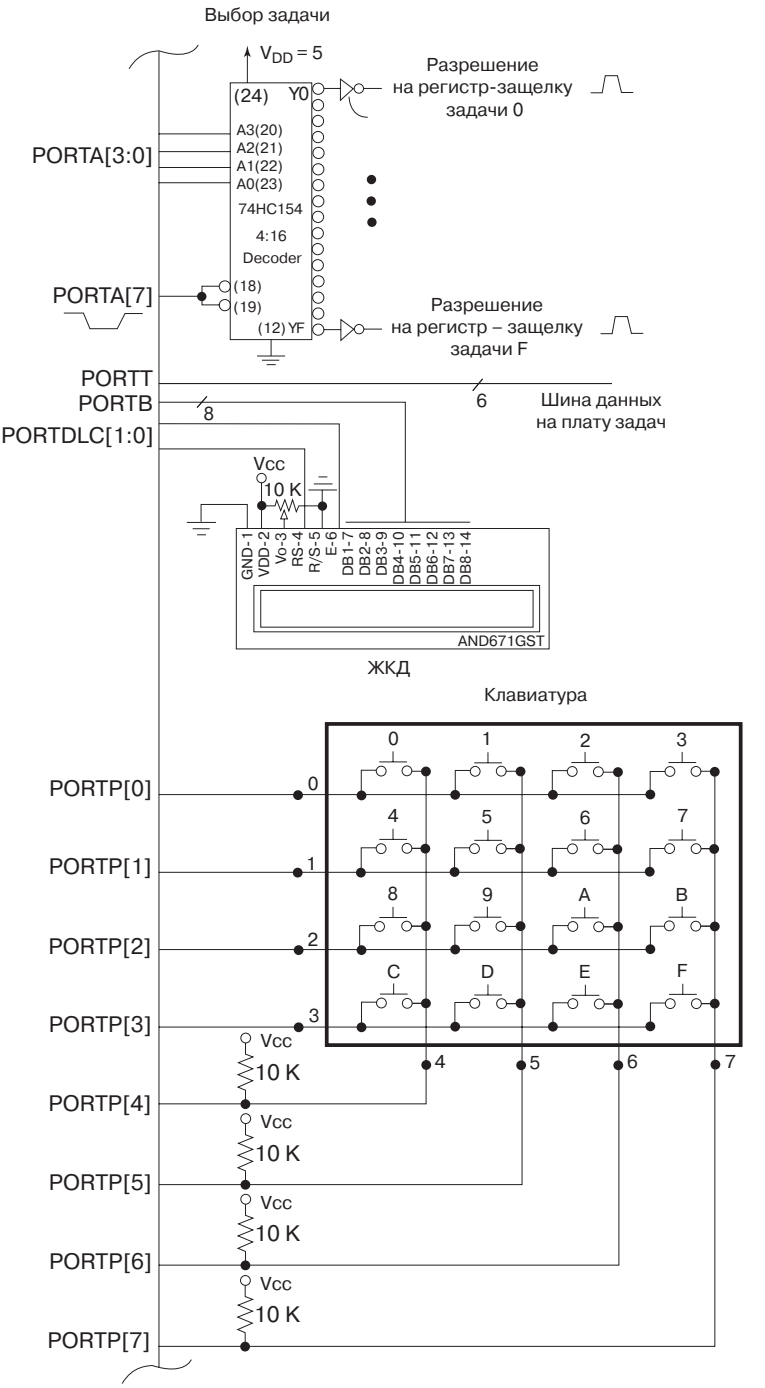


Рис. 8.23. *Аппаратное обеспечение симулятора ОСРВ*

```
//*****************************************************************
//имя файла: realtime.c
//авторы: Steve Barrett and Daniel Pack
//разработан: 1 июля, 2004
//последняя редакция: 1 июля, 2004
//*****************************************************************

//включенные файлы ************************************
#include<912b32.h>


//функции-прототипы*****************************************

void initialize_ports(void);      //инициализация портов
void initialize_LCD(void);        //инициализация ЖКД
void putchars(unsigned char);     //функция поддержки ЖКД - ввести символ
void putcommands(unsigned char); // функция поддержки ЖКД - ввести ко-
манду
void delay_5ms(void);              //задержка на 5 мс
void delay_100us(void);            //задержка на 100 мкс
void update_task_status(unsigned char task, char task_status);
                                   //изменить состояние задачи
//глобальные переменные*********************************************

//главная прпрограмма***************************************************
void main(void)
{
asm(" .area vectors(abs)\n"    //inline assembly statement
    " .org 0xFFF8\n" //initialize 68HC12 B32 reset vector
    " .word 0x8000, 0x8000, 0x8000, 0x8000\n"
    " .text");

initialize_ports();                    //инициализировать порты
initialize_LCD();                      // инициализировать ЖКД
update_task_status (0x00, 'R' ) ;      //изменить состояние задачи
}

//*****************************************************************
//initialize_ports: provides initial configuration for I/O ports
//*****************************************************************

void initialize_ports(void)
{
DDRA  =  OxFF;  //установить PORTA как выходной - управляющие линии
                // демультиплексора
DDRT   = OxFF;  // установить PORTT как выходной - состояние задачи
DDRB   = OxFF;  // установить PORTB как выходной - порт данныхдля ЖКД
CTDRDLC = OxFF; // установить PORT DLC как выходной -  сигналы
                //управления для ЖКД
DDRP   =  OxOF; // установить PORTP[3:0] как выходной, PORT[7:4] как
                //входной - для клавиатуры
PORTA  =  OxFF; //установить для PORTA все линии декодера высоомными (Hi-Z)
}
/*****************************************************************/
```

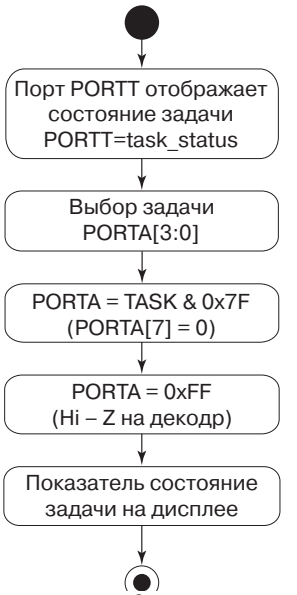


Рис. 8.24. *Алгоритм UML для функции update_task_status*

```
/*************************************************************/
/*update_task_status: изменить состояние конкретной задачи       */
/*в соответсвии с допустимым переходом (рис. 8.14)            */
/*************************************************************/

void update_task_status(unsigned char task, char task_status)
{                    //установить состояние задачи на выходе порта T
switch(task_status)(
  case 'D': //бездействие (D)
           PORTT = 0x01;
           break;

  case'R' : //готовность (R)
           PORTT = 0x02
           break;

  case'A' : //активность (A)
           PORTT = 0x04;
           break;

  case'W' : //ожидание (W)
           PORTT = 0x08
           break;

  case'S' : // приостановка(S)
           PORTT = 0x10;
           break;

  case'X' : //восстановление (X)
           PORTT = 0x20;
           break;
```

```
  }
PORTA = task& 0x7F;  /*выбор задачи, активизируйте декодер */
PORTA = OxFF         /*Высокоомный выход (Hi-Z) декодера */
}

/*************************************************************/
/*************************************************************/
```

8.11.Заключение по главе 8

В этой главе мы познакомили вас с концепциями операционных систем реального времени. Мы не хотели выбирать конкретные ОСРВ и затем рассматривать их работу. Вместо этого, мы сосредоточились на концепциях, связанных с самими ОСРВ и на ключевых вопросах их применения. Мы рассмотрели терминологию ОСРВ, структуры данных, алгоритмы планирования и затруднения, встречающиеся при разработке этих систем.

8.12. Что еще почитать?

1. Barrett S. F, D. J. Pack, C Straley, L. Sircin, and G. Janack. «Real-Time Operating Systems: A Visual Simulator.» Paper presented at the annual meeting of the American Society for Engineering Educations, June 2004.
2. Ganssle, J. «Writing a Real-Time Operating System-Part I: A Multitasking Event Scheduler for the HD64180.» *Circuit Cellar Ink* (January/February 1989): 41-51.
3. Ganssle, J. «Writing a Real-Time Operating System-Part II: Memory Management and Applications for the **HD64180.**» *Circuit Cellar Ink* (March/April 1989): 30-33.
4. Ganssle, J. «An OS in a CAN.» *Embedded Systems Programming* (January 1994): 1-6. ImageCraft Creations, Inc. «ICC12, ImageCraft C Compiler and Development Environment for Motorola HC12.» 2001.
5. Korsch, J. F., and L. J. Garrett. *Data Structures, Algorithms, and Program Style Using C* Boston: PWS-Kent Publishing Company, 1988.
6. Labrosse, J. J. *Micro C/OS-II The Real-Time Kernel,* 2nd ed. Lawrence, KS: CMP Books, 2002.
7. Lafore, R. *The Waite Group's Microsoft C Programming for the PC,* 2nd ed. Carmel, IN, Howard W. Sams and Company, 1990.
8. Laplante, P. *Real-Time Systems Design and Analysis:* An Engineer's Handbook. New York: IEEE Computer Society Press, 1993.
9. Miller, G. H. *Microcomputer Engineering,* 2nd ed. Englewood Cliffs, NJ: Pearson Education, 1998.
10. Moore, R. *How to Use a Real-time Multitasking Kernels in Embedded Systems,* Costa Mesa, CA: Micro Digital Associates, 2001.
11. Motorola Inc. «68HC12 M68EVB912B32 Evaluation Board User's Manual.» Motorola Document 68EVB912B32 UM/D, 1997.
12. Motorola Inc. «HC12 M68HC12B Family Advance Information.» Motorola Document M68HC12B/D, 2000.
13. Pack, D. J., and S. F. Barrett. *68HC12 Microcontroller: Theory and Applications.* Upper Saddle River, NJ: Prentice Hall, 2002.

8.13. Вопросы и задания

Основные

1. Что такое – ОСРВ?
2. Что называется задачей в системах реального времени?
3. Что такое контекст задачи? Приведите конкретные примеры содержимого контекста задачи.
4. Опишите действия, которые ОСРВ должен выполнить относительно задачи.
5. Что такое – ядро ОСРВ? Какими основными свойствами оно должно обладать?
6. Каковы различия между глобальной и локальной переменной?
7. Что понимается под динамическим распределением памяти?
8. Какая память (RAM, ROM, и т.д.) используется при динамическом распределении памяти? Объясните почему.
9. Опишите следующие структуры данных. Где они обычно используются:
   - Структура / запись;
   - Список с указателями;
   - Очередь;
   - Круговая очередь;
   - Стек.

Более сложные

1. Объясните различие между жесткой, твердой, и мягкой системами в режиме реального времени. Приведитете пример для каждой из них.
2. Сравните динамическую память со стеком. Где они обычно размещаются в системе памяти? Почему?
3. Каких методов программирования нужно избегать, чтобы сохранить память RAM? Почему?
4. Определите каждое из различных состояний, в которых может находиться задача. Во скольких состояниях задача может находиться одновременно?
5. Что такое – управляющий блок задачи (TCB)? Из чего он должен состоять? Какая структура данных была бы хорошим выбором для TCB? Почему?
6. Какова функция диспетчера / планировщика в ядре ОСРВ? Определите каждый из различных алгоритмов планирования и их свойственные им преимущества и недостатки.
7. Что такое конкуренция? Как это происходит? Как этого избежать?
8. Что такое повторная входимость? Как это происходит? Как это предотвратить?
9. Что понимается под отказоустойчивой работой ОСРВ? Почему – это проблема является критичной при разработке ОСРВ?
10. Управляемая прерыванием ОСРВ должна быть выполнена на 68HC12. Вы решили, что система всегда должна отвечать на прерывания с более высоким приоритетом, когда они происходят. Как это может быть выполнено? Вспомните, что 68HC12 автоматически отключает систему прерывания при ответе на прерывание, Подсказка: Посмотрите описание команд CLI и SEI ассемблера 68HC12.

Исследовательские

1. Разработайте стек и связанные с ним функции, использовав список с указателями для динамического распределения памяти.
2. Разработайте приоритетную часть системы фонового опроса с передним планом, для защиты от перегрева транзисторов, описанной в применениях раздела 8.9.
   На рис. 8.25 (совпадающим с рис. 8.21 и повторенном здесь для удобства) показана системаз ащиты от транзистора от перегрева . Температура транзистора постоянно контролируется датчиком температуры LM34 (в пластмассовое корпусе) приклеенным к металлическому корпусу мощного транзистора К-220. Напряжение на на выходе датчика линейно связано с его температурой (коэффициент 10 мВ/°С). Выход LM34 подан на один из входов аналогового компаратора, построенного на ОУ. На другой вход подается опорное напряжение, задающее порог температуры. Когда температура мощного транзистора достигает этого порога, на входе системы прерывания микроконтроллера появляется активный сигнал низкого уровня формирующий запрос на прерывание IRQ.
   При обработке запроса на прерывание, PORTX [0] формируется импульс с активным высоким уровнем, подключающий дополнительный вентилятор через мощный МОП-транзистор IRF530. Можно предумотреть определенную задержку в программе обработки прерывания и затем и затем выключить вентилятор. Когда температура транзистора упадет ниже установленного предела, вывод запроса на прерывание перейдет на высокий логический уровень, возвращая процессор к выполнению основной программы.
3. Разработайте и проверьте функцию, позволяющую модифицировать состояние задачи.

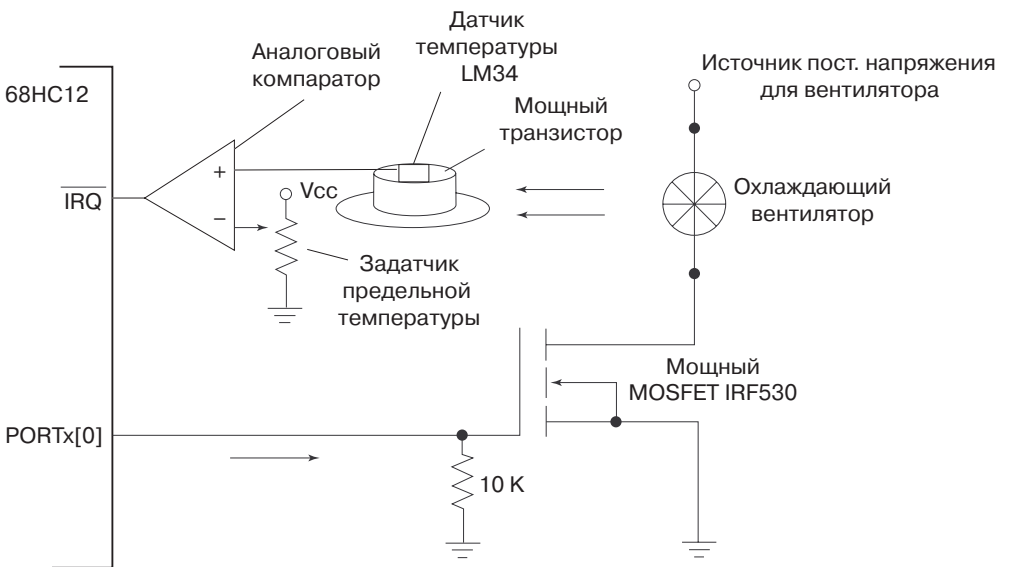


Рис. 8.25. *Система защиты транзистора от перегрева*

4. Выполните управляющий блок задачи (TCB) при помощи соответствующей структуры данных. Обеспечьте функции поддержки, чтобы обращаться к различным информационным полям TCB и модифицировать их.
5. Опишите различные методы выполнения межзадачной связи в ОСРВ.
6. Напишите короткую статью на две страницы, рассмотрев все за и против для двух вариантов: создание собственной ОСРВ и приобретение готовой системы.

Глава 9

РАСПРЕДЕЛЕННЫЕ СЕТИ С ИНТЕРФЕЙСОМ msCAN

ПОСЛЕ ИЗУЧЕНИЯ ГЛАВЫ ВЫ СМОЖЕТЕ:

ЦЕЛИ

- Рассказать о назначении информационных промышленных сетей;
- Описать протокол CAN;
- Рассказать об аппаратных средствах, необходимых для объединения в CAN сеть нескольких микропроцессорных контроллеров;
- Создать программу синхронизации контроллеров в CAN;
- Описать различия между модулем msCAN семейства HC12 и модулем msCAN семейства HCS12;
- Создать программу для модуля msCAN12, позволяющую осуществлять связь с другими узлами CAN сети;
- Кратко рассказать о работе контроллера последовательного обмена BDLC, также позволяющего объединять МК в исформационную сеть, но более медленную и менее надежную, чем CAN.

9.1 Компьютерные сети 596
9.2 Промышленные сети 597
9.3 Сети с протоколом CAN 598
9.4 Различия между контроллерами msCAN в составе 68HC12 и HCS12 626
9.5 Пример программирования контроллера msCAN 627
9.6 Контроллер последовательного обмена BDLC 633
9.7 Заключение по главе 9 633
9.8 Что еще почитать? 634
9.9 Вопросы и задания 634

В этой главе, мы изучим еще один тип контроллера последовательного обмена, который наряду с ранее рассмотренными модулями SPI и SCI, присутствует в составе МК семейства 68HC12/HCS12. Сначала мы кратко остановимся на информационных сетях из обычных компьютеров, рассмотрим проблемы, связанные с их организацией. Далее мы исследуем возможность создания распределенных систем на основе микроконтроллеров, изучим аппаратные средства и режимы работы специального сетевого интерфейса msCAN в составе МК 68HC12/HCS12, научимся составлять простые программы для обмена сообщениями между несколькими узлами распределенной системы управления с обменом по протоколу CAN.

9.1. Компьютерные сети

Система, состоящая из некоторого количества компьютеров, объединенных локальной вычислительной сетью, имеет ряд преимуществ перед совокупностью отдельных компьютеров. Одной из основных причин, по которым целесообразно объединять независимые компьютеры в сеть, является способность совместно использовать ресурсы. Например, при создании сети компьютеров в офисе, нет необходимости оснащать каждый компьютер отдельным принтером. Принтер, связанный с информационной сетью, может использоваться всеми компьютерами этой сети. Точно так же программные ресурсы могут быть разпределены между объединенными в сеть компьютерами. Тогда каждый компьютер сети может обращаться к необходимым программам, которые хранятся на жестком диске другого компьютера. В дополнение к совместному использованию программ, информационые сети позволяют быстро обмениваться данными. Наиболее ярко преимущества обмена информацией по сети демонстрирует гигантская сеть Internet. Каждый пользователь Internet имеет доступ к чрезвычайно обширному объему данных, хранящихся в миллионах компьютеров во всем мире. И в заключение, сегодняшние компьютерные сети облегчают связь между пользователями компьютеров, увеличивая производительность, эффективность работы и прибыль компании.

Существует множество различных типов компьютерных сетей. Глобальная сеть (WAN -Wide Area Network), такая, например, как Internet – это компьютерная сеть, которая охватывает большую область информационных ресурсов, включая компьютерные ресурсы многих государств, стран и целых континентов. Локальная сеть (LAN – Local Area Network) – это сеть, которая существует внутри учреждения, компании,

или организации – например, сеть для вашей школы или вашей компании. Малая локальная сеть (SAN – Small Area Network) – это компьютерная сеть, созданная для небольшого офиса или для нескольких домашних компьютеров. При этом типе сети число компьютеров, связанных с сетью, обычно не превышает десяти.

9.2. Промышленные сети

В этой главе мы рассмотрим локальные промышленные сети, которые объединяют некоторое количество встраиваемых микропроцессорных систем. Поскольку встраиваемые системы в соответствие с их назначением принято также именовать именовать контроллерами (от слова control – управлять), то один из наиболее перспективных стандартов промышленных сетей назвали CAN (Controller Area Network). Сети CAN появились в середине 1980-ых в автомобильной промышленности, когда возникла необходимость связать несколько микроконтроллеров автомобиля с целью повышения эффективность управления. На рис. 9.1. приведен пример такой системы, в которой сеть CAN объединяет контроллер системы подачи топлива, контроллер, следящий за уровнями нескольких жидкостей в агрегатах автомобиля, контроллер антиблокировки тормозов, контроллер привода каждоо из четырех колес, контроллер регулятора температуры, контроллер приборной панели и навигационный контроллер. Достаточно быстро концепция CAN вышла за пределы автомобильной промышленности. Сегодня мы можем обнаружить сети CAN в звуковых системах, домашних театрах, системах связи, военных системах и в ряде бытовых приборов.

Все сети должны иметь протоколы – правила, по которым устанавливается связь между компонентами сети. Эти правила включают общие данные о длине, синхронизации передаваемых или получаемых битов, методах проверки правильности полученного сообщения, и способах согласования большого числа узлов сети, одновременно осуществляющих обмен информацией.

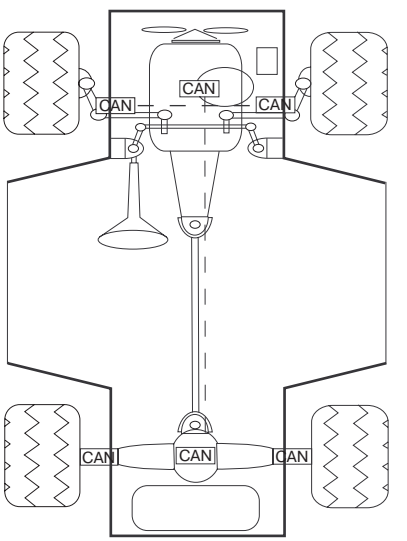


Рис. 9.1. *Простая локальная контроллерная сеть для автомобиля*

9.3. Сети с протоколом CAN

В этом разделе мы подробно рассматрим локальные контроллерные сети с протоколом CAN. В частности мы изучим принципы работы интерфейсного модуля msCAN12 в составе микроконтроллеров 68HC12/HCS12 и шаги, необходимые, чтобы конфигурировать и программировать модули msCAN12 для связи с другими такими же модулями при их совместной работе в сети. Но прежде чем начать обсуждение свойств периферийных модулей msCAN12, опишем протокол обмена, используемый в CAN сетях.

9.3.1. Протокол CAN

Протокол CAN был первоначально создан для автомобильных прикладных решений. Но вскоре концепции и преимущества CAN привлекли внимание разработчиков из других областей промышленности, и этот протокол стал одним из самых распространенных методов работы с сетями для небольших распределенных систем реального времени. Мы проведем здесь лишь краткое рассмотрение этого протокола; более подробную информацию читатели могут найти в дополнительной литературе, приведенной в конце этой главы.

Последняя редакция протокола CAN (версия 2.0) состоит из двух частей: части A (стандартный формат) и B (расширенный формат). Часть A представлена следующими тремя уровнями: объектным, уровнем передачи и физическим уровнем. Объектный уровень является связующим звеном между уровнем передачи и прикладной программой, выполняемой ценральным процессором МК. На этом уровне происходят все программные обработки CAN-сообщений. Уровень передачи обеспечивает полное соответствие сообщения стандарьному протоколу обмна, в то время, как на физическом уровне происходит реальная передача сингналов сообщения.

Часть B версии 2.0 протокола касается уровня передачи данных и физического уровня. Уровень передачи данных в свою очередь включает в себя подуровень управления логическими связями LLC (Logical Link Control) и подуровень управле-

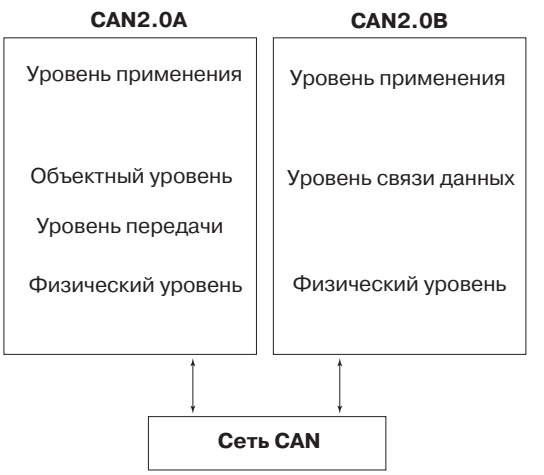


Рис. 9.2. *Соответствие терминов модели ISO/OSI для протоколов CAN 2.0 A/B*

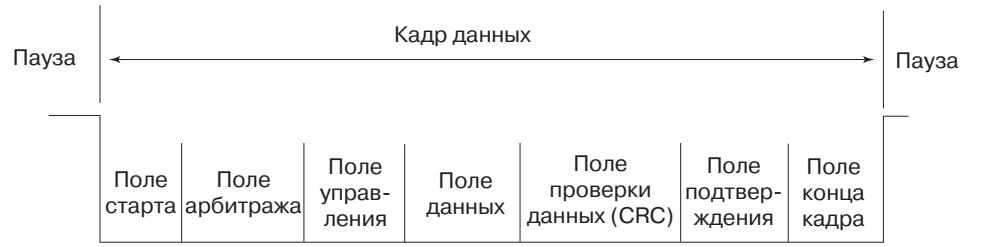


Рис. 9.3. *Кадр данных CAN*

ния доступом к среде MAC (Medium Access Control). Совокупность функций, выполняемых подуровнями LLC, MAC и физическим уровнем, соответствует функциям объектного уровня, уровня передачи и физического уровня для части A протокола CAN 2.0. На рис. 9.2 показаны части A и B протокола CAN 2.0. В дальнейшем мы будем в равной степени использовать термины из частей A и B протокола CAN 2.0.

Уникальным свойством протокола CAN является то, что при работе с сообщениями не указываются адреса узла отправителя и узла получателя. Вместо этого, в каждое сообщение вставляется идентификатор, который позволяет этому сообщению быть принятым любым узелом сети CAN без всякого изменения содержимого передаваемых кадров. Это также означает, что одно и то же сообщение может быть одновременно принято несколькими узлами, позволяя осуществлять так называемый широковещательный режим или мультикастинг. Протокол CAN содержит прямую методику арбитража и сложные механизмы обнаружения ошибок. Он поддерживает также энергосберегающие режимы для узлов: спящий режим и режим пробуждения.

Данные в CAN передаются короткими сообщениями – кадрами стандартного формата. В CAN существуют четыре типа сообщений: кадр данных, кадр удаленного запроса, кадр ошибки и кадр перегрузки. Проведем короткий обзор этих типов кадров.

Кадр данных содержит семь полей, показанных на рис. 9.3. Поле старта состоит из одного активного бита (логический нуль). Получающий узел использует этот бит, чтобы синхронизировать начало получения кадра данных. Поле арбитража содержит идентификационный номер сообщения, который используется принимающим узлом для принятия решения о приеме данного конкретного кадра данных. Идентификатор содержит 11 для стандартного формата и 29 бит расширенного формата кадра данных. Поле арбитража также содержит бит удаленного запроса передачи (RTR), который используется, чтобы отличить кадр данных от кадра удаленного запроса. Для кадра данных, этот бит должен быть доминантным (логический нуль), для кадра удаленного запроса – рецессивным (логическая 1).

Поле управления содержит четыре бита, которые определяют длину данных в байтах. Длина данных определяется четырьмя отдельными битами, позволяя устанавливать значение от одного до восьми байт. Поскольку рецессивный бит соответствует передаче логической 1, то четыре активных бита в этом поле информируют о длине поля данных, равной 0. Поле данных содержит фактическое сообщение кадра передачи. Старший бит каждого байта передается первым. Поле проверки данных содержит контрольное число, которое используется принимающим узлом для проверки отсутствия ошибок в принятом кадре. Контрольное число формируется по специальным правилам (CRC код), с которыми читатель может подробно ознакомиться в литературе, приведенной в конце главы. Поле подтверждения (ACK) со-

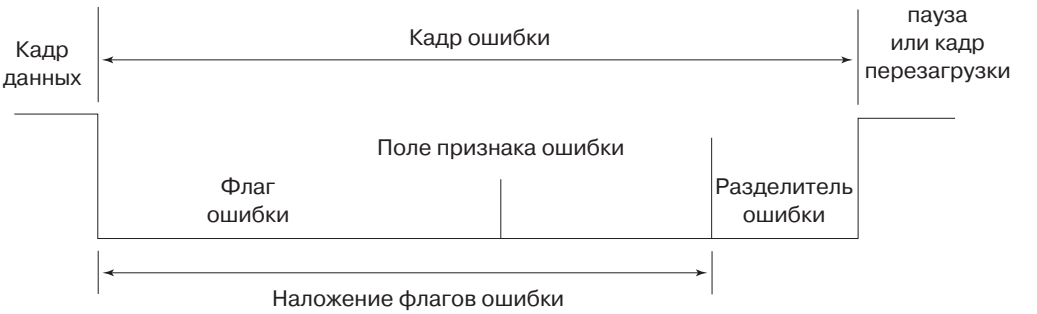


Рис. 9.4. *Кадр ошибки CAN*

держит два бита. Передающий узел в поле подтверждения выставляет две 1. Если прием данных завершился успешно, то принимающий узел в первом бите выставляет на шину 0. Передающий узел в процессе передачи осуществляет считывание состояния шины. Поэтому наличие доминантного уровня 0 в первом бите поля подтверждения будет воспринято им как подтверждение приема данных. Первый бит поля подтверждения называется ACK-Slot, второй ACK-Delimiter. Поле конца кадра представляет собой последовательность из семи единичных битов, которые свидетельствуют об окончании кадра.

Кадр удаленного запроса используется принимающими узлами, чтобы запросить повторную передачу кадра данных. Кадр удаленного запроса идентичен кадру данных за исключением того, что он не содержит поля данных. Для того, чтобы принимающий узел мог отличить кадр удаленного запроса от кадра данных, в поле арбитража предусмотрен специальный бит RTR. Рецессивное состояние бита RTR (логическая 1) обозначает кадр удаленного запроса, а доминантное состояние (логисческий 0) – кадр данных.

Кадр ошибки, представленный на рис. 9.4, информирует узлы сети о том, что на шине CAN произошла ошибка. Каждый кадр ошибки состоит из поля признака ошибки и поля разделителя ошибки. Поле признака ошибки содержит либо активные флаги ошибки (шесть доминантных бит), либо пассивные флаги ошибки (шесть рецессивных бит). Мы дадим определение понятиям активных и пассивных ошибок в разделе обработки ошибок. Следует заметить, что в системах с многочисленными узлами на шине CAN число доминантных бит в признаке ошибки может увеличиваться до 12. Это необходимо, чтобы все компоненты системы могли использовать флаги ошибки. Поле разделителя ошибки состоит из восьми рецессивных бит.

Кадр перегрузки, как показано в рис. 9.5, имеет тот же формат, что и кадр ошибки. Флаг перегрузки составлен из шести доминантных бит. Биты флага перегрузки устанавливаются, когда:

- принимающий узел не может обработать корректные кадры за выделенное время, и требует задержки;
- на интервале паузы появился доминантный бит.

Кадры данных и удаленного запроса на шине CAN отделяются от других кадров интервалами паузы, которые состоят, по крайней мере, из трех рецессивных бит. Разделитель перегрузки составлен из восьми рецессивных бит.

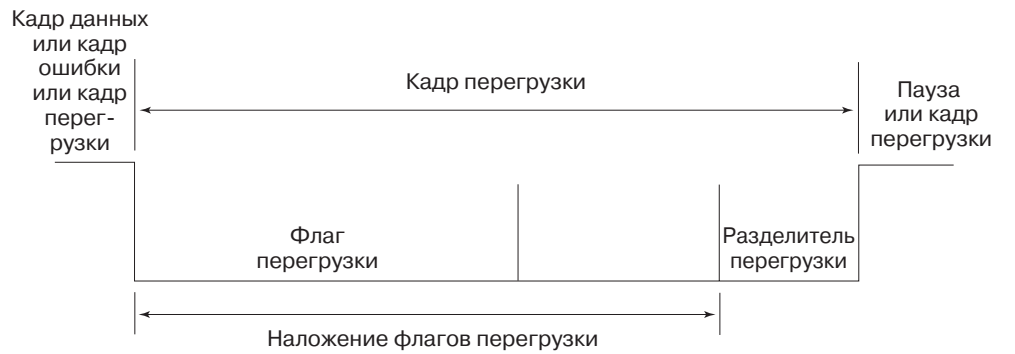


Рис. 9.5. *Кадр перегрузки CAN*

Обработка ошибок. Во время передачи сообщения по CAN шине могут происходить ошибки. Когда активным (передающим сообщение) или пассивным (принимающим) узлом системы обнаружена ошибка, соответствующий узел выставляет на шину кадр ошибки, рассмотренный выше. Если активный узел передает кадр ошибки, флаг ошибки называется активным флагом ошибки; если ошибка зафиксирована в пассивном узле, то и флаг называется пассивным. Имеются пять типов ошибок, которые могут вызывать передачу кадра ошибки: ошибка разряда (1), ошибка заполнения (2), ошибка избыточности (3), ошибка формы (4) и ошибка подтверждения (5).

Ошибка разряда происходит, когда передающий узел обнаруживает несоответствие выставляемого на шину бита и реального состояния шины в тот же момент времени. Флаг ошибки заполнения устанавливается, когда контроллер CAN обнаруживает в передаваемом кадре шесть последовательных доминантных или шесть последовательных рецессивных бит. Ошибка контроля происходит, когда значение контрольного числа CRC, вычисленное приемником, не соответствует контрольному числу CRC, полученному в конце передачи. Ошибка формы фиксируется, когда в одном из полей кадра содержатся недопустимые биты. И наконец, ошибка подтверждения происходит, когда отсутствует доминантный бит в поле ACK-Slot.

Синхронизация бита. На рис. 9.6 показано, что интервал времени передачи одного бита по шине CAN разбивается на четыре временных сегмента: сегмент синхронизации, сегмент распространения, сегмент буфера фазы 1 и сегмент буфера фазы 2. На интервале первого сегмента появляется фронт, который используется, чтобы синхронизировать узлы, подключенные к шине. Выборка логического состояния бита производится после окончания сегмента буфера фазы 1 (точка выборки). Сегмент времени распространения учитывает задержку передатчика и приемника и время распространения сигнала по шине. Сегменты фазы 1 и 2 могут увеличиваться или уменьшаться по длительности посредством программных уставок при инициализации. Такое решение позволяет увеличить надежность передачи данных по шине.

9.3.2. Модуль контроллера последовательного обмена msCAN12

Микроконтроллеры 68HC912BC32, 68HC912D60, 68HC912DG128 и 68HC912DT128 имеют в своем составе один или несколько встроенных контроллеров последовательного обмена в стандарте CAN. Эти контроллеры принято называть модулями msCAN (Motorola Scalable Controller Area Network). Семейство 68HC12/HCS12 отличается тем, что модели МК, входящие в его состав, оснащены сразу несколькими модулями msCAN. Например, МК 68HC912BC32 и 68HC912D60 имеют в своем составе один модуль msCAN, 68HC912DG128 – 2 модуля, 68HC912DT128 – 3 модуля, а в семействе HCS12 возможет даже вариант МК с 5 модулями «на борту».

Модули msCAN в составе разных МК полностью идентичны, работают в соответствии с протоколом CAN версии 2.0 A/B. Далее мы подробно рассмотрим, как работает один из них, в составе МК MC9S12DP256.

Каждый модуль msCAN связывается с внешним миром, используя выход TxCAN, подключенный к передатчику, и вход RxCAN, связанный с приемником. Названные входы/выходы МК через схему преобразования уровней, называемую трансивером CAN, подключены к CAN шине. Каждый модуль msCAN состоит трех подсистем: блока передатчика, блока приемника и блока прерывания. Мы опишем каждую из этих подсистем в следующих разделах.

Режимы работы CAN. Прежде, чем перейти к рассмотрению каждой из подсистем, входящих в состав модуля контроллера msCAN, сравним нормальный режим работы контроллера msCAN12 с его работой в режиме энергосбережения. Когда модуль CAN неактивен (т.е. не производит обмена с другим устройством по CAN шине), пользователь заинтересован в том, чтобы уменьшить мощность потребления самого модуля msCAN12. Для этого необходимо перевести модуль msCAN12 в энергосберегающий режим. Каждый модуль msCAN12 может работать не только в нормальном, но и в трех энергосберегающих режимах: спящем режиме, режиме программного сброса и режиме отключения.

Выбор одного из трех энергосберегающих режимов выполняется путем соответствующей конфигурации msCAN12 посредством изменения состояния регистра управления CMCR0, формат которого показан на рис. 9.7.

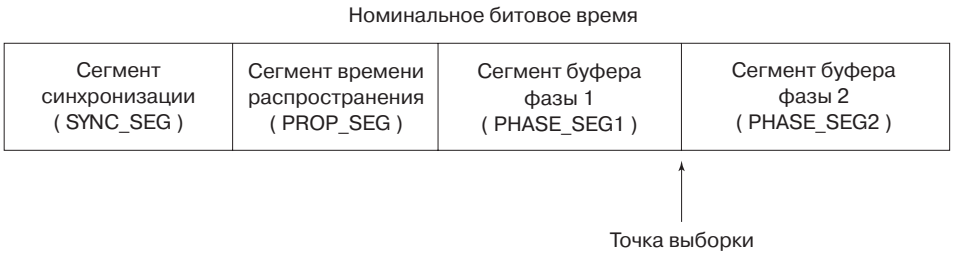


Рис. 9.6. *Номинальные сегменты времени передачи бита*

Имя регистра: CMCR0 Адрес :$0100

	7	6	5	4	3	2	1	0
	0	0	CSWAI	SYNCH	TLNKEN	SLPAK	SLPRQ	SFTRES
Сброс :	0	0	1	0	0	0	0	1

Рис. 9.7. *Формат регистра управления CMCR0*

Пятый бит этого регистра CSWAI (CAN-Stops-in-Wait) представляет собой бит выбора режима останова или режима ожидания. Если этот бит очищен, модуль переходит в режим ожидания. Если бит CSWAI =1, модуль не получает сигналов синхронизации, т.е останавливается. Четвертый бит – бит синхронизации SYNCH – указывает, готов ли модуль CAN к связи с другими узлами сети. Когда этот бит очищен, msCAN12 не синхронизирован с шиной CAN, когда же бит установлен, то msCAN12 и шина CAN синхронизированы. Третий бит TLNKEN представляеет собой бит состояния входа таймера. Когда этот бит очищен, то вход таймера связан с соответствующим портом. Если же установлен, это указывает, что на вход таймера подается выходной сигнал msCAN12.

Второй бит SLPAK является флагом спящего режима. Он показывает, находится или нет контроллер msCAN12 в спящем режиме: логический 0 указывает, что msCAN12 не находится в спящем режиме, в то время, как логическая 1, показывает что этот режим установлен. Бит запроса спящего режима SLPRQ используется, чтобы запросить перевод модуля msCAN12 в спящий режим: установка этого бита заставляет модуль функционировать в рабочем режиме, очистка – переводит модуль в спящий режим.

И наконец, бит SFTRES используется центральным процессором, чтобы немедленно перевести модуль msCAN12 в состояние сброса. Когда этот бит установлен, любая текущая передача информации прерывается и прекращается синхронизация контроллера CAN с шиной CAN. Когда этот бит очищен, msCAN12 работает в нормальном режиме.

Флаги SFTRES, SLPAK и CSWAI регистра CMCR0 используются, чтобы выбрать один из четырех режимов работы msCAN12 (нормальный, программного сброса, спящий и выключения). Мы обсудим эти режимы в следующих трех разделах. Поскольку ЦП микроконтроллера 68HC12 может работать в трех режимах (рабочем, ожидания и останова), имеются 12 возможных комбинаций режимов функционирования ЦП и модуля msCAN12. Четыре из этих 12 комбинаций не могут быть реализованы.

Когда ЦП функционирует в рабочем режиме, модуль msCAN12 может функционировать в спящем режиме, режиме программного сброса или в нормальном режиме. Режим выключения не допускается. ЦП может выбрать один из режимов msCAN12, конфигурируя три бита регистра CMCR0 следующим образом:

- Спящий режим: CSWAI = 0 или 1, SLPAK = 1 и SFTRES = 0;
- Режим программного сброса: CSWAI = 0 или 1, SLPAK = 0 и SFTRES = 1;
- Нормальный режим: CSWAI = 0 или 1, SLPAK = 0 и SFTRES = 0.

Обратите внимание, что бит SLPAK предназначен только для чтения и не может быть изменен под управлением программы. Чтобы установить или очистить бит SLPAK, Вы должны установить или очистить бит SLPRQ (бит 1) регистра CMCR0.

Когда ЦП функционирует в режиме ожидания, модуль msCAN12 может функционировать в любом из четырех режимов. Следующие конфигурации из трех битов в регистре CMCR0 используются, чтобы назначить один из нижеперечисленных режимов:

- Режим выключения: CSWAI = 1, SLPAK = 0 или 1 и SFTRES = 0 или 1,
- Спящий режим: CSWAI – 0, SLPAK = 1 и SFTRES = 0,
- Режим программного сброса: CSWAI = 0, SLPAK = 0, и SFTRES = 1
- Нормальный режим: CSWAI = 0, SLPAK = 0 и SFTRES = 0.

И наконец, когда ЦП функционирует в режиме останова, контроллер msCAN12 может находиться только в выключенном режиме. Когда ЦП вводит режим останова, то независимо от состояния битов CSWAI, SLPAK, и SFTRES регистра CMCR0, модуль msCAN12 вынужден перейти в режим выключения. Все другие режимы для модуля msCAN12 при этом невозможны.

Спящий режим msCAN12. Когда 68HC12 не вовлечен в активный обмен информацией с другими узлами сети, а ЦП функционирует в активном режиме либо в режиме ожидания, модуль msCAN12 может находиться в спящем режиме, переводя бит SLPAK в состояние логической 1. Еще раз повторим, что бит SLPAK устанавливается или очищается, битом SLPRQ регистра CMCR0. При этом внутренний таймер модуля msCAN12 останавливается. Модуль MsCAN12 переходит в спящий режим немедленно после того, как установлен бит SLPAK. Исключение составляют два следующих случая:

- когда при установке бита SLPAK модуль msCAN12 передает данные, он переходит в спящий режим только после того, как передача данных заканчивается;
- когда модуль получает данные, когда устанавливается бит SLPAK, прием данных должен быть закончен прежде, чем модуль перейдет в спящий режим.

Модуль msCAN12 выходит из спящего режима, если происходит одно из трех следующих событий:

- выполняется команда ЦП очистить бит SPLPRQ, вызывающая сброс бита SLPAK;
- выполняется команда ЦП установить бит SFTRES;
- на шине CAN появляются данные, предназначенные для соответствующего контроллера.

Режим программного сброса msCAN12. Этот режим используется, чтобы конфигурировать модуль msCAN12 при инициализации. При этом прекращается любая активость CAN и по командам ЦП изменяется конфигурация регистров msCAN12, регистров фильтра, и регистров, управляющих синхронизацией. Обычно перед переводом в режим программного сброса, модуль должен находиться в спящем режиме. В противном случае могут возникнуть ошибки, связанные с тем, что любая передача или прием данных могут быть прерваны при переходе контроллера msCAN12 в режим сброса, как только бит SFTRES будет установлен.

Режим отключения msCAN12. Режим отключения должен использоваться только для того, чтобы полностью или временно прекратить связь по CAN. Как и в случае режима мягкого сброса, необходимо переводить msCAN12 в режим отключения из спящего режима.

Модуль входит в режим отключения либо при переходе ЦП в режим останова, либо при установке бита CSWAI, в то время как ЦП работает в режиме ожидания.

Когда модуль msCAN12 переходит в этот режим, любая передача или прием данных немедленно прерывается.

Подсистема передатчика. Как только ЦП микроконтроллера 68HC12 создаст сообщение, передатчик модуля msCAN12 должен корректно переслать его в сеть CAN. Рассмотрим структуру подсистемы передатчика.

Адрес	Имя Регистра
0150	Регистр идентификатора 0
0151	Регистр идентификатора 1
0152	Регистр идентификатора 2
0153	Регистр идентификатора 3
0154	Регистр сегмента данных 0
0155	Регистр сегмента данных 1
0156	Регистр сегмента данных 2
0157	Регистр сегмента данных 3
0158	Регистр сегмента данных 4
0159	Регистр сегмента данных 5
015A	Регистр сегмента данных 6
015B	Регистр сегмента данных 7
015C	Регистр длины данных

Передающий буфер 0

Адрес	Имя Регистра
0160	Регистр идентификатора 0
0161	Регистр идентификатора 1
0162	Регистр идентификатора 2
0163	Регистр идентификатора 3
0164	Регистр сегмента данных 0
0165	Регистр сегмента данных 1
0166	Регистр сегмента данных 2
0167	Регистр сегмента данных 3
0168	Регистр сегмента данных 4
0169	Регистр сегмента данных 5
016A	Регистр сегмента данных 6
016B	Регистр сегмента данных 7
016C	Регистр длины данных

Передающий буфер 1

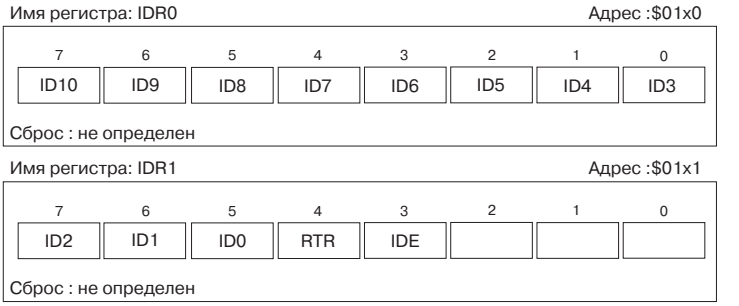
Адрес	Имя Регистра
0170	Регистр идентификатора 0
0171	Регистр идентификатора 1
0172	Регистр идентификатора 2
0173	Регистр идентификатора 3
0174	Регистр сегмента данных 0
0175	Регистр сегмента данных 1
0176	Регистр сегмента данных 2
0177	Регистр сегмента данных 3
0178	Регистр сегмента данных 4
0179	Регистр сегмента данных 5
017A	Регистр сегмента данных 6
017B	Регистр сегмента данных 7
017C	Регистр длины данных

Передающий буфер 2

Рис. 9.8 *Структура буферов передачи модуля msCAN12*

Как показано на рис. 9.8, передающий модуль msCAN12, содержит три 13-байтовых буфера. Все они имеют идентичную структуру данных, в которой первые четыре байта формируют идентификатор формата сообщения, следующие восемь байтов содержат фактическое сообщение, а последний байт определяет длину сообщения.

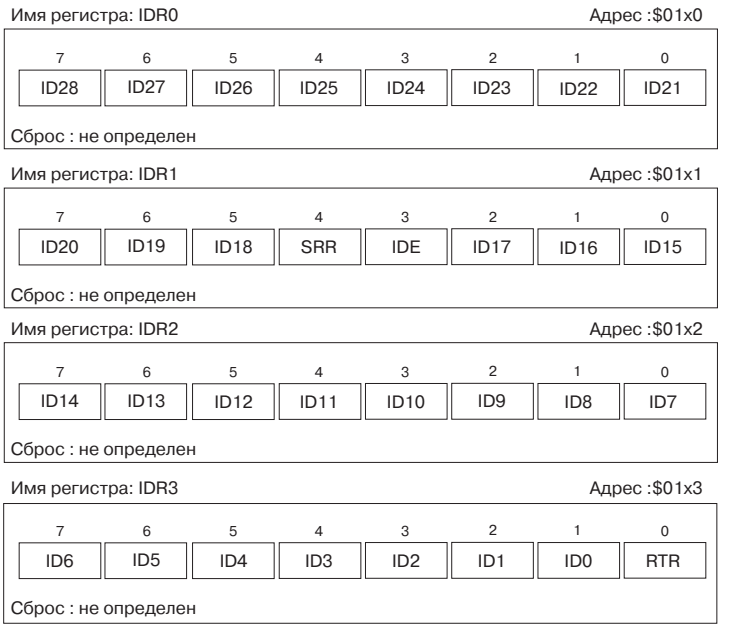
Рассмотрим каждый регистр передающего буфера. Четырехбайтовый регистр идентификатора определяет, соответствует ли сообщение протоколу Bosch CAN 2. 0A или Bosch CAN 2.0B. Первый из них требует 11-разрядного идентификатора, (стандартный формат), последний – 28-разрядного идентификатора (расширенный формат). На рис. 9.9а



Регистр: IDR2 - Регистр идентификатора 2 не используется
Регистр: IDR3 - Регистр идентификатора 3 не используется

x в адресе означает 5, 6 или 7 для буферов передачи 0, 1 и 2, соответственно

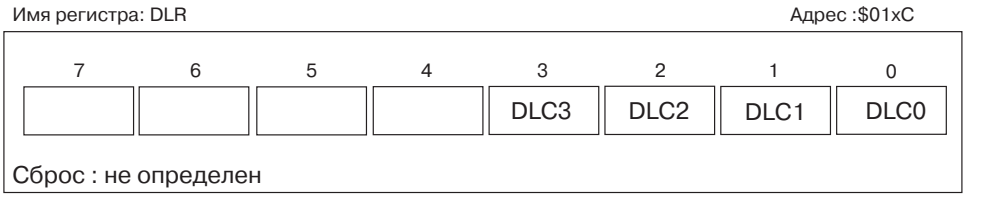
а)



x в адресе означает 5, 6 или 7 для буферов передачи 0, 1 и 2, соответственно

б)

Рис. 9.9. *Формат регистров идентификатора IDRO, IDR1, IDR2, и IDR3*
а) *при использовании протокола CAN 2 0A*
б) *при использовании протокола CAN 2 0B*



x в адресе равно 4, 5, 6 или 7 для буферов передачи 0,1,2 и 3, соответственно

Рис. 9.10. *Формат регистра длины данных*
Биты DLC3, DLC2, DLC1 и DLC0 используются, чтобы указать число байтов, содержащихся в регистрах сегмента данных

показано назначение каждого бита для четырех регистров идентификатора при использовании стандартного формата, на рис. 9.9б – то же для регистров расширенного формата.

Значения регистра идентификатора присваивают каждому сообщению номер, который другие модули сети CAN используют, чтобы распознать предназначенные для них сообщения. Флаг удаленного запроса передачи RTR (бит 4 IDR1 для стандартного формата или бит 0 IDR3 для расширенного формата) дает информацию о том, содержит ли буфер, посылающий данные другим модулям CAN, фактические сообщения или это просто запрос к другому модулю CAN, чтобы снова повторить сообщение. Если этот флаг сброшен, значит буфер содержит фактическое сообщение, если же установлен, то буфер используется, чтобы просто запросить повторную передачу сообщения. Контроллер CAN принимающего узла может использовать этот флаг, чтобы отличить данные от запроса.

Идентификатор расширенного формата IDE (бит 3 IDR1) определяет, использует ли передающий буфер стандартный или расширенный формат: логический 0 соответствует стандартный формату (11 бит), а логическая 1 – расширенному формату (29 бит). И наконец, флаг удаленного запроса SRR (бит 4 регистра IDR1) для расширенного формата должен всегда быть логической 1.

Восьмибайтовые регистры данных содержат фактические данные для пересылки. Передается содержимое регистра данных 0, затем содержимое регистра данных 1, регистра 2, и т.д, пока число байтов данных, не достигнет значения, записанного в регистре длины данных. Для каждого регистра сегмента данных, сначала передается бит 7. Регистр длины данных определяет число байтов данных, содержащихся в буфере передачи. На рис. 9.10 показан каждый бит регистра длины данных. Мы используем биты 0, 1, 2 и 3, чтобы обозначить число байтов данных. Для этого достаточно использовать только четыре бита, так как максимальное число байтов данных установлено равным восьми, в соответствии с числом регистров сегмента данных. Мы определяем число байтов данных как двоичное число при использовании бита 3 в качестве старшего бита. Например, если число байтов данных составляет 5, мы задаем следующие значения четырех битов: DLC3 = 0, DLC2 = 1, DLC1 = 0 и DLC0 = 1. Если мы используем все восемь байтов качестве данных, необходимо установить другие значения битов: DLC3 = 1, DLC2 = 0, DLC1 = 0 и DLC0 = 0.

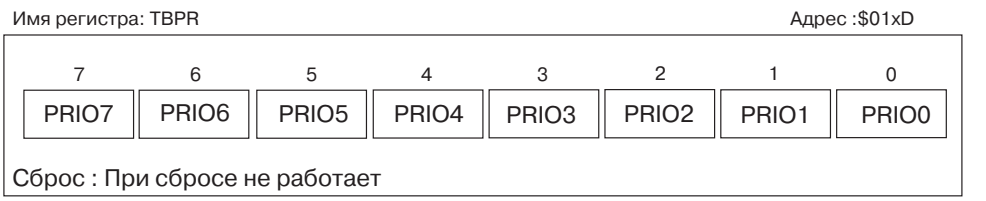
Чтобы загрузить регистры, показанные на рис. 9.8, желаемыми значениями прежде, чем осуществить любую передачу, должны быть выполнены соответствующие команды 68HC12. Например, следующий фрагмент программы на Си загружает передающий буфер 0 идентификатором $4D43 (представляющим собой симво-

лы «MC» в ASCII коде, поскольку используется контроллер фирмы Motorola) и сообщением $12345678, используя расширенный формат.

```
char *buffer;

buffer = 0x0150;
*buffer = 0x4D;      /*загрузить  IDR0 кодом символа M                  */
*(buffer+1) = 0x58;  /* загрузить IDR1, устаовить SRR = 1 и IDE=1       */
*(buffer+2) = 0xC0;  /* загрузить IDR2 кодом символа C                  */
*(buffer+3) = 0x00;  /* биты от  ID20 до ID13 ,чтобы представить C      */
*(buffer+4) = 0x12;  /* загрузить                                       */

*(buffer+5) = 0x34;
*(buffer+6) = 0x56;
*(buffer+7) = 0x78;
*(buffer+8) = 0x00;
*(buffer+9) = 0x00;
*(buffer+10) = 0x00;
*(buffer+11) = 0x00;

*(buffer+12) = 0x04;  /* данные длиной в 4 байта                        */
*(buffer+13) = 0x00;  /* установить наивысший приоритет                 */
```

Так как имеется три передающих буфера, модуль msCAN12 должен иметь некоторые средства, чтобы передавать записанную в них информацию в порядке, определяемом ее важностью. Упорядочение трех буферов осуществляется с помощью регистра приоритета буферов передачи (рис. 9.11). Регистр приоритета буферов передачи, также как и 13-байтовый буфер, связан с каждым из передаюших буферов. Контроллер msCAN12 оценивает значение приоритета для трех передаюших буферных регистров, и определяет порядок передачи информации из каждого регистра. Чем меньше значение, записанное в регистре приоритета, тем выше приоритет, соответствующего буфера. В случаях, когда значения приоритетов равны, буфер с самым низким индексом получает наиболее высокий приоритет.

Рассмотрим теперь три специальных регистра передающего модуля: регистр флагов CTFLG передатчика, регистр управления передатчика CTCR и регистр счетчика ошибок CTXERR. Формат регистра CTFLG показан на рис. 9.12. Биты 2, 1 и 0 являются флагами пустого буфера передачи, которые указывают, пуст или заполнен соответствующий буфер передачи. Логическая 1 указывает, что данный буфер передачи пуст и готов к новому использованию. Мы должны очистить этот флаг (записать 1 в соответствующий бит) после загрузки передающего буфера. Флаг также ус-



x в адресе равно 4, 5, 6 или 7 для буферов передачи 0,1,2 и 3, соответственно

Рис. 9.11. *Регистр приоритета передающего буфера msCAN12. Этот регистр используется, чтобы ранжировать порядок передачи сообщений из этого буфера.*

Имя регистра: CTFLG Адрес :$0106

7	6	5	4	3	2	1	0
0	ABTAK2	ABTAK1	ABTAK0	0	TXE2	TXE1	TXE0
0	0	0	0	0	1	1	1

Сброс :

Рис. 9.12. *Формат регистра CTFLG*

танавливается в случае успешного выполнения запроса на аварийное прекращение работы передатчика. Биты 7 и 3 этого регистра не используются.

Биты от 6 до 4 представляют собой флаги аварийного прекращения работы, показывающие, что выполнен запрос на аварийное прекращение работы. Логическая 1 показывает, что данное сообщение не было прервано; логический 0 свидетельствует, что сообщение было успешно прервано. Флаги запроса аварийного прекращения работы автоматически очищаются, когда сбрасываются флаги пустого буфера передачи.

Запрос на аварийное прекращение работы может быть сделан при изменении состояния регистра другого модуля передачи, названного регистром управления передатчиком CTCR. Формат регистра CTCR показан на рис. 9.13. В этом регистре также не используются биты 7 и 3. Программист может разботать программу, позволяющую запросить аварийное прекращение передачи с помощью битов 4, 5 и 6 этого регистра. Логический 0 соответствует отсутствию запроса, а логическая 1 показывает, что появился запрос. Когда сообщение успешно прерывается, соответствующий флаг TXE и флаг ABTAK устанавливается в CTFLG. Программист не может непосрелственно очистить биты ABTRQ, но они очищаются при установке соответствующих TXE флагов в регистре CTFLG.

Биты 2, 1 и 0 – это флаги разрешения на локальное прерывание, связанные с флагами пустого буфера в регистре CTFLG. Логическая 1 указывает, что соответствующий буфер передачи пуст (установлен флаг TXE в регистре CTFLG), вызвано прерывание из-за пустого передающего буфера. Логический нуль выключает систему прерывания.

Последним регистром, связанным с подсистемой передатчика модуля msCAN12, является регистр счетчика ошибок CTXERR. Как показывает его название, этот регистр подсчитывает число ошибок передачи. Когда контроллер msCANI2 находится в спящем режиме или в режиме программного сброса, регистр счетчика ошибок доступен только для чтения. Формат регистра счетчика ошибок показан на рис. 9.14.

Подсистема приемника. Приемный модуль контролера msCAN12 состоит из двух 13-байтовых буферов, восьми регистров идентификаторов приема CIDAR0...C1DAR7, восьми маскируемых фильтров идентификаторов приема CIDMR0...CIDMR7, регист-

Имя регистра: CTCR Адрес :$0107

7	6	5	4	3	2	1	0
0	ABTRQ2	ABTRQ1	ABTRQ0	0	TXEIE2	TXEIE1	TXEIE0
0	0	0	0	0	0	0	0

Сброс :

Рис. 9.13. *Формат регистра управления передатчиком msCAN12*

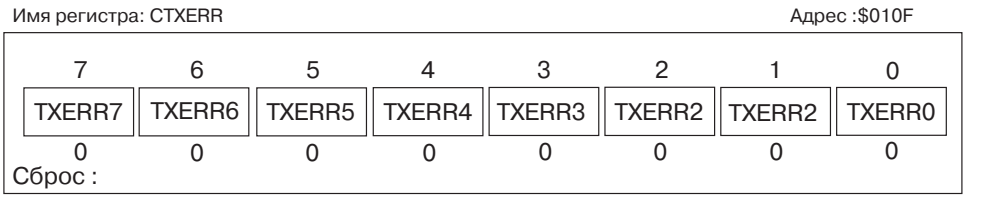


Рис. 9.14. *Счетчик ошибок передачи msCAN*

ра флагов приемника CRFLG, регистра разрешения прерываний приемника CRIER, регистра управления идентификаторами приема CIDAC и одного счетчика ошибок приемника CRXERR. В этом разделе, мы рассмотрим каждый из этих компонентов приемного модуля msCAN12.

Естественно, два 13-байтовых буфера используют тот же формат, что и буфер передачи msCAN12, показанный на рис 9.8. Сообщения, посылаемые устройством, подключенным к сети CAN, принимаются другими устройствами сети через эти буферы. Первым из приемных буферов, является так называемый предварительный буфер приема RxBG. Именно в него поступают сначала внешние сообщения. Второй приемный буфер, называемым основным буфером RxFG, доступен для ЦП 68HC12. Два буфера физически различны, но RxBG отображается в RxFG с помощью механизма, который мы рассмотрим далее в этом разделе, что заставляет буферы содержать одни и те же адреса. RxFG занимает физические адреса от $ 0140 до $014C.

Когда в сети появляется сообщение, оно сначала записывается в буфер RxBG. Параллельно с этим, сообщение проходит также через ряд фильтров, позволяющих определить, должно ли быть принято данное сообщение, и предназначено ли оно для данного устройства. Процесс фильтрации программируется с помощью регистра управления идентификаторами приема CIDAC, восьми регистров идентификаторов приема CIDAR0...CIDAR7 и восьми маскируемых фильтров идентификаторов приема CIDMR0...CIDMR7. Рассмотрим функции этих регистров.

Регистр CIDAC управляет типом фильтрации входящих сообщений. Кроме того, регистр содержит флаги, которые показывают, что получены сообщения с корректными идентификаторами, которые готовы к прочтению их ЦП 68HC12. На рис. 9.15 показан формат регистра CIDAC. Все биты за исключением битов 5 и 4 (IDAM1:IDAM0) предназначены только для чтения. Информация в биты IDAM1:IDAM0 может быть записана только, если при инициализации установлен бит SFTRES в регистре CMCR0. Биты 7, 6 и 3 не используются.

Следующие комбинации битов IDAM1:IDAM0 определяют число фильтров и размер каждого фильтра, необходимые чтобы установить режим принятия идентификатора.

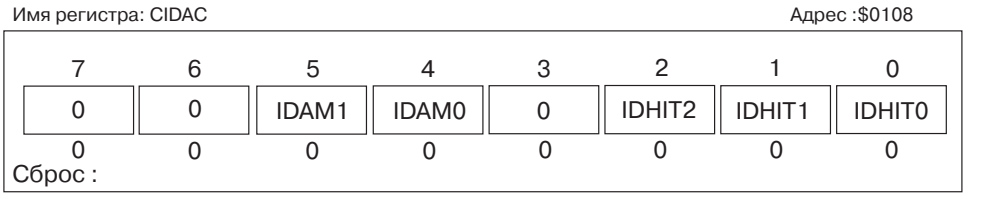


Рис. 9.15. *Формат регистра CIDAC*

- IDAM1=0 и IDAM0=0: используется два 32-разрядных фильтра приема
- IDAM1=0 и IDAM0=1: используется четыре 16-разрядных фильтра приема
- IDAM1=1 и IDAM0=0:используется восемь 8-разрядных фильтров приема
- IDAM1=1 и IDAM0=1: не используется никаких фильтров приема (фильтры закрыты)

Когда IDAM1 и IDAM0 равны 1, все сообщения игнорируются, и не записываются в основной буфер RxFG. Биты IDHIT2, IDHIT1, и IDHIT0 указывают на совпадения идентификаторов приема. Эти флаги устанавливаются, когда фильтры указывают, что обнаружены совпадения и буфер RxFG модифицирован. Состояния этих трех битов соответствуют следующем событиям:

- IDHIT2 = 0, IDHIT1 = 0 и IDHIT0 = 0: совпадение с фильтром 0
- IDHIT2 = 0, IDHIT1 = 0 и IDHIT0 = 1: совпадение с фильтром 1
- IDHIT2 = 0, IDHIT1 = 1 и IDHIT0 = 0: совпадение с фильтром 2
- IDHIT2 = 0, IDHIT1 = 1 и IDHIT0 = 1: совпадение с фильтром 3
- IDHIT2 = 1, IDHIT1 = 0 и IDHIT0 = 0: совпадение с фильтром 4
- IDHIT2 = 1, IDHIT1 = 0 и IDHIT0 = 1: совпадение с фильтром 5
- IDHIT2 = 1, IDHIT1 = 1 и IDHIT0 = 0: совпадение с фильтром 6
- IDH1T2 = 1, IDHIT1 = 1 и IDHIT0 = 1: совпадение с фильтром 7

Эти флаги также используются, чтобы осуществлять прерывания сообщений, если они разрешены. Если обнаружено больше чем одно условие совпадения, то приоритет получает фильтр с более низким номером.

Пропустит или не пропустит данный фильтр сообщение, определяется состоянием регистров идентификаторов приема CIDAR0...CIDAR7 и регистров идентификаторов маскирования CIDMR0...CIDMR7. Функции этих регистров изменяются в зависимости от состояния битов IDAM1 и IDAM0 регистра CIDAC. Когда эти биты инициализируют режим двух 32-разрядных фильтров, каждый из этих фильтров сравнивает поразрядно часть регистра идентификатора с четырьмя байтами входного сообщения. Регистры маскирования идентификатора затем определяют, принимается или не принимается сообщение. На рис. 9.16 показаны восемь регистров идентификаторов приемников, на рис. 9.17 восемь регистров маскирования идентификаторов.

Данные в регистры CIDAR0-CIDAR7 и CIDMR0-CIDMR7 заносятся только в том случае, когда при инициализации в CMCRO установлен бит SFTRES. Для регистров CIDMR, логическая 1 сообщает, чтобы фильтры игнорировали данный бит кода идентификатора, в то время как логический 0 указывает, что должно быть установлено соответствие для данного бита. На рис. 9.18 показан случай, когда программируется режим двух 32-разрядных фильтров приема. На рисунке приведен как случай соответствия фильтру 0, так и случай соответствия фильтру 1 и отображены процессы при использовании стандартного формата идентификаторов (Bosch CAN, 2. 0A) и расширенного формата (Bosch CAN 2. OB). Напомним еще раз, что стандартный формат использует только 11-разрядные идентификаторы, а расширенный – 29-разрядные идентификаторы.

Когда состояние битов IDAM1 и IDAM0 регистра CIDAC соответствует режиму четырех 16-разрядных приемных фильтров, могут использоваться только 11- разрядные идентификаторы и биты RTR (в регистрах IDR0 и IDR1) согласуются с сообщениями формата CAN 2. 0A. Сообщениям в формате CAN 2. OB сопоставляются 14 значащих разядов идентификатора. На рис. 9.19 показан процесс идентификации приемников для режима с четырьмя 16-разрядными фильтрами.

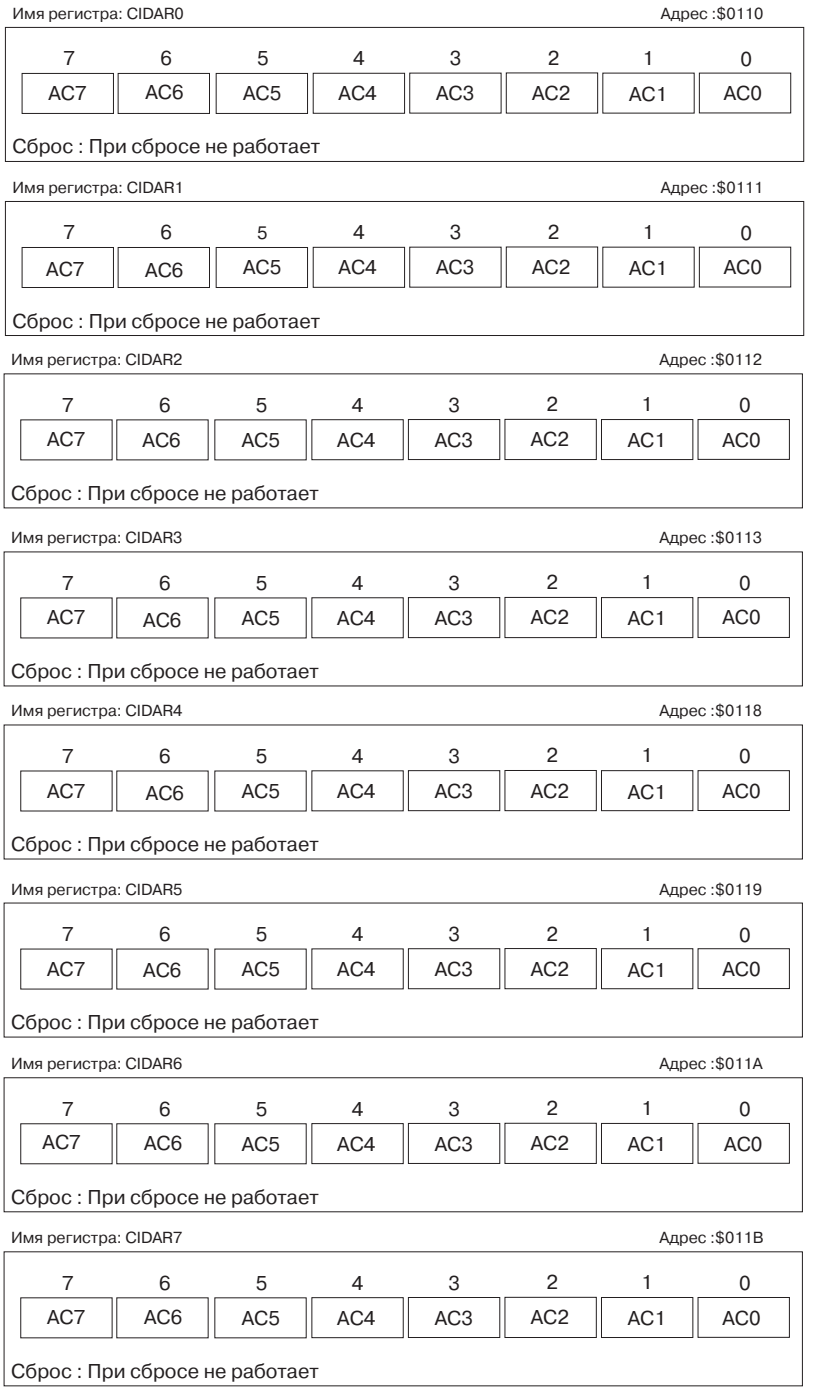


Рис. 9.16. *Формат регистров идентификаторов приемников CIDAR0...CIDAR7*

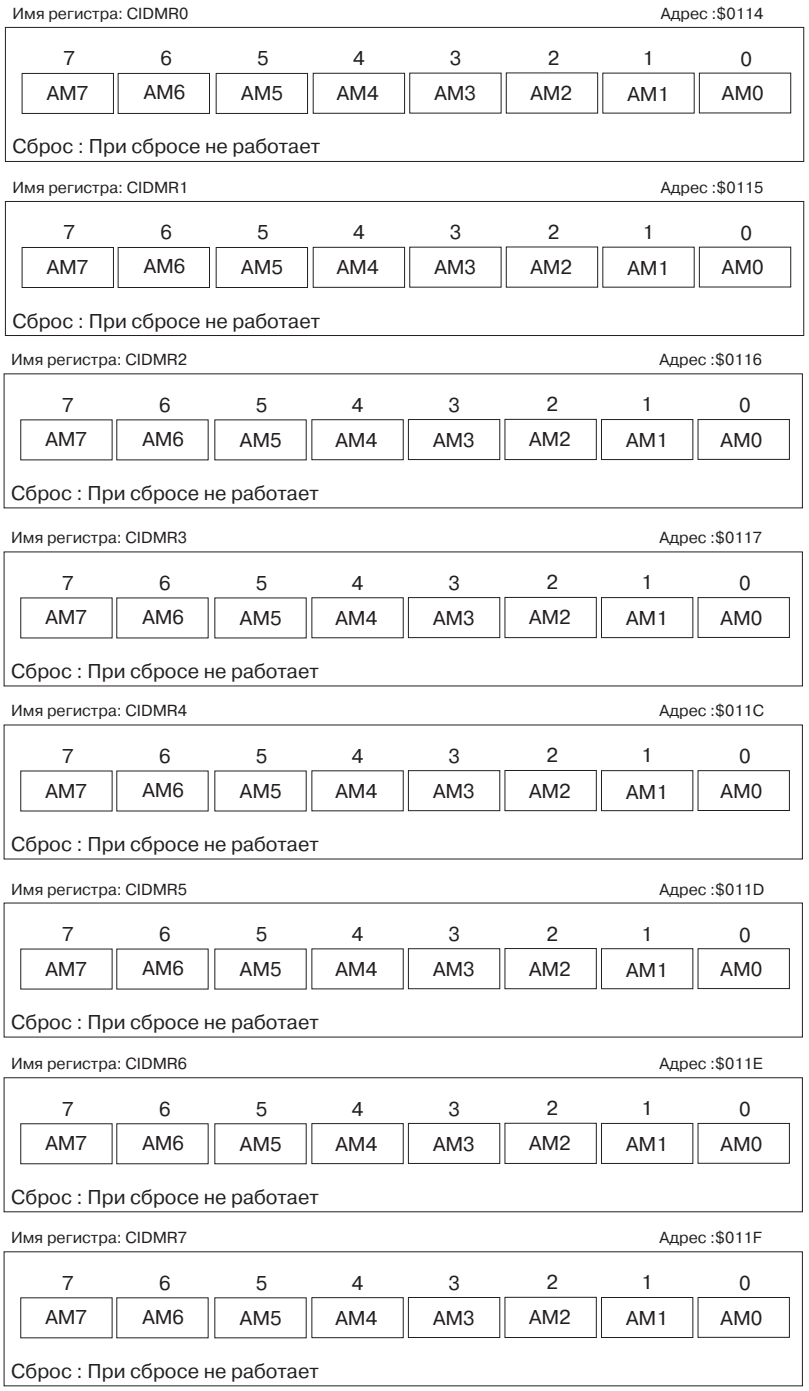


Рис. 9.17. *Формат регистров маскирования идентификаторов CIDMRO...CIDMR7*

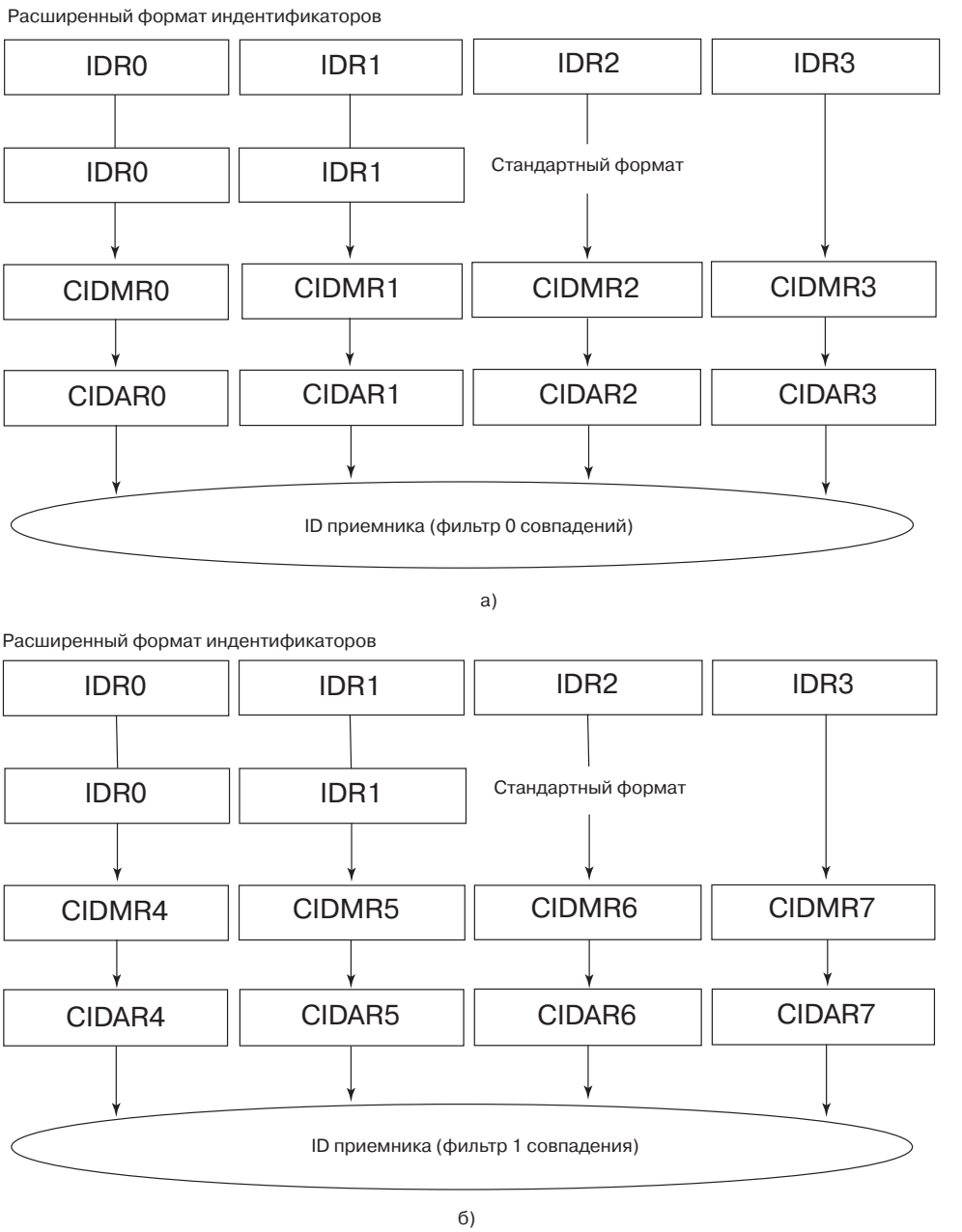


Рис. 9.18. *Процесс идентификации приемников: режим с двумя 32-разрядными фильтрами*

И наконец, когда биты IDAM1 и IDAM0 программируют работу в режиме восьми 8-разрядных фильтров, только восемь старших битов используются, чтобы пропустить или заблокировать входящие сообщения как в формате CAN 2.0A, так и в формате CAN 2.0B. На рис. 9.20 показан процесс идентификации для режима с восемью 8-разрядными фильтрами.

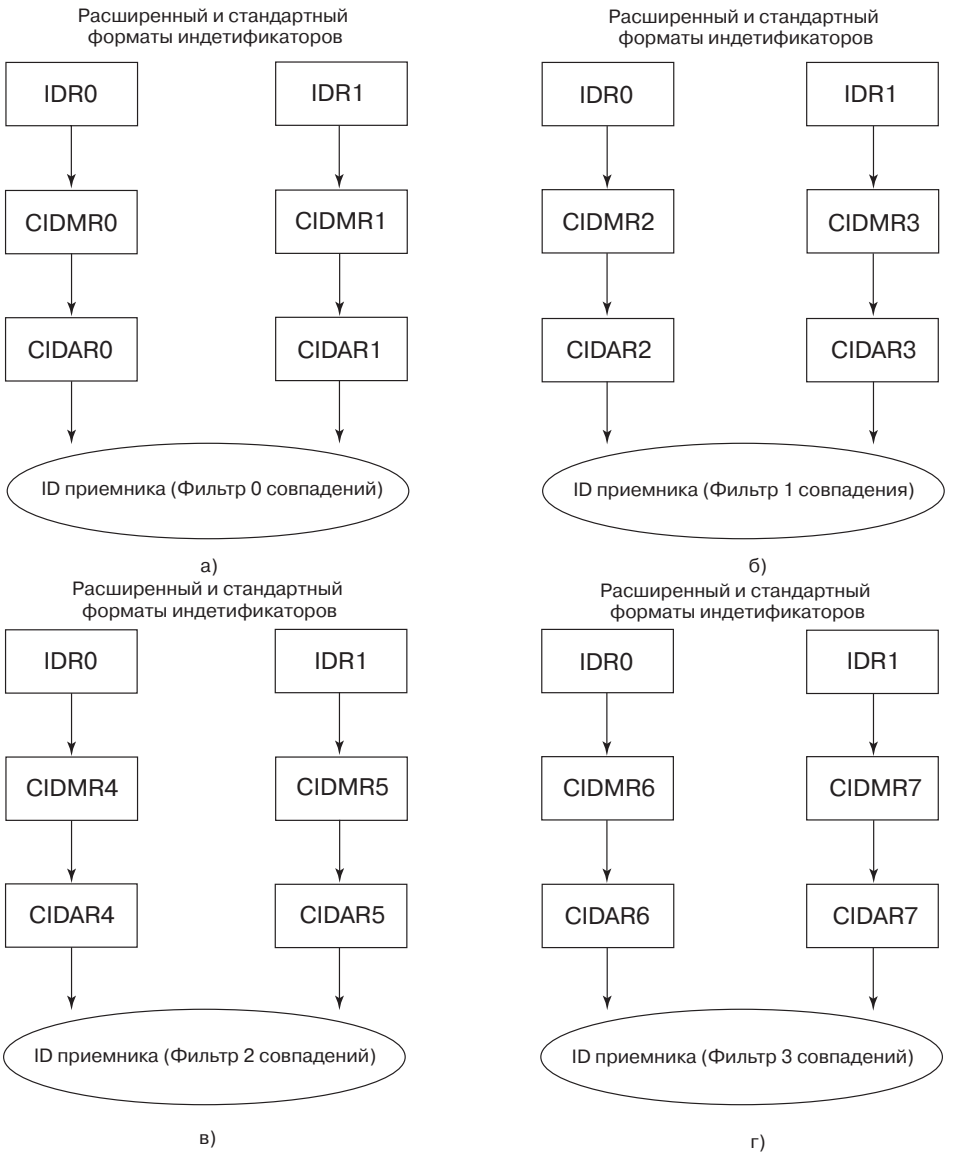


Рис. 9.19. *Процесс идентификации приемников: режим с четырьмя 16-разрядными фильтрами*

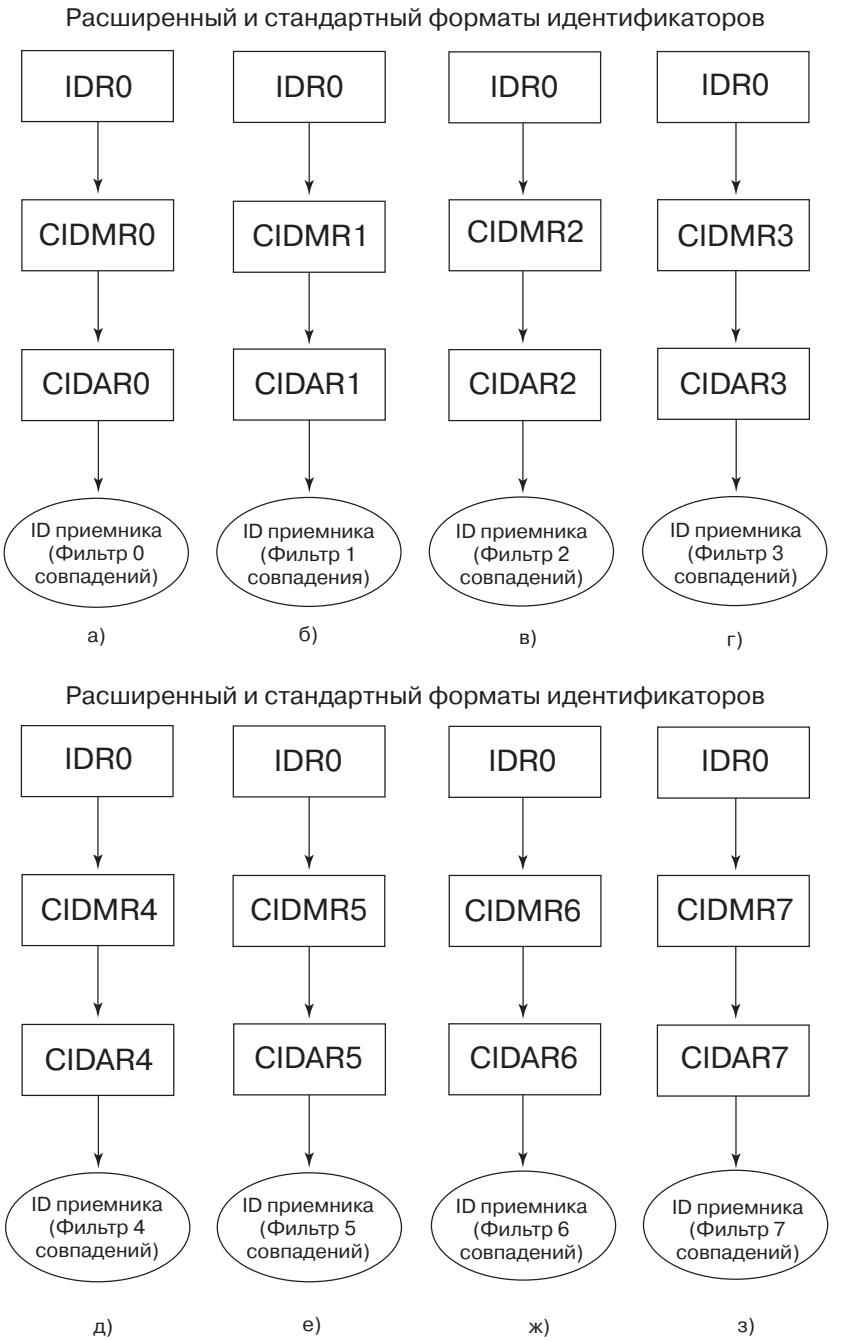


Рис. 9.20. *Процесс идентификации приемников: режим с восемью 8-разрядными фильтрами*

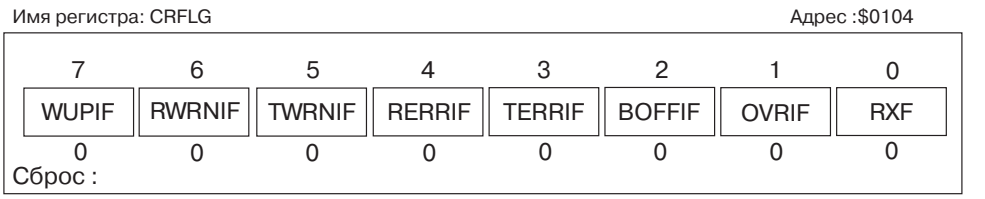


Рис. 9.21. *Формат регистра флагов приемников CRFLG*

Обсудим теперь еще два регистра, связанные с приемным модулем. Первый из них – регистр флагов приемника CRFLG (рис. 9.21). Каждый бит этого регистра, кроме RXF, может быть очищен посредством записи в него логической 1. За исключением флага полного буфера RXF, все другие флаги используются, чтобы вызывать ряд прерываний. Флаг прерывания по событию пробуждения WUPIF (бит 7) используется, чтобы обнаружить воздействие сети CAN, когда контроллер msCAN12 находится в спящем режиме: логический 0 в этом бите указывает, что не имеется никакого воздействия шины, логическая 1 показывает, что на вход приемника контроллера msCAN12 пришел запрос на прерывание спящего состояния. Флаг аварийного прерывания приема RWRNIF устанавливается, когда одновременно происходят следующие события: число ошибок превышает 96; флаг пассивного прерывания по ошибке приемника RERRIF сброшен; флаг пассивного прерывания по ошибке передатчика TERRIF сброшен; флаг прерывания «шина отключена» BOFFIF также сброшен. Низкое логическое состояние показывает, что нет аварийной ситуации на всех приемниках. Высокое логическое состояние показывает, что приемник msCAN12 находится в аварийном состоянии.

Флаг TWRNIF (бит 5) используется, чтобы проверить, аварийное состояние передатчика. Логическая единица, соответствующая аварийному состоянию, появляется, когда одновременно происходят следующие четыре события:

- число ошибок передачи превышает 96;
- бит RERRIF находится в состоянии нуля;
- бит TERRIF находится в состоянии нуля;
- бит BOFF1F находится в состоянии нуля.

Логический нуль TWRNIF указывает на отсутствие передатчика, находящегося в аварийном состоянии.

Флаг RERRIF (бит 4) используется, чтобы указать, находится или нет модуль msCAN12 в состоянии пассивной ошибки. Этот флаг устанавливается, когда число полученных ошибок находится в диапазоне от 128 до 255, и бит BOFFIF равен нулю. Установка бита RERRIF в 1 показывает, что модуль msCAN12 вошел в состояние пассивной ошибки приемника, в то время как очистка бита показывает, что мо-

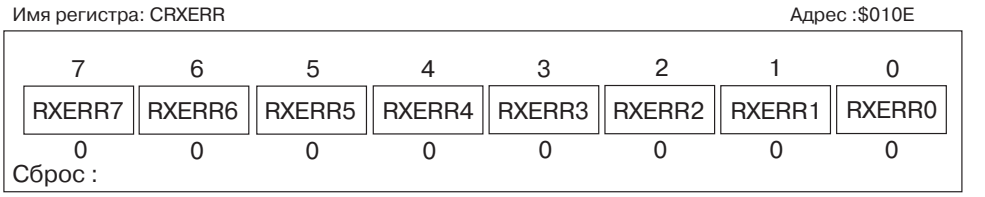


Рис. 9.22. *Формат регистра счетчика ошибок приемника CRXERR*

дуль msCAN12 не находится в этом состоянии. Точно так же установка флага TERRIF (бит 3) показывает, что модуль msCAN12 вошел в состояние пассивной ошибки передатчика. Модуль входит в это состояние, когда число ошибок передачи находится в диапазоне от 128 до 255 и флаг BOFFIF сброшен.

Разряд BOFFIF (бит 2) сообщает, находится или нет модуль msCAN12 в состоянии отключения от шины (bus-off): логическая 1 и 0 передают соответственно состояние отключения от шины или его отсутствие. Флаг прерывания из-за потери данных (OVR1F, бит 1) устанавливается, когда обнаружена потеря данных. Логический 0 в разряде этого флага показывает, что потери данных не обнаружено. Потеря данных происходит, если сообщение приходит на приемник, когда заполнен и основной и предварительный буферы приемника.

И, наконец, флаг заполнения приемного буфера RXF (бит 0) устанавливается, когда модуль msCAN12 желает «сообщить» процессорному ядру МК, что получено новое непрочтенное сообщение. ЦПУ сбрасывает этот флаг, чтобы перегрузить новое сообщение с предварительного буфера в основной. Логическая 1 указывает, что имеется готовое сообщение, а логический 0, что новых сообщений нет.

Последним регистром, связанным с приемным модулем, является регистр счетчика ошибок приемника CRXERR (рис. 9.22). Этот регистр подсчитывает число ошибок приема и предназначен только для чтения.

Мы заканчиваем этот раздел фрагментом программы, позволяющим конфигурировать приемный модуль для приема определенного набора сообщений. Текс программного фрагмента предполагает, что все имена используемых регистров были правильно определены в файле заголовка А мы лишь устанавливаем модуль приемника в режим 16-разрядного фильтра, чтобы получить данные со значением идентификатора $28E при использовании стандартного формата. Кроме того, мы программируем это, чтобы искать соответствие только по фильтру 0.

```
       :
       :
       :              /*включить модуль msCAN, инициализировать его  */
       :       /*ввести режим программного сброса: установить SFTRES =1*/
       :              /*конфигурировать  16-разрядный режим приемного */
                      /* фильтра: установить  IDAM1 = 1, IDAMO = 0*/
line i     CIDAC = $10;

line i+1   CIDMRO = $00;
                      /*задать значения первых 11 бит, RTR и IDE бит*/
line i+2   CIDMR1 = $07;
       :              /*установить все другие CIDMRx регистры to have*/
                      /*$00: игнорировать остальные биты*/
line j     CIDAR0 = $51;
                      /*%01010001 bits 10 - 3 of ID*/
line j+1   CIDAR1 = $C0;
                      /*%11000000 bits 2 - 0 of ID, RTR = 0, и */
                      /*IDE =0*/
       :
       :              /*ввести рабочий режим : установить SFTRES = 0
       :
       :              /*дождаться установки флага приема и просмотреть  */
                      /*соответствие фильтру 0 , чтобы считать данные*/
```

Подсистема прерывания контроллера msCAN12. Модуль msCAN12 имеет четыре типа прерываний: прерывание пробуждения (1), прерывания по ошибкам (6), прерывание при заполнении буфера приема (1) и прерывания при пустом буфере передатчика (3). В скобках указано число возможных прерываний, связанных с каждым из классифицированных типов.

Прерывание пробуждения допускается, когда установлен флаг WUPIE (бит 7) в регистре CRIER разрешения прерываний приемника msCAN12 (рис. 9.23). Если обнаружена активность шины, прерывание, связанное с пробуждением, инициализируется сразу после установки флага WUPIE.

Контроллер msCAN12 имеет шесть различных прерываний по ошибкам. Первое из эти шести – аварийное прерывание приема, которое допускается, когда в регистре CRIER установлен бит RWRNIE (бит 6). Если при этом обнаружено событие, вызывающее состояние аварийного прерывания приема, то устанавливается связанный с этим флаг RWRNIF в регистре CRFLG и выполняется соответствующая пользовательская программа обработки прерывания. Когда установлен бит TRWN1E (бит 5) регистра CRIER, состояние аварийного прерывания передачи устанавливает флаг TWRNIF в регистре CRFLG, и выполняется другая пользовательская программа обработки прерывания.

Третье и четвертое прерывания ошибки связаны с состояниями пассивной ошибки приемника и передатчика. Биты RERRIE и TERRIE в регистре CRIER – это локальные биты маскирования, которые разрешают пассивное прерывание по ошибке приемника и передатчика, соответственно. Запросы на прерывания генерируются флагами RERRIF и TERRIF регистра CRFLG.

Пятое прерывание ошибки связано с состоянием отключения от шины. Это прерывание, разрешаемое битом BOFFIE регистра CRIER, возникает, когда обнаружено подсистема приемника контроллера msCAN12 обнаружила отключение от CAN шины и установила бит BOFFIF в регистре CRFLG.

Последнее прерывание ошибки связано с потерей данных, которая происходит, когда в момент поступления значимого сообщения заполнены и предварительный и основной буферы приема. Прерывание разрешается битом OVRIF в регистре CRIER, ему соответствует флаг OVRIF в регистре CRFLG.

Прерывание по событию приема сообщения разрешается установкой бита RXFIE в регистре CRIER. Запрос на прерывание генерируется, когда приемником модуля msCAN12 получены достоверные данные, готовые для чтения ЦПУ МК. Это событие отмечается установкой флага заполнения приемного буфера RXF регистра CRFLG.

Три прерывания, создаваемые при пустых передающих буферах, разрешаются битами TXEIEO, TXE1E1, и TXEIE2 регистра управления передатчиком CRCR для буферов передатчиков 0, 1 и 2, соответственно. Когда буфер передатчика освобожден для

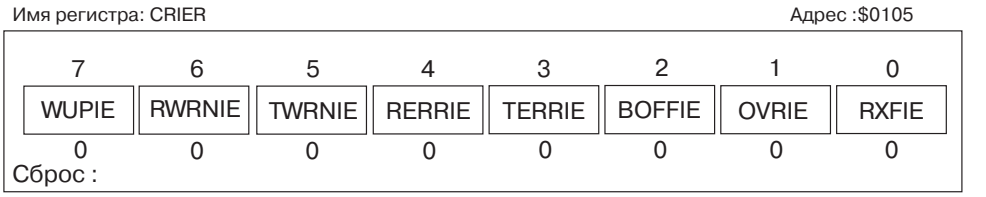


Рис. 9.23. Формат регистр разрешения прерывания приема CRIER

нового сообщения (пуст), устанавливается соответствующий флаг TXE0, TXE1 или TXE2 регистра CTFLG, и, если соответствующее прерывание разрешено, то выполняется соответствующая пользовательская программа обслуживания прерывания.

9.3.3. Проблемы синхронизации

Прежде, чем рассматривать систему из трех компонентов (передатчик, приемник и модуль прерывания), обсудим проблемы синхронизации, возникающие при работе контроллера CAN в составе МК 68HC12. Первой проблемой, с которой сталкиваемся разработчик, является выбор источника тактирования для контроллера CAN. Второй проблемой является конфигурация системы синхронизации приема битов с шины CAN.

Модуль msCAN12 рассчитан передачу информации по шине CAN со скоростью от 10 000 до 1 000 000 бит/с. Программист самостоятельно выбирает соответствующий источник синхронизации контроллера CAN с помощью бита CLKSRC в регистре управления CMCR1 (рис. 9.24).

Информация в регистр CMCR1 может быть записана только после установки бита SFTRES в регистре CMCR0 в состояние логической 1. Объясним вкратце назначение трех битов регистра CMCR1 прежде, чем продолжить обсуждение. Бит циклического режима самопроверки (LOOPB, бит 2) используется, чтобы конфигурировать msCAN12 контроллер для самопроверки функций шины CAN для данного контроллера CAN. Если этот бит очищен, CAN контроллер работает как узел сети CAN, подключенный к шине. Если этот бит установлен, активизируется цикл, самопроверки, вызывая передачу последовательности битов на приемник самого контроллера. В этом режиме логическое состояние на входе RxCAN игнорируется, а логическое состояние на выходе TxCAN устанавливается в 1 (рецессивное состояние). Все прерывания CAN могут происходить в этом режиме. Любой бит, посланный в течение ACK-Slot поля подтверждения кадра, игнорируется.

Флаг режима пробуждения WUPM (бит 1) регистра CMCR1 позволяет программисту устанавливать тип пробуждения, которое контроллер msCAN12 должен обнаружить перед выводом ЦП из спящего режима. Когда этот бит установлен, контроллер msCAN12 выведет ЦП из спящего режима только при появлении на шине CAN импульса определенной длительности. Бит WUPM устанавливается, чтобы предотвратить выход из спящего режима при любых сбоях или шумах, появившихся на CAN шине. Когда этот бит очищен, контроллер msCAN12 выводит ЦП из спящего режима при любом переходе шины из рецессивного в доминантное состояние.

Бит CLKSRC используется для выбора источника синхронизации контроллера msCAN12. Если этот бит сброшен, используется внешний тактовый генератор. Если этот бит установлен, в качестве источника синхронизации контроллера msCAN12 назначается внутренняя импульсная последовательность, частота которой вдвое превышает частоту ECLK (см. система тактирования).

Имя регистра: CMCR1 Адрес :$0101

	7	6	5	4	3	2	1	0
	0	0	0	0	0	LOOPB	WUPM	CLKSRC
Сброс :	0	0	0	0	0	0	0	0

Рис. 9.24. *Формат регистра управления CMCR1*

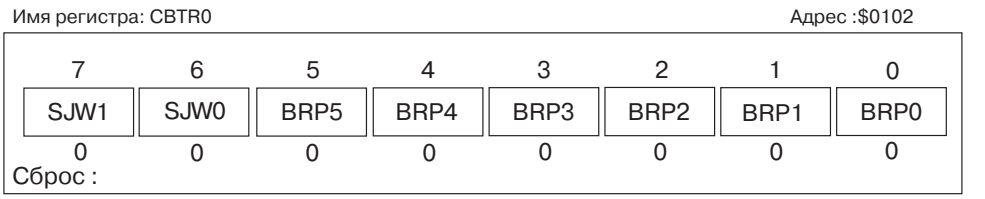


Рис. 9.25. *Формат регистра синхронизации шины CBTR0*

Регистр CBTR0 определяет число периодов тактовой частоты msCAN12, из которого будет состоять квант времени синхронизации. На рис. 9.25 показано содержимое CBTR0. Назначение битов 7 и 6 мы рассмотрим ниже. Остальные биты в регистре, BRP5-BRP0, используются для масштабирования источника синхронизации msCAN12, чтобы генерировать желательный квант времени синхронизации. Снова напомним, что регистр может быть установлен только тогда, когда установлен бит SFTRES в CMCR0. Таблица 9.1 показывает как могут быть конфигурированы биты BRP5-BRP0, чтобы установить желательный масштаб.

Контроллер msCAN12 использует квант времени в качестве временной базы, чтобы решить проблемы синхронизации. Как был упомянуто при обсуждении протокола CAN, для удобства синхронизации период времени присутствия каждого бита на CAN шине разделен на четыре сегмента: `sync_seg, prop.seg, phase.seg1 и phase_seg2`. В msCAN12 эти четыре сегмента отображены тремя сегментами: `sync_seg, time segment 1 и time segment 2`. При этом `sync_seg`, протокола CAN непосредственно отображается в `sync_seg` msCAN12, `prop.seg и phase.seg1` протокола CAN отображается в `time segment 1`, а `phase_seg2` протокола CAN отображается в `time segment 2` контроллера msCAN12.

Сегмент `sync_seg` используется, чтобы синхронизировать узлы CAN сети. Сегмент имеет длину, равную одному кванту времени. В течение этого периода контроллер msCAN12 ожидает фронт сигнала. Длительность `time segment 1` может составлять от 4 до 16 квантов времени, что определяется состоянием битов 0...3 регистра синхронизации шины CBTR1 (рис. 9.26). Длительность этого сегмента должна программироваться в соответствии с задержкой распространения передатчика и приемника, а также фазовой ошибкой фронта сигнала. Продолжительность сегмента `time segment 2` составляет от 2 до 8 квантов времени, и может также программироваться, с помощью битов 4, 5 и 6 регистра CBTR1. Таким образом, продолжительность времени передачи одного бита может составлять от 7 до 25 квантов времени в зависимости от установки битов в регистре CBTR1.

BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	Значение масштаба (P)
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
0	0	0	0	1	1	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	1	64

Таблица 9.1. *Конфигурация системы масштабирования кванта времени*

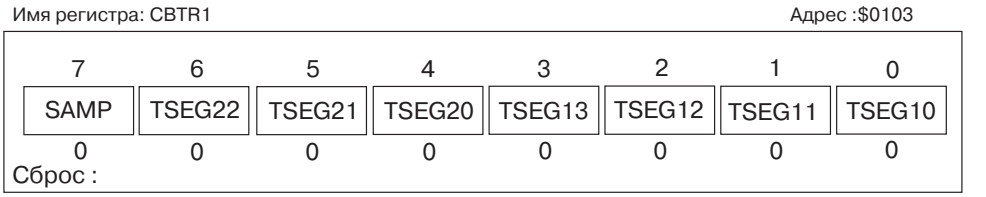


Рис. 9.26. *Формат регистра CBTR1*

Биты от 0 до 3 (TSEG13, TSEG12, TSEG11, и TSEG10) используются, чтобы установить длительность `time segment 1`, в соответствии с табл. 9.2. Биты от 4 до 6 (TSEG22, TSEG21 и TSEG20) используются, чтобы конфигурировать время `time segment 2`, как показано в табл. 9.3. Бит SAMP определяет, будет ли использоваться одна или три выборки, чтобы измерить принимаемый бит сообщения. Если бит SAMP установлен, используются три выборки, если он очищен, то только одна. Регистр CBTR1 может изменяться только при установленном бите SFTRES в регистре CMCR0.

Вернемся теперь к битам 7 и 6 регистра CBTR0. При синхронизации битов на

TSEG13	TSEG12	TSEG11	TSEG10	time segment 1
0	0	0	0	1 квант времени
0	0	0	1	2 кванта времени
0	0	1	0	3 кванта времени
0	0	1	1	4 кванта времени
⋮	⋮	⋮	⋮	⋮
1	1	1	1	16 квантов времени

Таблица 9.2. *Конфигурация длительности time segment 1*

TSEG22	TSEG21	TSEG20	time segment 2
0	0	0	1 квант времени
0	0	1	2 кванта времени
0	1	0	3 кванта времени
0	1	1	4 кванта времени
⋮	⋮	⋮	⋮
1	1	1	8 квантов времени

Таблица 9.3. *Конфигурация длительности time segment 2*

шине, используются биты скачка ресинхронизации (SJW1:SJW0), позволяющие уменьшить или увеличить ширину скачка ресинхронизации (число квантов времени). Таблица 9.4 показывает ширину скачка ресинхронизации, соответствующую различным значениям SJW1:SJWO.

SJW1	SJW0	Ширина скачка ресинхронизации
0	0	1 квант времени
0	1	2 кванта времени
1	0	3 кванта времени
1	1	4 кванта времени

Таблица 9.4. *Выбор ширины скачка ресинхронизации*

В заключение покажем дополнительную аппаратную возможность контроллера msCAN12 контроллера. Когда контроллер принял или передал корректное сообщение, генерируется импульс с длительностью, равной времени передачи одного бита. Этот импульс может быть послан на встроенный модуль таймерного интерфейса (TIM). Чтобы установить эту внутреннюю связь, используется бит TLNKEN в регистре CMCR0. Такое решение обеспечивает программисту необходимую гибкость. Модуль таймерного интерфейса может быть запрограммирован на определенное действие при появлении любого кадра приема или передачи, например, для каждого кадра может создаваться своя собственная метка времени.

9.3.4. Конфигурирование модуля msCAN12 для работы в сети

В этом разделе мы обьединим три подсистемы контроллера msCAN12 в единое целое и покажем последовательность действий по его инициализации.

До начала нашего обсуждения конфигурации сети, отметим, что шесть из восьми выводов физического порта CAN могут использоваться как универсальные входы/выходы порта общего назначения PORTCAN. На рис. 9.27 показано использование порта CAN для таких целей. Когда выводы со 2 по 7 находятся в режиме универсальных линийй ввода/вывода, следующие три регистра используются для записи и чтения данных с порта PORTCAN: регистр данных PORTCAN, регистр направленияпередачи DDRCAN, регистр управления PCTLCAN.

Регистр DDRCAN используется, чтобы программировать соответствующие выводы как входные или выходные. На рис. 9.28 показано назначение каждого бита регистра. Обратите внимание, что биты 0 и 1 зарезервированы как биты передачи и приема контроллера CAN и не могут быть использованы как универсальные выводы входов/выходов. Установка бита направления передачи данных в состояние логической 1 конфигурирует соответствующий вывод как выходной; установка его в состояние логического нуля – как входной. Регистр PCTLCAN, формат которого представлен на рис. 9.29, используется, чтобы разрешать или запрещать перевод выводов от 2 до 7 порта CAN в режим драйверов высокого или низкого уровня. Бит PUECAN используется, разрешать (при переводе в состояние логической 1) или запрещать (при переводе в состояние логического 0) режим драйвера высокого уровня. Бит RDPCAN выбирает конфигурацию, разрешающую (при переводе в сос-

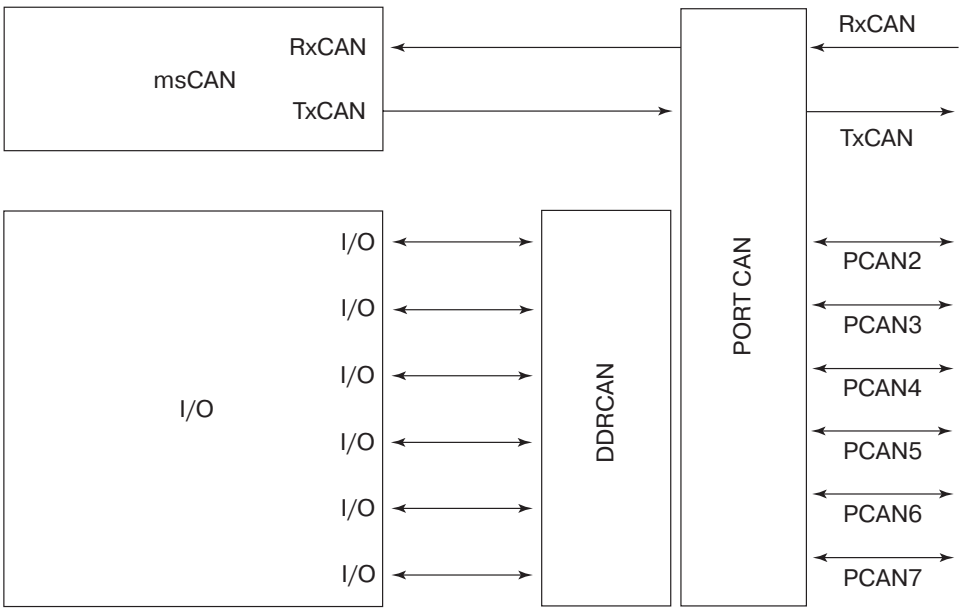


Рис. 9.27 *CAN портовая конфигурация*

Имя регистра: DDRCAN — Адрес :$013F

7	6	5	4	3	2	1	0
DDRCAN7	DDRCAN6	DDRCAN5	DDRCAN4	DDRCAN3	DDRCAN2	0	0
0	0	0	0	0	0	0	0

Сброс :

Рис. 9.28. *Формат регистра направления передачи DDRCAN*

Имя регистра: PCTLCAN — Адрес :$013D

7	6	5	4	3	2	1	0
0	0	0	0	0	0	PUECAN	RDPCAN
0	0	0	0	0	0	0	0

Сброс :

Рис. 9.29. *Формат регистра управления PCTLCAN*

тояние логической 1) или запрещающую (при переводе в состояние логического 0) режим драйвера низкого уровня.

И наконец, регистр PORTCAN содержит фактические данные (логические уровни) посылаемые или получаемые физическими выводами порта CAN. На рис. 9.30 показано содержимое регистра. В соответствии с установками в регистре DDRCAN, каждый бит используется передачи или приема данных из внешней среды. Обратите внимание, что, даже если модуль CAN конфигурирован для связи с сетью CAN,

Имя регистра: PORTCAN Адрес :$013E

7	6	5	4	3	2	1	0
PCAN7	PCAN6	PCAN5	PCAN4	PCAN3	PCAN2	TxCAN	RxCAN
0	0	0	0	0	0	0	0

Сброс : При сбросе не работает

Рис. 9.30 *Формат регистра данных порта PORTCAN*

этот регистр будет передавать или принимать данные, в зависимости от состояния битов 1 и 0 этого регистра.

Процедура полной установки. Далее мы приведем процедуру полной установки, позволяющей конфигурировать модуль msCAN12 для связи на сетью CAN. Мы обсудим также использование прерываний, связанных модулем CAN.

Два физических внешних вывода, RxCAN и TxCAN, используются для связи с сетью. Выводы RxCAN и TxCAN соответствуют битам 0 и 1 регистра PORTCAN. Приемопередатчик CAN, который используется, чтобы управлять логическими состояниями на сетевой шине, должен быть связан с двумя выводами и должен в свою очередь быть подключен к сети CAN. Типичными примерами таких приемопередатчиков являются микросхема MC33388 или PCA82C250.

После того, как 68HC12 будет правильно подключен к сети CAN, необходимо обеспечить правильную процедуру инициализации, позволяющую установить модуль msCAN12. Обсудим эту процедуру. Прежде, чем предпринять любые шаги по инициализации, модуль msCAN12 должен быть помещен в состояние мягкого сброса, путем установки в состояние логической 1 бита SFTRES (бит 0 регистра CMCR0). Как только бит SFTRES установлен, ЦП может задать рабочую конфигурацию msCAN12, с помощью следующих регистров, связанных с CAN:

- При сбросе значение по умолчанию в регистре CMCR0 равно $21, при этом устанавливаются биты CSWAI и SFTRES. По умолчанию также очищаются биты TLNKEN и SLPRQ. Модуль msCAN12 находится в состоянии мягкого сброса и готов к работе в нормальном режиме. Вход таймера связан с портом и модуль конфигурирован таким образом, чтобы в режиме ожидания отсутствовали импульсы синхронизации.
- Значение по умолчанию в регистре CMCR1 равно $00. Если не используется внешний генератор синхронизации, то бит CLKSRC должен быть переведен в состояние логической 1. Если применяется низкочастотный фильтр для подавления помех, которые могут восприниматься как фронты, должен быть установлен бит WUPM (бит1). Для самопроверки должен быть установлен бит LOOPB (бит2).
- Значение по умолчанию в регистре CBTR0 равно $00. Биты, устанавливающие скорость передачи в бодах и биты ширины скачка ресинхронизации должны программироваться согласно требованиям конкретного применения.
- Наряду с регистром CBTR0, должны быть запрограммированы биты регистра CBTR1, чтобы удовлетворять требования к соответствующим сегментам времени.
- Если используется прерывание, то должны быть установлены соответствующие биты прерывания в регистре CRIER или в регистре CTCR и написаны соответствующая программы обработки прерываний.

- Конфигурация фильтров идентификаторов должна быть запрограммирована с помощью регистра CIDAC.
- Соответствующие значения фильтров идентификаторов должны быть запрограммированы с помощью регистров CIDAR0-CIDAR7 и CIDMR0-CIDMR7.

Сразу после окончания процесса инициализации msCAN12 будет готов к связи. Если все прерывания запрещены, то один, два или все три буфера передатчиков должны быть заполнены перед тем, как начать передачу сообщения в сеть. Перед тем, как передать в сеть дополнительные сообщения должны быть проверены флаги TXE2, TXE1 и TXE0. Флаг RxF в регистре CRFLG должен быть опрошен, чтобы узнать, не получено ли допустимое сообщение.

В большинстве случаев, используется большое число прерываний CAN. Как минимум, для связи в сетях CAN должно использоваться прерывание передачи и прерывание приема. Чтобы разрешить прерывание передачи, должны быть установлены биты TXEIE2, TXEIE1, TXEIE0 в регистре CTCR, а соответствующие сервисные программы должны снова наполнить передающие буферы и очистить биты TXE2, TXE1, и TXEO в регистре CTFLG. Прерывание приема разрешается при установке бита RXFIE (Receiver Full Interrupt Enable) в регистре CRIER. Соответствующая сервисная программа должна очистить бит RxF в регистре CRFLG и обработать полученные данные. Еще раз повторим, что для осуществления любого допустимого прерывания бит I в регистре CCR должен быть очищен.

9.4. Различия между контроллерами msCAN в составе 68HC12 и HCS12

Одним из наиболее популярных вариантов микроконтроллеров семейства HCS12 является MC9S12DP256. В этом разделе мы опишем дополнительные свойства модуля контроллера msCAN12 в составе MC9S12DP256.

Основные изменения обсуждаемого модуля по сравнению с ранее рассмотренным состоят в следующем:

- число буферов в приемном модуле увеличивается с двух до пяти;
- осуществляется более жесткое программное управление передачей из трех передающих буферов;
- увеличивается число управляющих регистров
- добавлен режим «только прослушивание»;
- добавлена возможность установки временных меток для сообщений,
- удалена функция совмещения линий универсального порта с линиями входа/выхода контроллера CAN;
- уменьшено пространство памяти, используемое модулем msCAN (от 128 до 64 однобайтовых ячеек памяти).

Рассматрим кратко каждое из перечисленных изменений. Двухбуферная структура контроллера CAN в HC12 часто вызывает запаздывание реакции сетевых узлов, когда от шины поступают несколько сообщений. Чтобы исправить этот недостаток, модуль msCAN12 в составе MC9S12DP256 дополнили пятью структурами данных типа очереди (FIFO – «первым пришел, первым вышел»).

Число передающих буферов (3), не изменилось, но появилась дополнительная управляющая структура. В новых контроллерах msCAN12 один передающий буфер из трех выбирается посредством установки бита передачи TXi в регистре выбора передающего буфера (i – номер желательного буфера). Напомним, что ранее рассмотренном контроллере msCAN12 передающий буфер выбирается на основе информации о приоритете. Это изменение уменьшило физическое адресное пространство, выделенное для передающего буфера и упростило доступ к буферам передач.

В дополнение к функциональным изменениям в существующих девяти регистрах управления, контроллер MC9S12DP256 msCAN содержит три дополнительных регистра: регистр выбора передающего буфера msCAN (CANTBSEL), регистры старшего байта и младшего байта временной метки (TSRH:TSRL).

Режим «только для прослушивания» (LISTEN) позволяет программисту устанавливать контроллер CAN в рецессивное состояние на все время пока принимаются достоверные данные и кадры удаленного запроса. Контроллер CAN может быть запрограммирован на работу в таком режиме путем установки бита LISTEN в регистре управления 1 msCAN12. Дополнительная возможность присваивать временную метку каждому сообщению, упомянутая в предыдущем параграфе, позволяет программисту легко отслеживать передаваемые и получаемые сообщения. Исключение порта регистров CAN не позволяет использовать его для создания универсального порта входов – выходов (I/O). Однако исключение этого порта делает контроллер CAN более компактным, а для создания порта входов/выходов можно использовать другие порты контроллера MC9S12DP256.

В результате изменений, которые мы перечислили, карта памяти, контроллера msCAN12 в микроконтроллера MC9S12DP256 изменится в соответсвии с рис. 9.31.

9.5. Пример программирования контроллера msCAN

В этом разделе, мы рассмотрим простое применение, в котором два МК семейства 68HC12 объединяются в CAN сеть при помощи своих встроенных модулей msCAN12. Сначала мы покажем используемые в примере аппаратные средства (рис. 9.32), а затем представим программы, необходимые для работы обоих контроллеров. Для этого применения, мы выбрали отладочную плату Axiom CMD912 с микроконтроллером MC9S12DP256. Специальная плата была выбрана, чтобы воспользоваться преимуществами трансивера CAN, встроенного в ИС PCA82C259 компании Philips. Каждая плата должна выполнять собственную программу, чтобы реализовать простую связь между двумя msCAN модулями. Первая из программ, показанных ниже, является файлом заголовков (header file) 68hc9s12dp256.h, который содержит описания адресов регистров. Показана только фрагмент этого файла, относящийся к нашим программам CAN.

```
#define _REG_BASE 0
#define P(off) *(unsigned char volatile *)(_REG_BASE + off)
#define COPCTL _P(0x3C)          /*управление сторожевым таймером */
```

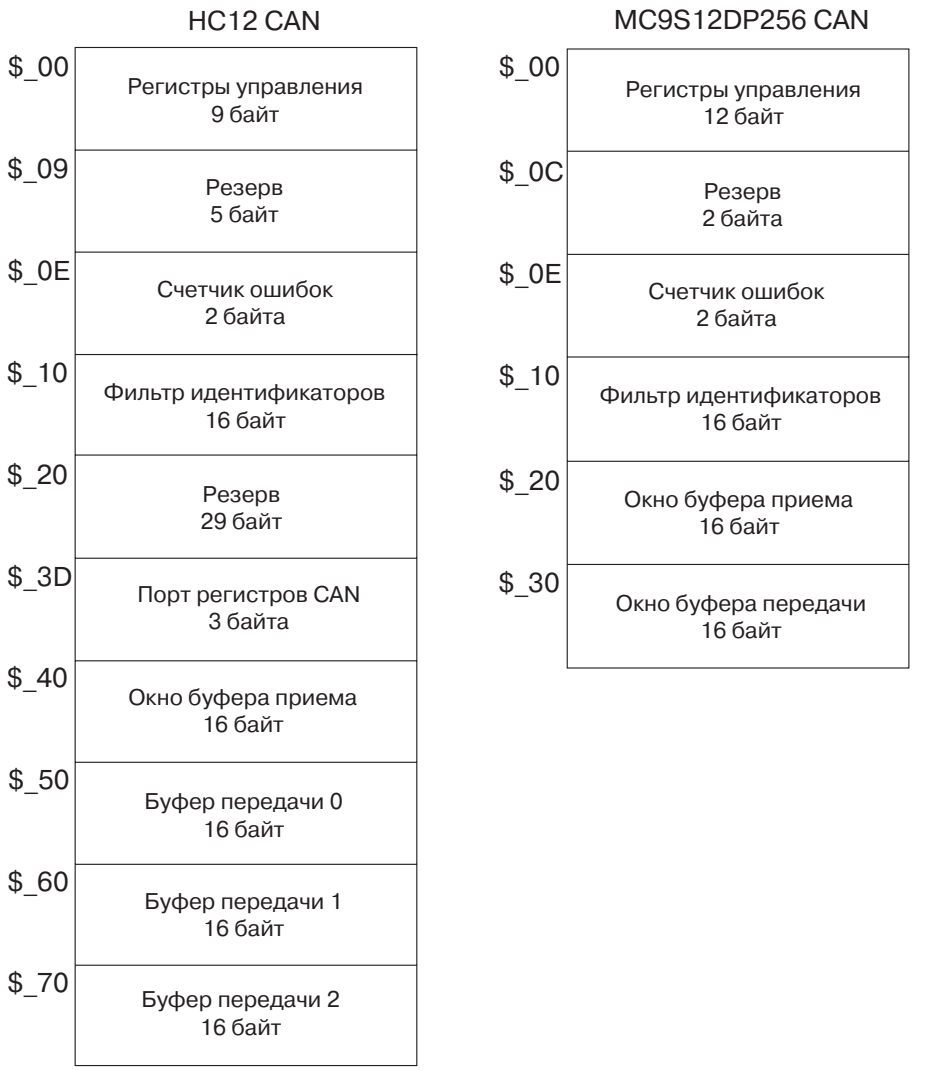


Рис. 9.31. *Карты памяти для контроллеров CAN в HC12 и MC9S12DP256*

```
#define CAN0CTL0 _P(0x0140)      /*управляющий регистр 0    */
#define CAN0CTL1 _P(Ox0141)/*CAN0 регистр управления 1              */
#define CAN0BTR0 _P{0x0142)/*CAN0 регистр синхронизации 0          */
#define CAN0BTR1 _P(0x0143)/*CAN0 регистр синхронизации 1          */
#define CAN0RFLG _P(0xO144)/*CAN0 флаги приема                      */
#define CAN0TFLG _P(0x0146)/*CAN0 флаги передачи                    */
#define CAN0TBEL _P(0x014A)/*CAN0 выбор передающего буфера         */
#define CAN01DM0 _P(0x0154)/*CAN0 регистр маскирования идентификаторов 0*/
#define CAN01DM1 _P(0x0155)/*CAN0 регистр маскирования идентификаторов 1*/
#define CAN01DM2_P0x0156)  /*CAN0 регистр маскирования идентификаторов 2*/
#define CAN0IDM3 _P(0x0157)/*CAN0 регистр маскирования идентификаторов 3*/
```

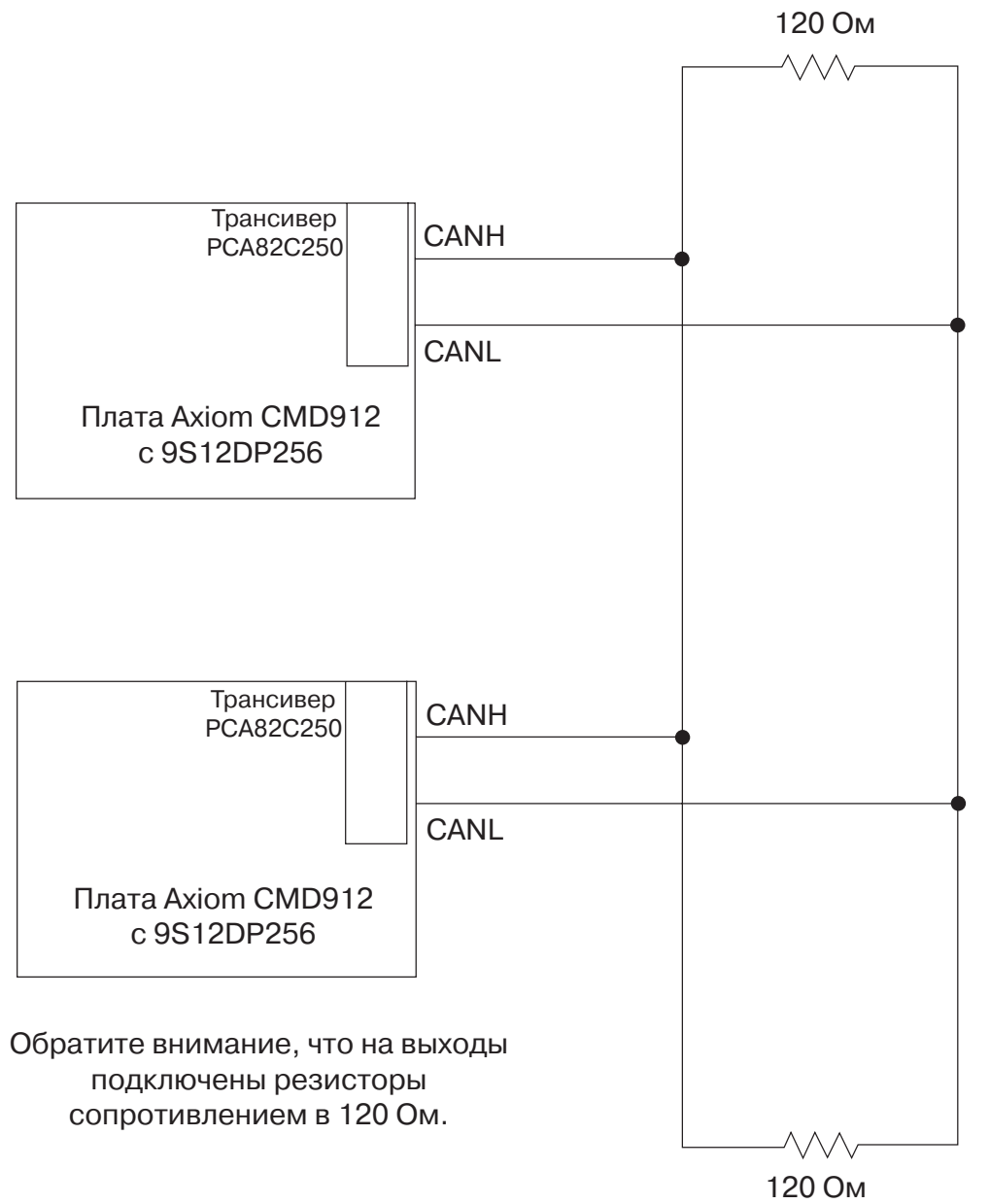


Рис. 9.32 *Схема включения а ппаратных средств для двух отладочных плат Axiom CMD912, связанныех в сеть CAN*

```
#define CAN0IDM4 _Pf0x015C)/*CAN0 регистр маскирования идентификаторов 4*/
#define CAN01DM5 _P(0x015D)/*CAN0 регистр маскирования идентификаторов 5*/
#define CAN0IDM6 _P(0x015E)/*CAN0 регистр маскирования идентификаторов 6*/
#define CAN0IDM7 _P(Ox015F)/*CAN0 регистр маскирования идентификаторов 7*/
#define CAHORXFG0 _P(0x0160)/*основной буфер RX CAN0                    */
#define CAN0RXFG1 _P(0x0161)/*основной буфер RX CAN0                    */
#define CAN0RXFG2 _P(0x0162) /*основной буфер RX CAN0                   */
#define CAN0RXFG3 _P(0x0163) /*основной буфер RX CAN0                   */
#define CAN0RXFG4 _P(0x0164) /*основной буфер RX CAN0                   */
#define CAN0RXFG5 _P(0xO165) /*основной буфер RX CAN0                   */
#define CAN0RXFG6 _P(0x0166) /*основной буфер RX CAN0                   */
#define CAN0RXFG7 _P(0x0167) /*основной буфер RX CAN0                   */
#define CAN0RXFG8 _P(0xO168) /*основной буфер RX CAN0                   */
#define CAN0RXFG9 _P(0x0169) /*основной буфер RX CAN0                   */
#define CAN0RXFGA _P(0x016A) /*основной буфер RX CAN0                   */
#define CAM0RXFGB _P(0xO16B) /*основной буфер RX CAN0                   */
#define CAN0RXFGC _P(0x016C) /*основной буфер RX CAN0                   */
#define CAN0ORXFGD _P(0x016D) /*основной буфер RX CAN0                  */
#define CAN0RXFGE _P(0xO16E) /*основной буфер RX CAN0                   */
#define CAN0RXFGF _P(0x016F) /*основной буфер RX CAN0                   */
#define CAN0TXFGO _P(0x017O) /*основной буфер TX CAN0                   */
#define CAN0TXFG1 _P(0xO171) /*основной буфер TX CAN0                   */
#define CAN0TXFG2 _P(0x0172) /*основной буфер TX CAN0                   */
#define CAN0TXFG3 _P(0x0173) /*основной буфер TX CAN0                   */
#define CAN0TXFG4 _P(0xO174) /*основной буфер TX CAN0                   */
#define CAN0TXFG5 _P(0x0175) /*основной буфер TX CAN0                   */
#define CAN0TXFG6 _P(0x0176) /*основной буфер TX CAN0                   */
#define CAN0TXFG7 _P(0x0177) /*основной буфер TX CAN0                   */
#define CAM0TXFG8 _P(0x0178) /*основной буфер TX CAN0                   */
#define CAN0TXFG9 _P(0x0179) /*основной буфер TX CAN0                   */
#define CAN0TXFGA _P(0xO17A) /*основной буфер TX CAN0                   */
#define CAN0TXFGB _P(0x017B) /*основной буфер TX CAN0                   */
#define CAN0TXFGC _P(Ox017C) /*основной буфер TX CAN0                   */
#define CAN0TXFGD _P(0xO17D) /*основной буфер TX CAN0                   */
#define CAN0TXFGE _P(0x017E) /*основной буфер TX CAN0                   */
#define CAN0TXFGF _P(0x017F) /*основной буфер TX CAN0                   */
/*****************************************************************/
```

Следующая программа выполняется контроллером первой платы, показанной в верхней части рис. 9.32. Эта программа инициализирует связь, а затем непрерывно посылает в сеть 8-байтовый блок данных ($01, $02, $03, $04, $05, $06, $07, и $08)

```
/*****************************************************************/
/* CANONE.C Эта программа запускает плату 68HC12 и сязываетее с другой  */
/* платой 68HC12 с помощью контроллера CAN                               */
/* Авторы: Даниэль Пак и Стив Барретт                                    */
/* Дата создания: 29 июля 2004                                           */
/*****************************************************************/

line   0   ttinclude   ' ' 68HC12DP256 .lv
line   1  void main()
line   2         {
line   3          COPCTL=0x00;    /*Выключить сторожевой таймер  COP*/
line   4                          /*установить модуль CAN */
line 5       CAN0CTL1= CAN0CTL1 | 0x80; /*разрешение для модуля  CAN  */
```

```
line 6        CAN0CTLl = CAN0CTLl & 0xEF; /*выключение режима  LISTEN*/
line 7        while ((CAN0CTLl | 0x01) == 0)
                                  /*режим инициализации CAN*/
Line 8        {
line 9        CAN0CTL0 = CAN0CTL0 | 0x01;
line 10       }
line 11       CAN0BTR0 = 0xCl;     /*установка бит синхронизации  CAN*/
line 12       CAN0BTR1 = 0xF7;
line 13       CANOCTL0 = CAN0CTL0 & 0xFE;/*выход CAN из режима инициализации*/
line 14       while ((CAN0CTL0 & 0x10) == 0){} /*ожидание синхронизации*/
line 15       CAN0TBEL = 0x01;     /*выбор передающего буфера 0 */
line 16                            /*установка передающего буфера      */
line 17       CAN0TXFG0 = 0xFF;
line 18       CAN0TXFG1 = 0xFF;
line 19       CAN0TXFG2 = 0xFF;
line 20       CAN0TXFG3 = 0xFE;    /*RTR = 0 для кадра данных          */
line 21       CAN0TXFG4 = 0x01;    /*сообщение                         */
line 22       CAN0TXFG5 = 0x02;
line 23       CAN0TXFG6 = 0x03;
line 24       CAN0TXFG7 = 0x04;
line 25       CAN0TXFG8 = 0x05;
line 26       CAN0TXFG9 = 0x06;
line 27       CANOTXFGA = 0x07;
line 28       CAN0TXFGB = 0x08;
line 29       CAN0TXFGC = 0x08;    /*спецификатор длины данных         */
line 30       CAN0TXFGD = 0x00;
line 31       while(1)
line 32       {
line 33       while ((CAN0TFLG & 0x01) == 0)
                                  /*ожидание флага окончания передачи */
Line 34      CAN0TFLG = CANOTFLG | 0x01;    /*очистка флага            */
line 35       }
line36     }                       /* конец основной программы       */
/*****************************************************************/
```

Команда на строке 3 выключает функцию сторожевого таймера COP контроллера. Команды на строках от 4 до 20 инициализируют контроллер msCAN12. Сначала, команда по строке 5 включает CAN контроллер. Команда на строке 6 выключает режим LISTEN, используемый для контроллеров, которые только прослушивают данные сетевого трафика не передавая никаких сообщения. Команды в строках с 7 по 10 используются, чтобы перевести контроллер CAN в режим инициализации. Сразу после инициализации, используются команды на строках 11 и 12, чтобы установить бит синхронизации CAN. Команда на строке 13 подготавливает контроллер CAN к работе с сетевым трафиком. Команда на строке 14 необходима, чтобы синхронизовать контроллер CAN с сетевым трафиком. Команда на строке 15 выбирает передающий буфер 0 для передачи информации, а команды на строках с 16-й по 30-ю готовят содержимое для передающего буфера. Отметим, что мы установили флаги SRR и IDE, выбрав тем самым расширенный формат, и, кроме того, очистили бит RTR, показав, что текущий буфер загружен кадром данных. Начиная со строки 31 до конца программы продолжается передача данных в сеть.

Ниже приведена программа, которая определяет работу второго МК, показанного в нижней части рис. 9.32.

```
/*****************************************************************
/* CANTWO.C Эта программа запускает плату 68HC12 и сязываетее с другой  */
/* платой 68HC12 с помощью контроллера CAN                               */
/* Авторы: Даниэль Пак и Стив Барретт                                    */
/* Дата создания: 29 июля 2004                                           */
*****************************************************************

line 0  #include ''68HC12DP256.h''
line 1 void main()
line 2   {
line 3 COPCTL=0x00;                    /*Выключить сторожевой таймер  COP */
line 4                                 /*установить модуль CAN           */
line 5 CAN0CTL1 = CAN0CTL1 | 0x80;     /*разрешение для модуля  CAN      */
line 6 CAN0CTL1 = CAN0CTL1 & 0xEF;     /*выключение режима  LISTEN       */
line 7 while ((CAN0CTL1 | 0x01) == 0)
                                       /*режим инициализации CAN       */
line 8   {
line 9       CAN0CTL0 = CAN0CTL0 | 0x01
line 10  }
line 11 CANOBTR0 = 0xCl;               /*установка бита синхронизации CAN*/
line 12 CAN0BTR1 =0xF7;
line 13 CAN01DM0 = 0xFF;               /*прием всех сообщений          */
line 14 CAN01DM1 = 0xFF;
line 15 CAN01DM2 = 0xFF;
line 16 CAN01DM3 = 0xFF;
line 17 CAN01DM4 = 0xFF;
line 18 CAN01DM5 = 0xFF;
line 19 CAN01DM6 = 0xFF;
line 20 CAN01DM7 = 0xFF;
line 21 CAN0CTLO = CAN0CTL0 & 0xFE; /*выход CAN из режима инициализации */
line 22 while ((CAN0CTL0 & 0x10) == 0){} /*ожидание синхронизации      */
line 23                                /*ожидание сообщения            */
line 24 while ((CAN0RFLG & 0x01) == 0){}
                                       /*ожидание флага сообщения      */
line 25 CAN0RFLG = CAN0RFLG | 0x01; /*очистка флага                    */
line 26 asm(''swi'');
line 27 }                              /* конец основной программы     */
```

Мы видим, что первое отличие этой программы от предыдущей – это спецификация для регистров маскирования приемника в строках с 13-й до 20-ю. Команды устанавливают все биты маскирования в состояние логической 1, игнорируя весь код, поступающий на соответствующие приемные регистры. Таким образом все сообщения с любым содержанием для четырех регистров идентификатора будут приняты CAN контроллером.

Основное различие между двумя программами начинается в строке 23. По команде в строке 24 МК ожидает заполнения приемного буфера, команда в строке 25 очищает флаг приемника. Команда в последней строке останавливает программу, после чего МК мы можем проверить получение данных, рассматривая содержание приемного буфера, размещенного в ячейках от $0160 до $016F.

Приведенные прикладные программы показывают простейший сценарий, в котором действуют только два узла CAN сети. Мы предельно упростили пример, чтобы помочь Вам освоить начальное программирование CAN контроллеров в составе

68HC12. Для простоты мы избегали использования любых прерываний и запрограммировали контроллеры CAN таким образом, чтобы они могли принимать сообщения с любыми идентификаторами битами.

9.6. Контроллер последовательного обмена BDLC

Ранние версии 68HC12, например MC68HC912B32 и MC68HC12BE32, не имеют модулей CAN. Вместо этого, контроллеры содержат другой модуль сетевой связи – контроллер связи байтов данных (byte data link controller – BDLC). Этот модуль Вы можете увидеть в составе МК MC68HC912B32 на рис. 1.3. Модуль контроллера BDLC был разработан для подключения МК семейства 68HC12 к информационной сети, которая использует протокол J1850 общества автомобильных инженеров (SAE).

Сети передачи данных класса B протокола SAE J1850 предназначены для последовательной передачи данных на скоростях до 125 кб/с. Модуль BDLC использует переменный формат длительности временного интервала передачи бита, помехоподвляющие фильтры, механизм определения коллизий и циклический контроль избыточности для контроля за сохранностью переданных данных. Сообщение протокола J1850 состоит из начального символа кадра, приоритета сообщения, идентификатора сообщения, фактических данных, байта контрольной суммы CRC и символа конца данных. Модуль может функционировать в трех режимах: режиме с отключенным питанием (1), режиме сброса (2) и рабочем режиме (3). Кроме того, модуль BDLC поддерживает три энергосберегающих режима: BDLC и ЦП в режиме ожидания (1), BDLC в режиме останова, ЦП в режиме ожидания (2), BDLC и ЦП в режиме останова (3).

Модуль BDLC 68HC12 состоит из конечного автомата, мультиплексора, двух приемных и двух передающих регистров данных. Пять регистров управления используются, чтобы конфигурировать модуль BDLC для выбора источника тактирования, скорости передачи информации, режима работы, кодирования битов, разрешения прерываний. Три дополнительных регистра (регистр порта управления DLC, регистр порта данных DLC и регистр порта направления передачи данных DLC) используются, чтобы конфигурировать порт BDLC как универсальный порт ввода -вывода.

Протоколы SAE J1850 и CAN конкурируют в области управления локальными контроллерными сетями. Текущая тенденция показывает, что CAN протокол приобретает большую популярность среди пользователей, занимающихся промышленным применением, стремясь покрыть весь сектор локальных контроллерных сетей.

9.7. Заключение по главе 9

В этой главе, мы рассмотрели основы связи между встраиваемыми микроконтроллерными системами, соответствующие протоколу локальных промышленных сетей Bosch CAN 2.0A и CAN 2.OB. Мы познакомились с аппаратными и программными средствами периферийного модуля последовательного обмена msCAN в составе микроконтроллеров семейства 68HC12, а также с более простым модулем последо-

вательного обмена BDLC 68HC12. Мы обсудили проблемы синхронизации, связанные с CAN протоколом, регистры, используемые в контроллерах msCAN12, алгоритмы программного обслуживания буферов для приема и передачи сообщений по CAN. Привели ряд простых демонстрационных программ по инициализации на прием и передачу модуля msCAN12.

9.8. Что еще почитать?

1. Motorola, Inc. «HC12-M68HC12B Family Advance Information, M68HC12B/D,» 2000.
2. Motorola, Inc. «CAN-Bosch Controller Area Network (CAN) Version 2.0,» Protocol Standard, BCANPSV2.0/D, Rev. 3, 1998.
3. Motorola, Inc. «The msCAN on the MC9S12DP256 Compared with the msCAN on the HC12 Family,» AN2011/D, Rev. 1, 01/2002.
4. Motorola, Inc. «Scalable Controller Area Network (msCAN) Interrupts,» AN2283/D, Rev. 0, 08/2002.
5. Motorola, Inc. «VPW J1850 Multiplexing and Motorola's Byte Data Link Controller (BDLC) Module,» 1998.

9.9. Вопросы и задания

Основные

1. Сравните распределенную систему управления систему из нескольких автономных встраиваемых систем.
2. Расшифруйте следующие аббревиатуры: WAN, LAN, SAN, CAN.
3. Дайте короткое определение понятия «протокол».
4. Сколько уровней ISO существует в протоколе CAN версии 2. 0/A?
5. Сколько уровней ISO существует в протоколе CAN версии 2. 0/B?
6. Какие биты являются доминантными и рецессивными в CAN протоколе?
7. Перечислите возможные типы кадров, передающихся по CAN шине.
8. Перечислите возможные режимы работы контроллера последовательлного обмена msCAN12 .
9. В контроллере последовательного обмена заполнены три буфера передатчика. Как контроллер msCAN12 решает, содержимое какого буфера передать первым?

Более сложные

1. CAN протокол не использует в сообщении адресов передающего и принимающего узлов. Как образом узел сети определяет, принимать сообщение, появившееся на шине, или нет?
2. Опишите механизм синхронизации приема бита, используемый в протоколе CAN с целью повышения надежности приема.

3. Опишите процесс передачи сообщения на шину CAN аппаратными и программными средствами контроллера msCAN12.
4. Опишите процесс, приема сообщения с шины CAN при использовании контроллера msCAN12.
5. Запишите на Си фрагмент программного кода, который загружает в буфер 0 подсистемы передатчика произвольное кодовое сообщение для контроллера CAN.
6. Запишите на Си фрагмент программного кода, который сначала опрашивает флаг состояния заполнения буфера 0 передатчика, а затем, когда буфер освободится, загружает в него новое произвольное сообщение.
7. Запишите на Си фрагмент программного кода, позволяющий конфигурировать подсистему приемника контроллера msCAN12 так, чтобы он получал любое сообщение с номерами идентификатора «2003» и «1995».
8. Запишите на Си фрагмент программного кода, который создает сообщение об ошибке, когда переполняется регистр счетчика ошибок приема. Используйте программу обработки прерываний, связанную с регистром счетчика ошибок, чтобы установить в передающий буфер 0 сообщение «Error».

Исследовательские

1. Нарисуйте схему аппартаного подключения для сети CAN с тремя узлами (A, B и C), использующую соответствующие приемопередатчики.
2. По техническому условию требуется, чтобы узел A передал сообщение с числовым значением узлу B, узел B добавил к этому значению 1 и переслал результат узлу C, а узел C еще раз добавил 1 к полученному значению и переслал полученный результат снова узлу A. Этот процесс должен быть периодическим. Узел A начинает весь процесс с нулевого начального значения. Нарисуйте блок-схему, выполняющую эту задачу.
3. Запишите на Си программы для всех трех узлов в предыдущей задаче.

Книги Издательского дома «ДМК-пресс» можно заказать в Торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: 123242, Москва, а/я 20 или по электронному адресу: post@abook.ru.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в Internet-магазине: www.abook.ru.
Оптовые покупки: тел. (495) 258-91-94, 258-91-95; электронный адрес abook@abook.ru.

С. Ф. Барретт, Д. Дж. Пак

Встраиваемые системы
Проектирование приложений
на микроконтроллерах семейства
68HC12 / HCS12
с применением языка C

Главный редактор Мовчан Д. А.
dm@dmk-press.ru

Перевод с английского Т. В. Ремизевич
Научный редактор Д. И. Панфилов
Дизайн М. М. Селеменив
Верстка В. М. Селеменив

Издательский дом «ДМК-пресс», г. Москва
(495) 505-10-80
www.dmk-press.ru
e-mail: books@dmk-press.ru

Дизайн и верстка издания:
ИПЦ «ДМК-Пресс»
(495) 540-04-18, e-mail: ipc@dmk-press.ru
http://ipc.dmk-press.ru

Подписано в печать 20.01.2007.
Печать офсетная.
Усл. печ. л. 52. Тираж 1000 экз.