Лабораторни упражнения с MSP430FR5739

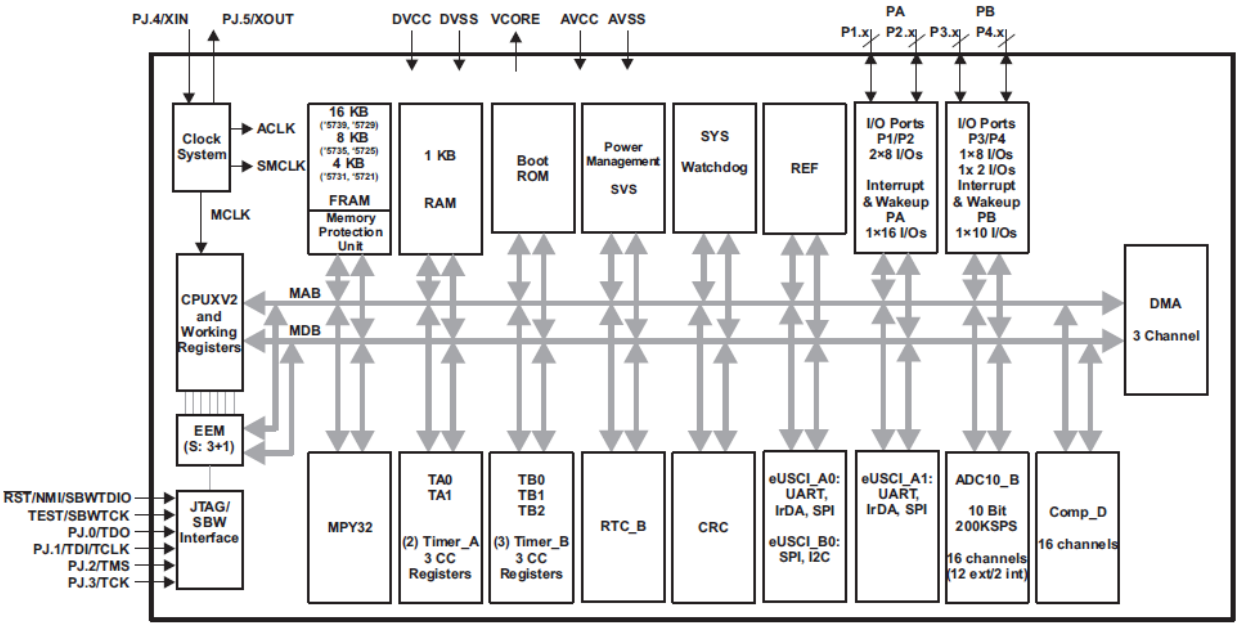
СЪДЪРЖАНИЕ

Лабораторно упражнение №1

“Запознаване със средата IAR Embedded Workbench и записване на примерна програма в MSP430FR5739”

1.1. Въведение. MSP430FR5739(RHA40) е 16-битов, 40-изводен RISC микроконтролер на фирмата Texas Instruments. Той работи с максимална тактова честота от 24 MHz и има 1 kB SRAM памет за данни. Програмната памет е 16 kB и е реализирана по технология Ferroelectric RAM (FRAM), докато по-старите контролери използват Flash памет. Предимствата на FRAM спрямо Flash са: 100 000 000 000 000 цикъла за запис (срещу 100 000 при Flash), около 250 пъти по-малка консумация, около 100 пъти по-кратки времена за достъп и около 10 пъти по-ниски нива на захранващото напрежение при запис (1,5 V). Големият брой цикли за запис позволява тя да бъде използвана като програмна и даннова памет едновременно.

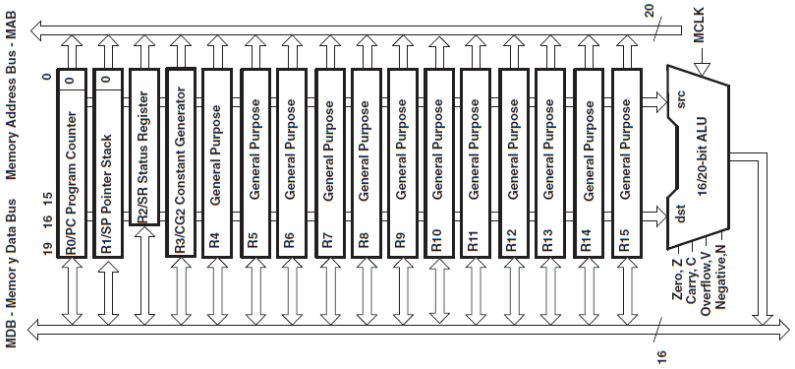
На **фиг. 1.1** е показана блокова схема на микроконтролера.



Фиг. 1.1 – Блокова схема на MSP430FR5739

Отделните модули са:

- CPU – 24-мегахерцов микропроцесор с 20-битова магистрала за адреси и 16-битова магистрала за данни (**фиг. 1.2**). Използват се 27 инструкции (+ 24 емулирани инструкции). Разполага с:
  - 12 регистъра с общо преназначение (R4 ÷ R15).
  - програмен брояч (PC, разположен в R0).
  - указател на стека (SP, разположен в R1).
  - регистър на състоянието (SR, разположен в R2).
  - регистър за генериране на константи (CG, разположен в R3) – генерира 6 често използвани константи в зависимост от адресацията на изпълняваната инструкция. Тези константи могат да служат като „операнд по подразбиране” и така инструкция с един операнд може да се превърне в инструкция с два операнда (емулирана инструкция).



Фиг. 1.2 – Блокова схема на микропроцесора, използван от микроконтролера MSP430FR5739

- JTAG – интерфейс за програмиране на чипа и четене на вътрешните регистри.
- EEM – Embedded Emulation Module – хардуерен модул за реализация на “breakpoints”. Това са точки на прекъсване, в които изпълнението на програмата спира и чрез JTAG може да се прочете състоянието на даден регистър. Накратко – използва се за улеснение при дебъгването на програмата.
- System Clock – модул за генериране на тактови честоти. Генерират се три вида честоти:
    - MCLK (Main Clock) – честота на микропроцесора
    - ACLK (Auxiliary Clock) – честота за периферните модули
    - SMCLK (Sub-Main Clock) – честота за периферните модули
- FRAM – програмна памет.
- RAM – даннова памет.
- Boot ROM – памет за съхранение на bootloader. Bootloader е специална програма, която помага за трансфер на същинската програма от персоналния компютър в програмната памет на микроконтролера.
- Power Management & SVS (Supply Voltage Supervisor) – модул за изработване на необходимите захранващи напрежения.
- SYS Watchdog – таймер, отброяващ интервал от време, при изтичането на което се подава сигнал за рестартиране на контролера. Програмистът трябва да заложи в програмата периодичното нулиране на този брояч. Така, ако някъде в програмата се получи „замръзване”, то броячът няма да се нулира и микроконтролерът ще се рестартира.
- REF – еталонно напрежение, използвано от някое периферно устройство (ADC или компаратор в случая).
- I/O Ports – входно-изходни портове. Това са регистри, достъпни за потребителя и свързани с изводите на микроконтролера. В литературата се отбелязват още като GPIO – General Purpose Input Output pins. С тях може да се управляват външни устройства.
- DMA (Direct Memory Access controller) – контролер за автоматично прехвърляне на данни (без намесата на микропроцесора) от един адрес на друг.
- Comp_D – аналогов компаратор.
- ADC10_B – 16-канално 10-битово АЦП.
- UART, IrDA, SPI, I2C – сериини синхронни (SPI, I2C) и асинхронни (UART, IrDA) интерфейси.

- CRC (Cyclic Redundancy Check) – модул за генериране на checksum.
- RTC_B – чаовник за реално време.
- Timer_A, Timer_B – 16-битови таймери.
- MPY32 – 32-битов хардуерен умножител.

Фърмуер (от англ. firmware) - софтуер, който се зарежда в постоянната памет на микроконтролера. Терминът се използва, за да не се бърка с приложния софтуер (от англ. application software), който се зарежда на персонален компютър. Микроконтролерните системи към днешна дата често се управляват от персонален компютър, което налага разграничаването фърмуер-приложен софтуер. В настоящите упражнения се разглежда само разработката на фърмуер.

Хардуерен дебъгер (от англ. hardware debugger) – устройство, с помощта на което развойната среда извършва следните действия (често обобщавани с израза дебъгване):

- зарежда фърмуера в паметта на микроконтролера;
- стартира изпълнението на фърмуера;
- трасира фърмуера (изпълнява програмата стъпка по стъпка, т.е. асемблерна инструкция след асемблерна инструкция или ред след ред от С програма);
- чете от или записва във всички регистри на микроконтролера;
- следи за точки на прекъсване (места в програмата, на които микропроцесорът спира изпълнението на инструкции);
- следи за регистри, които са си променили съдържанието.

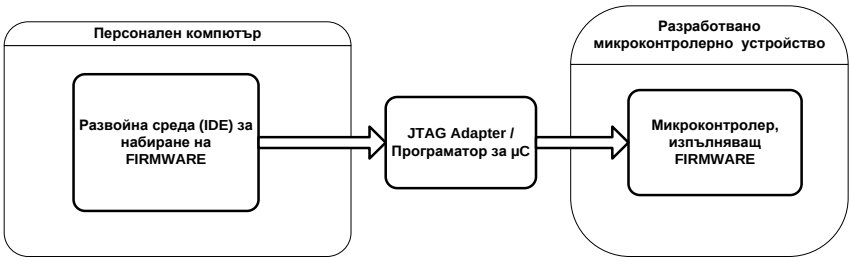
За да може дебъгерът да изпълнява всички тези функции, микроконтролерът трябва да съдържа хардуерени модули, които да изпълняват командите на дебъгера и да контролират микропроцесора. Такива са JTAG, SWD, Breakpoint, Watchpoint и др. модули. Повечето микроконтролери притежават поне JTAG или SWD.

Примери за хардуерни дебъгери са Keil ULINK, Texas Instruments ICDI, LPC Link и др. Някои от тях са реализирани като отделни устройства, други – включени в демо платка и неделими от нея.

Софтуерен дебъгер (от англ. software debugger) – програма, която следи действията на програмиста в развойната среда (натискане на бутони) и в зависимост от тях изпраща команди на хардуерния дебъгер. Той от своя страна изпраща команди на JTAG модула, който извършва желаното действие (например прочети даден регистър или спри микропроцесора).

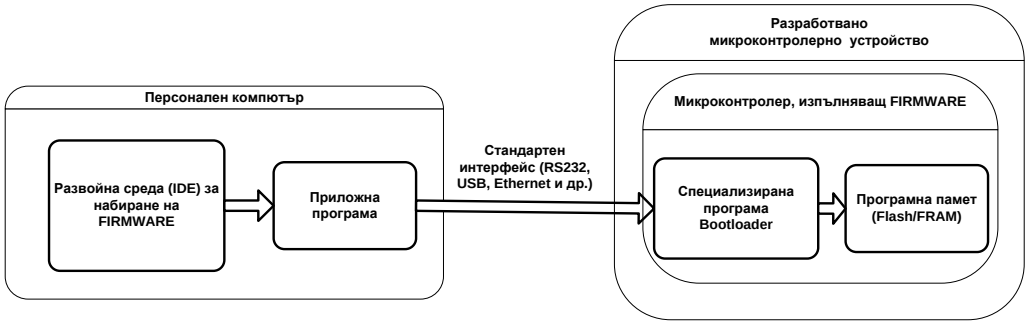
Пример за софтуерен дебъгер е програмата с отворен код GDB, която има версии за различни фамилии микропроцесори от Intel до ARM.

Използване на IAR Embedded Workbench IDE. Програмите за микроконтролери се разработват с помощта на развойни среди (IDE – Integrated Development Environment), работещи на персонални компютри. IDE представляват пакет от програмни продукти, обединени в един потрибителски интерфейс (прозорец). Повечето IDE включват текстови редактор, компилатор, линкер, асемблер, дебъгер и програма, генерираща изпълнимият код, често наричан фърмуер. Хардуерните методи за зареждане на фърмуера в микроконтролера са посредством JTAG адаптер или програматор (**фиг. 1.3**).

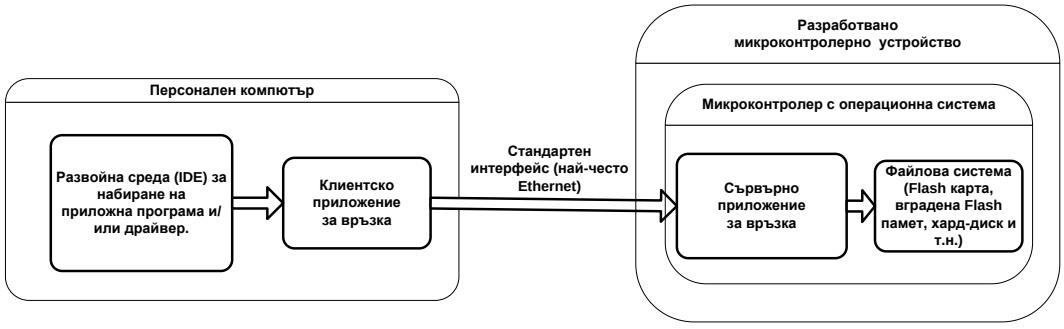


Фиг. 1.3 – Програмиране на микроконтролер чрез JTAG адаптер или програматор.

Съществуват и софтуерни методи за програмиране – чрез предварително зареден bootloader (**Фиг. 1.4 а**) или чрез копиране на файл във файловата система на микроконтролерно устройство с операционна система.



Фиг. 1.4 а) – Програмиране чрез bootloader.



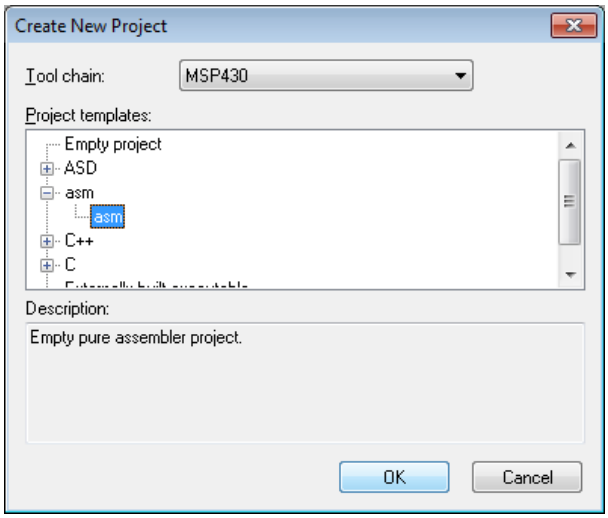
Фиг. 1.4 б) – Програмиране на устройство с операционна система.

В настоящото лабораторно упражнение ще се използва методът, показан на **фиг. 1.3**. Развойната платка е MSP-EXP430FR5739 (Experimenter's Board) на фирмата Texas Instruments и включва един микроконтролер MSP430FR5739, както и допълнителна периферия. Средата за програмиране е на фирмата IAR.

Тя може да се стартира като се щракне два пъти върху иконката IAR Embedded Workbench, след което ще се отвори нейният прозорец.

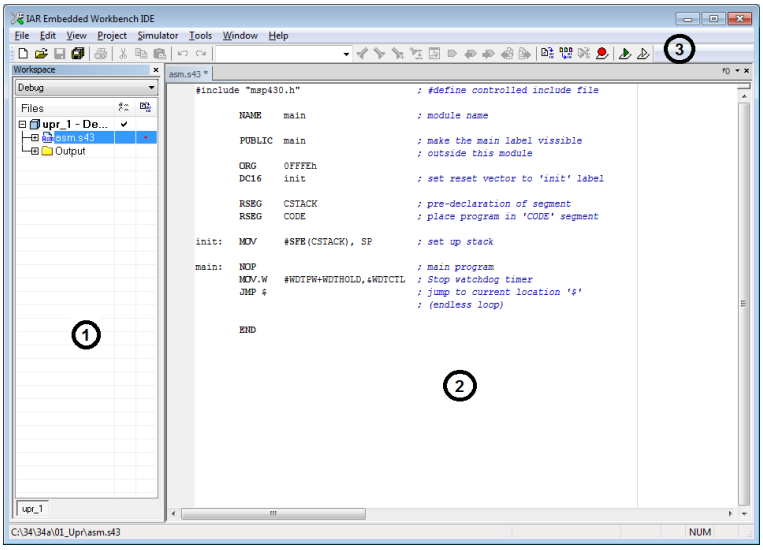
За програмиране на *асемблер* трябва да се премине през следните стъпки:

- Избира се File → New → Workspace
- Project → Create New Project…
- Tool chain → MSP430
- Project Templates → щраква се върху +asm → избира се asm
- Щраква се бутон OK (**фиг.1.5**).



Фиг. 1.5

- Указва се мястото, където ще се намира проекта.
- Дава се име на проекта и се натиска Save.
- Избира се номерът на микроконтролера от Project → Options → Category: *General Options* → tab Target → Device → избира се MSP430FR5739.
- В Category: *Debugger* → tab Setup → Driver → FET Debugger.
- Избира се файлът asm.s43 от дървото на проекта вляво на екрана (позиция 1, **фиг. 1.6**).
- Програмата на асемблер се въвежда в текстовия редактор вдясно (позиция 2, **фиг. 1.6**). За целта трябва да се изтрие редът за безкраен цикъл JMP $ и на негово място да се въведат новите инструкции.
- Програмата се асемблира с бутона Compile от toolbar-ът, показан на позиция 3 от **фиг. 1.6**. При първото натискане на този бутон средата IAR ще поиска въвеждане име на workspace (съдържа един или повече проекти, които имат някаква логическа връзка между тях). Указва се име по желание.
- Натиска се бутон Make. Ако програмата съдържа повече от един файл, то това е стъпката, в която останалите файлове ще се компилират/асемблират и свържат (link-нат) в един обектов файл. След това ще се генерира и изпълнимият файл, готов за зареждане в микроконтролера.



Фиг. 1.6

- Микроконтролерът се програмира с бутон Download and Debug. Преди натискането на този бутон макетът трябва да е включен с USB кабел към компютъра. При успешно зареждане на програмата ще се отвори нов toolbar с бутони, осигуряващи дебъг на програмата, докато се изпълнява в микроконтролера (**фиг. 1.7**).

Фиг. 1.7

Описанието на бутоните важи и при дебъгване на C/C++ програми.

Бутонът  служи за рестартиране на микроконтролера.

Бутонът  служи за спиране изпълнението на програмата.

Бутонът  служи за изпълняване на следваща инструкция (извикване на C/C++ функция) без да се влиза в подпрограми (в самата C/C++ функция).

Бутонът  служи за изпълняване на следваща инструкция (извикване на C/C++ функция) с влизане в подпрограми (в самата C/C++ функция).

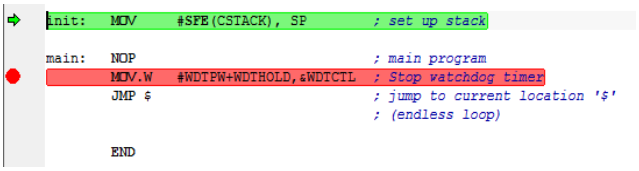
Бутонът  служи за излизане от C/C++ функцията, в която се намираме.

Бутонът  служи за придвижване изпълнението на програмата от настоящата инструкция (операция) до инструкция (операция), която е избрана чрез курсора на мишката.

Бутонът  служи за изпълнение на програмата от настоящото положение до края на програмата. Ако има поставени точки на прекъсване (breakpoints), то изпълнението ще спре при тях.

Бутонът  служи за изпълнение на програмата стъпка по стъпка, дори когато се изпълняват цикли с празно тяло (иначе дебъгерът ги прескача).

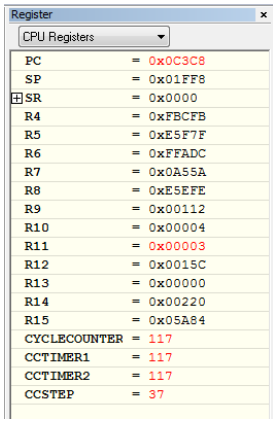
Поставяне на точка на прекъсване – щраква се с деснен бутон на реда, на който искаме изпълнението на програмата да спре. Избира се Toggle Breakpoint (Code). Срещу избрания ред ще се появи червена точка (**фиг. 1.8**).



фиг. 1.8

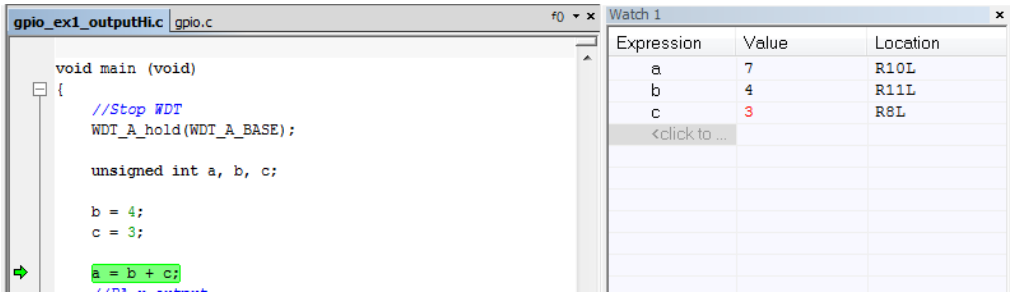
Докато изпълнението на програмата е спряно може да се наблюдават регистрите на микроконтролера благодарение на JTAG адаптера. В дебъг режим разнообразната информация може да се визуализира от менюто:

- View → Register – за наблюдение на всички регистри в микроконтролера. От падащото меню Register се избира периферен модул или микропроцесор. На **фиг. 1.9** са дадени моментните стойности на регистрите на микропроцесора от **фиг. 1.2**.



Фиг. 1.9

- View → Watch → Watch 1 → в полето Expression се записва име на променливата, която искаме да инспектираме. На **фиг. 1.10** е показана програма на C и наблюдаване на 3 променливи от целочислен тип a, b, c.

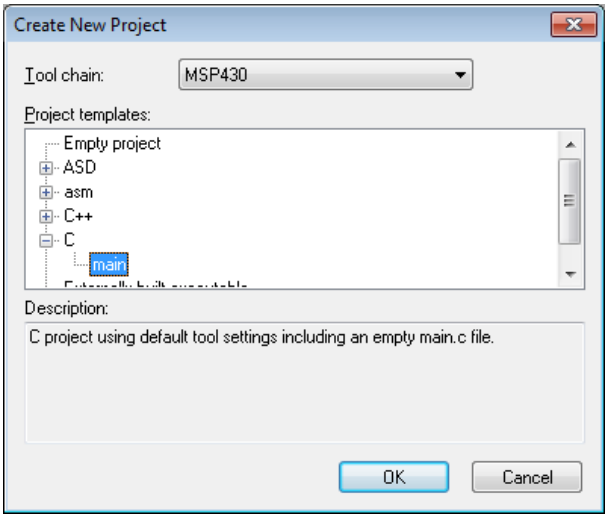


Фиг. 1.10

За програмиране на *C без използване на библиотеки* трябва да се премине през следните стъпки:

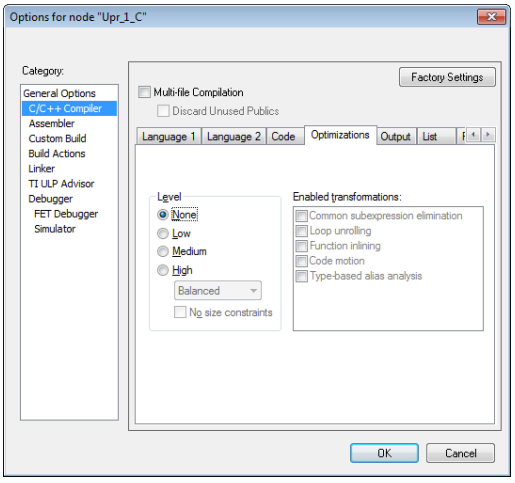
- Избира се File → New → Workspace
- Project → Create New Project…

- Tool chain → MSP430
- Project Templates → щраква се върху +C → избира се main
- Щраква се бутон OK (**фиг.1.11**).



Фиг. 1.11

- Указва се мястото, където ще се намира проекта.
- Дава се име на проекта и се натиска Save.
- Избира се номерът на микроконтролера от Project → Options → Category: *General Options*→ таб Target → Device → избира се MSP430FR5739.
- В Category: *Debugger* → таб Setup → Driver → Избира се FET Debugger.
- В Category: *C/C++ Compiler* → таб Optimizations → Level → None. Оптимизациите се изключват за лабораторните упражнения с цел по-ясно показване работата на микроконтролера. Натиска се OK (**фиг. 1.12**).



Фиг. 1.12

- От дървото на проекта се избира файлът main.c и програмата на С се въвежда след реда WDTCTL = WDTPW + WDTHOLD;
- Всички стъпки като компилирането, програмирането и дебъгването, описани за асемблер, също са приложими тук.

Компанията Texas Instruments предлага готови библиотеки на C, помагащи при инициализацията и използването на отделните периферни модули на MSP430FR5739. Благодарение на тях всяка опция се програмира чрез API функции, а не чрез директен регистров достъп. Последното означава, че нивото на абстракция в програмирането се увеличава, т.е. отдалечаваме се от програмирането на асемблер. Библиотеките са предоставени за свободно ползване и обединени в един пакет, наречен MSP430Ware. Редица микроконтролери от фамилията MSP430 може да ги използват. По подразбиране те се инсталират в директория driverlib_X_XX_XX_XX, където са включени и много полезни примери за първоначални тестове при разработката на фърмуер.

По-долу е разгледана програма, която спира watchdog таймера, инициализира порт J и установява изводи PJ.0, PJ.1, PJ.2 и PJ.3 във високо ниво. Представени са варианти на асемблер, на C без библиотеки и на C с използване на библиотеките MSP430Ware (start-up код не е показан).

Програма на асемблер:

```
main:   MOV.W   #WDTPW+WDTHOLD, &WDTCTL

        BIS.W   #0Fh, &PJDIR
        BIS.W   #0Fh, &PJOUT

        JMP $

        END
```

Програма на C без използване на библиотеки (директен регистров достъп)

```
void main( void )
{
  WDTCTL = WDTPW + WDTHOLD;

  PJDIR |= 0x0F;
  PJOUT = 0x0F;
}
```

Програма на C с използване на библиотеките MSP430Ware

```
void main (void)
{
   WDT_A_hold(WDT_A_BASE);

   GPIO_setAsOutputPin(GPIO_PORT_PJ, GPIO_PIN0 | GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3);
   GPIO_setOutputHighOnPin(GPIO_PORT_PJ, GPIO_PIN0 | GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3);
}
```

За програмиране на *С с използване на библиотеки* трябва да се премине през следните стъпки:

- Създава се С проект.
- Въвежда се програма на С, която използва API функции.
- API функциите, които могат да се използват са документирани във файла:

  MSP430F5xx_6xx_DriverLib_Users_Guide-X_XX_XX_XX.pdf

  където е даден списък с имената на API функциите и параметрите, които те приемат.
- Project → Options → C/C++ Compiler → Preprocessor → Additional Include directories: (one per line) → натиска се бутон „...“ (Browse) → натиска се полето <Click to add> и се указва пътя до директорията със сорс файлове на библиотеката. Например:

  MSHT_LAB \ 02_driverlib_1_60_02_01 \ driverlib \ MSP430FR57xx

- От дървото на проекта се щраква с десен бутон върху top-level директорията, избира се Add → Add Group и се указва произволно име (например driverlib).
- Избира се новосъздадената група driverlib от дървото на проекта и с десен бутон Add → Add Files… → указва се пътя до сорс файловете на MSP430Ware библиотеката. Например:

  MSHT_LAB \ 02_driverlib_1_60_02_01 \ driverlib \ MSP430FR57xx

  след което се избират всички файлове с CTRL + A → OK.
- Във файла main.c се включва #include директивата:

  #include "inc/hw_memmap.h"

  и допълнителните хедърни файлове за всяка една използвана периферия, например ако се използват Watchdog таймера и GPIO изводите:

  #include "gpio.h"
  #include "wdt_a.h"

- Ако е включена директивата #include "io430.h", то тя трябва да се изтрие.

1.2. Задачи за изпълнение.

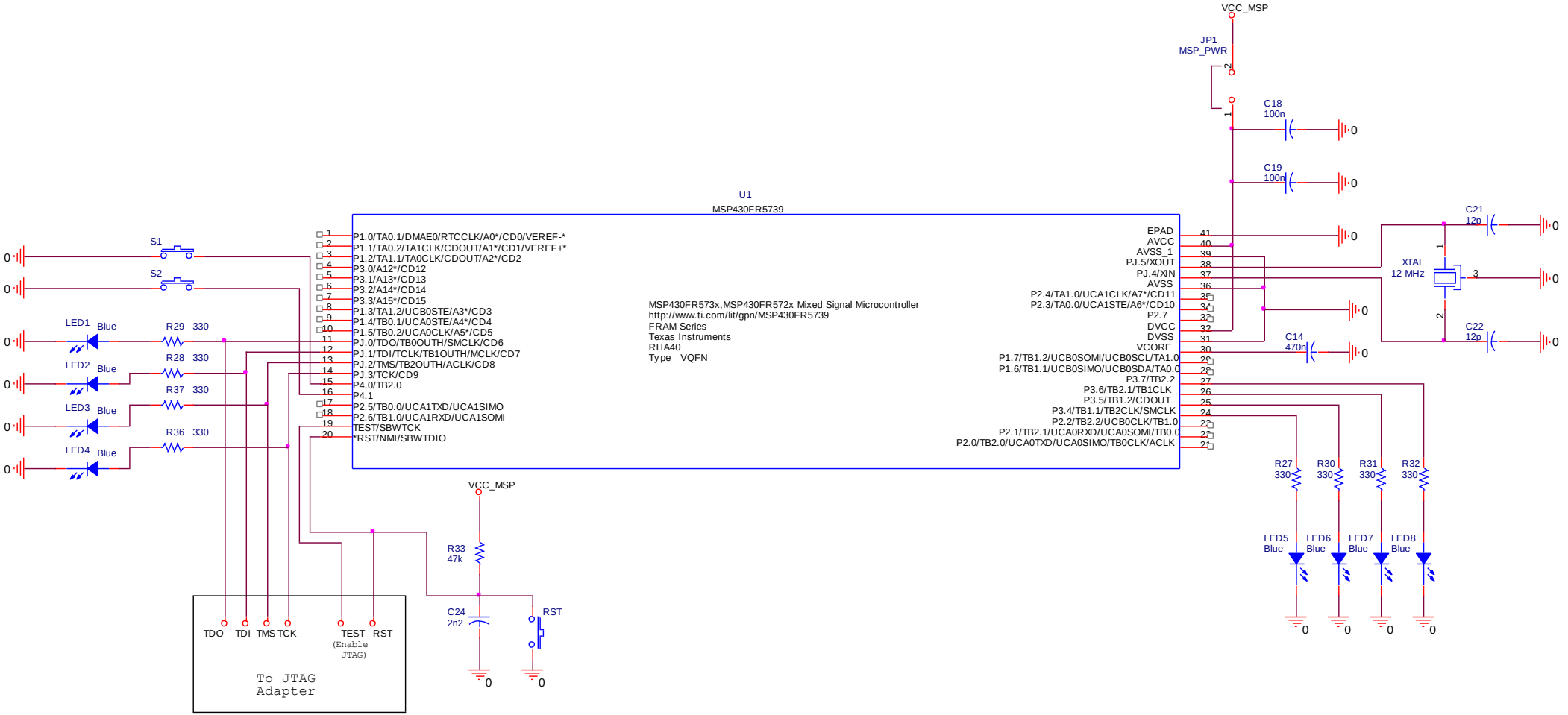
1.2.1. Да се създаде нов проект на Desktop-а в отделна директория с име 01_2_1_Lab, да се копира програмата на асемблер, която последователно включва само един от светодиодите LED1 ÷ 8 (ефект „бягаща точка”). Реализирано е софтуерно закъснение между всяко преместване на позицията, за да може да се наблюдава ефекта.

1.2.2. Да се създаде нов проект на Desktop-а в отделна директория с име 01_2_2_Lab, да се копира програмата на C, която извършва същите действия като програмата в предходната точка. Да се трасира програмата стъпка по стъпка. Да се наблюдават регистрите на микропроцесора и изходния порт. Да се постави точка на прекъсване. Да се наблюдава изменението на променливата i чрез View → Watch → Watch 1 → въвежда се i.

1.2.3. Да се създаде нов проект на Desktop-а в отделна директория с име 01_2_3_Lab, да се копира програмата на C, която извършва същите действия като програмата в предходната точка, но сега управлението на порта става чрез API функции от библиотеката MSP430Ware. Да се трасира програмата стъпка по стъпка. По време на дебъгване да се влезе в сорс кода на библиотеката (за целта трасирането да се извърши с бутон „Step into”).

1.2.4. Да се покаже съответствието между инструкциите на асемблерната програма (от задача 1.2.1.) и операциите на C програмата (от задача 1.2.2. и 1.2.3.).

1.2.5. Според вас коя програма е с най-голямо бързодействие?


VCC_MSP
JP1
MSP_PWR
C18
100n
C19
100n
U1
MSP430FR5739
MSP430FR573x,MSP430FR572x Mixed Signal Microcontroller
http://www.ti.com/lit/gpn/MSP430FR5739
FRAM Series
Texas Instruments
RHA40
Type VQFN
P1.0/TA0.1/DMAE0/RTCCLK/A0*/CD0/VEREF-*
P1.1/TA0.2/TA1CLK/CDOUT/A1*/CD1/VEREF+*
P1.2/TA1.1/TA0CLK/CDOUT/A2*/CD2
P3.0/A12*/CD12
P3.1/A13*/CD13
P3.2/A14*/CD14
P3.3/A15*/CD15
P1.3/TA1.2/UCB0STE/A3*/CD3
P1.4/TB0.1/UCA0STE/A4*/CD4
P1.5/TB0.2/UCA0CLK/A5*/CD5
P0.0/TDO/TB0OUTH/SMCLK/CD6
P0.1/TDI/TCLK/TB1OUTH/MCLK/CD7
P0.2/TMS/TB2OUTH/ACLK/CD8
P0.3/TCK/CD9
P4.0/TB2.0
P4.1
P2.5/TB0.0/UCA1TXD/UCA1SIMO
P2.6/TB1.0/UCA1RXD/UCA1SOMI
TEST/SBWTCK
RST/NMI/SBWTDIO
EPAD
AVCC
AVSS_1
PJ.5/XOUT
PJ.4/XIN
AVSS
P2.4/TA1.0/UCA1CLK/A7*/CD11
P2.3/TA0.0/UCA1STE/A6*/CD10
P2.7
DVCC
DVSS
VCORE
P1.7/TB1.2/UCB0SOMI/UCB0SCL/TA1.0
P1.6/TB1.1/UCB0SIMO/UCB0SDA/TA0.0
P3.7/TB2.2
P3.6/TB2.1/TB1CLK
P3.5/TB1.2/CDOUT
P3.4/TB1.1/TB2CLK/SMCLK
P2.2/TB2.2/UCB0CLK/TB1.0
P2.1/TB2.1/UCA0RXD/UCA0SOMI/TB0.0
P2.0/TB2.0/UCA0TXD/UCA0SIMO/TB0CLK/ACLK
C21
12p
C22
12p
XTAL
12 MHz
C14
470n
S1
S2
LED1
Blue
LED2
Blue
LED3
Blue
LED4
Blue
R29 330
R28 330
R37 330
R36 330
R27
330
R30
330
R31
330
R32
330
LED5
Blue
LED6
Blue
LED7
Blue
LED8
Blue
VCC_MSP
R33
47k
C24
2n2
RST
TDO
TDI
TMS
TCK
TEST
(Enable
JTAG)
RST
To JTAG
Adapter

Фиг. 1.13

Задача 1.2.1.

```
#include    "msp430.h"                          ;#define controlled include file
            NAME        main                    ; module name
            PUBLIC      main                    ;make the main label vissible
                                                ;outside this module
            ORG         0FFFEh
            DC16        init                    ;set reset vector to 'init' label
            RSEG        CSTACK                  ;pre-declaration of segment
            RSEG        CODE                    ;place program in 'CODE'
                                                ;segment
init:       MOV         #SFE(CSTACK), SP        ;set up stack
main:       NOP                                 ;main program
            MOV.W       #WDTPW+WDTHOLD,&WDTCTL  ;Stop watchdog timer
            BIS.W       #0x0F, &PJDIR           ;Конфигуриране на изводите
                                                ;като изходи
            BIS.B       #0xF0, &P3DIR           ;Конфигуриране на изводите
                                                ;като изходи
L0          MOV.W       #0x01, PJOUT            ;Инициализация за обхожда-
                                                ;не на първи квартет
            MOV.B       #0x00, &P3OUT
            CALL        #Delay
            MOV.B       #3, R14                 ;Променлива, указваща
                                                ;позицията на точката.
L1          RLA.W       &PJOUT
            CALL        #Delay
            DEC.B       R14
            JNZ         L1
            MOV.W       #0x00, &PJOUT           ;Инициализация за
                                                ;обхождане на втори квартет
            MOV.B       #0x10, &P3OUT
            CALL        #Delay
            MOV.B       #3, R14                 ;Променлива, указваща
                                                ;позицията на точката.
L3          RLA.B       &P3OUT
            CALL        #Delay
            DEC.B       R14
            JNZ         L3
            JMP         L0

Delay:      MOV.W       #65535, R15
L2          DEC.W       R15
            JNZ         L2

            RET
            END
```

Задача 1.2.2.

```
#include "io430.h"

void init()
{
  PJDIR |= 0x0F;
  P3DIR |= 0xF0;
  PJOUT = 0x00;
  P3OUT = 0x00;
}

void delay()
{
  volatile long i;
  for(i = 0; i < 65535; i++){ }
}

void rotate_light()
{
  static int i = 1;

  PJOUT = i & 0x0F;
  P3OUT = i & 0xF0;

  i<<=1;
  if(i > 0x80) i = 1;
}

void main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  init();

  while(1)
  {
     rotate_light();
     delay();
  }
}
```

Задача 1.2.3.

```
#include "inc/hw_memmap.h"
#include "gpio.h"
#include "wdt_a.h"

void init()
{
  GPIO_setAsOutputPin(GPIO_PORT_PJ, GPIO_PIN0 | GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3);
  GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN4 | GPIO_PIN5 | GPIO_PIN6 | GPIO_PIN7);
  GPIO_setOutputLowOnPin(GPIO_PORT_PJ, GPIO_PIN0 | GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3);
  GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN4 | GPIO_PIN5 | GPIO_PIN6 | GPIO_PIN7);
}

void delay()
{
  volatile long i;
  for(i = 0; i < 65535; i++){ }
}

void rotate_light()
{
  static int i = 0;

  if((i >= 0) && (i < 4))
  {
    GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN4|GPIO_PIN5|GPIO_PIN6|GPIO_PIN7);
    GPIO_setOutputLowOnPin(GPIO_PORT_PJ, GPIO_PIN0|GPIO_PIN1|GPIO_PIN2|GPIO_PIN3);
    GPIO_setOutputHighOnPin(GPIO_PORT_PJ, GPIO_PIN0<<i);
  }
  else
  {
    GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN4|GPIO_PIN5|GPIO_PIN6|GPIO_PIN7);
    GPIO_setOutputLowOnPin(GPIO_PORT_PJ, GPIO_PIN0|GPIO_PIN1|GPIO_PIN2|GPIO_PIN3);
    GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN4<<i-4);
  }

  i++;
  if(i > 8) i = 0;
}

void main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  init();

  while(1)
  {
     rotate_light();
     delay();
  }
}
```

Лабораторно упражнение №2
"Видове адресации и инструкции на микроконтролер MSP430FR5739"

2.1. Въведение. Всяка една програма на С за микроконтролери се преобразува в програма на Асемблер, а след това всяка една асемблерна инструкция се преобразува в двоичен код, който се зарежда в микроконтролера. Познаването на Асемблер за поне един микроконтролер е от особено значение в програмирането на вградени системи. Макар Асемблерите да са различни за различните микроконтролери, между тях има много общи черти. Ето и някой примери, в които познаването на Асемблер е от значение:

а) Понякога програмистът не може да предвиди как точно един ред от С програмата ще се „преведе" на Асемблер. Това води до грешки, чието отстраняване (debug) изисква преглед на „преведената" програма. Например празен for-цикъл от вида for(i = 0; I < 20; i++){ } може да се премахне от С-компилатора, който да счете тази конструкция за ненужна. Реално обаче програмистът може да я използва като софтуерно закъснение. Тогава единствено чрез поглеждане в „преведената" на Асемблер програма може да се види липсата на този код.

б) Оптимизация от гледна точка на бързодействие и/или размер на програмата може да се постигне чрез програмиране на Асемблер. Микроконтролерите обикновено разполагат с минимален хардуерен ресурс (в сравнение с настолните компютри) и комбинирането С/Асемблер е често срещано. Пример → инициализиращ код на Linux за вградени системи.

2.2. Видове инструкции. Всяка една инструкция се състои от код на операцията и операнди. Фирмата Texas Instruments дефинира три вида инструкции на микропроцесора MSP430:
Инструкция с два операнда (наричани още Формат I инструкции)
Пример: add R3, R4 → Събери стойностите на регистри R3 и R4 и запиши резултата в R4 върху старата стойност.

Инструкция с един операнд (наричани още Формат II инструкции)
Пример: RRA R3 → премести всички битове на числото в R3 с една позиция надясно като най-младшия бит запиши в STATUS регистъра на позиция Carry).

Инструкции за преход (наричани още Формат III инструкции)
Пример: jmp 0x1000 → иди на абсолютен адрес 0x1000.

Разни инструкции
Пример: <u>Хендлер на прекъсване:</u>
add R3, R4
rra R3
reti → Излез от хендлер на прекъсване

Прекъсванията и хендлерите на прекъсвания са обяснени в лабораторното упражнение за входно/изходните портове.

Формат I и II инструкциите имат наставка .b или .w, които означават работа със съответно 8-битови и 16-битови операнди . Така например инструкцията add може да се запише като add.b R3, R4 или add.w R3, R4. Ако не се укаже наставка, асемблерът по подразбиране слага .w наставка. Използването на наставки произлиза от факта, че част от периферните модули са разположени на адреси 0x00 ÷ 0xFF, а друга част на 0x100 ÷ 0x1FF. Първите изискват работа с „.b" инструкции, а вторите – работа с „.w" инструкции. Не може да се използва .w на периферия в обхвата 0x00 ÷ 0xFF, както и .b на периферия в 0x100 ÷ 0x1FF.

Микропроцесорът MSP430 има 16-битова адресна шина. Това води до ограничение на адресираната памет от 0 до 65535 адреса (или 0 ÷ 64 kB). За да може да се произведат микроконтролери с повече памет и периферия (периферията се адресира като памет), от Тексас Инструментс са проектирали разширен вариант на MSP430 и това е MSP430X. В него адресна шина е 20-битова, което позволява $0 \div 2^{20} = 0 \div 1048576$ (или 0 ÷ приблизително 1 MB). MSP430FR5739 е с MSP430X микропроцесор. За да се адресира модул или памет, разположен на адрес по-голям от 0xFFFF се използва наставката A след мнемониката на инструкцията, например BR 0xFFFF (16-bit) и BRA 0x10000 (20-bit).

2.3. Дисасемблер (Асемблерен листинг). Дисасемблер (disassembly), или още асемблерен листинг, е изходен файл, съдържащ програма на Асемблер, заедно с машинния код, съответстващ на всяка една инструкция и адреса, от който тя се изпълнява. На **фиг. 2.1** е показан дисасемблера на С програма. Байтовете,

```
e04:    b5f8        push    {r3, r4, r5, r6, r7, lr}
e06:    4e0b        ldr     r6, [pc, #44] ; (e34 <UARTwrite+0x30>)
e08:    460d        mov     r5, r1
e0a:    4604        mov     r4, r0
e0c:    1847        adds    r7, r0, r1
e0e:    42bc        cmp     r4, r7
e10:    d00d        beq.n   e2e <UARTwrite+0x2a>
e12:    f814 3b01   ldrb.w  r3, [r4], #1
e16:    2b0a        cmp     r3, #10
e18:    d103        bne.n   e22 <UARTwrite+0x1e>
e1a:    6830        ldr     r0, [r6, #0]
e1c:    210d        movs    r1, #13
e1e:    f7ff ffb1   bl      d84 <UARTCharPut>
```

Адрес на инстр. | Машинен код | Програма на Асемблер

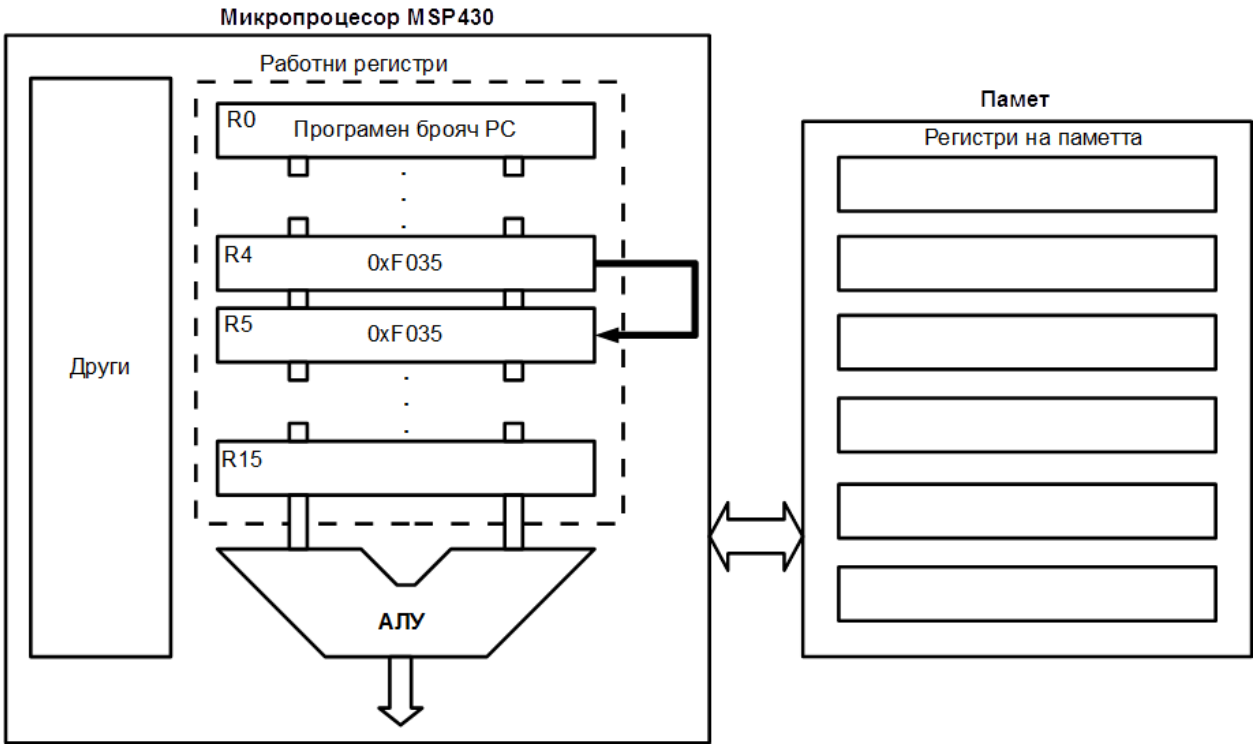
Фиг. 2.1

които се зареждат в паметта на контролера са показани в средната колонка. Сама по себе си тази информация не е достатъчна за успешното изпълнение на програмата от микропроцесора. Трябва да се знае всеки един байт на кой адрес се намира. Адресите са показани в най-лявата колонка. Всичко друго – програма на Асемблер, програма на С, програма на С++, програма на Java е абстракция за улеснение на програмиста.

2.4. Видове адресации. Операндите на една инструкция може да се извлекат по различен начин, от различни адреси в паметта или работните регистри на ядрото. Това обуславя различни видове т.нар. адресации или още режими на адресиране (Addressing modes). Техният брой варира при различните микропроцесори. За MSP430 са 7 броя. Трябва да се прави разлика между работните регистри на ядрото и външната (за ядрото) памет.

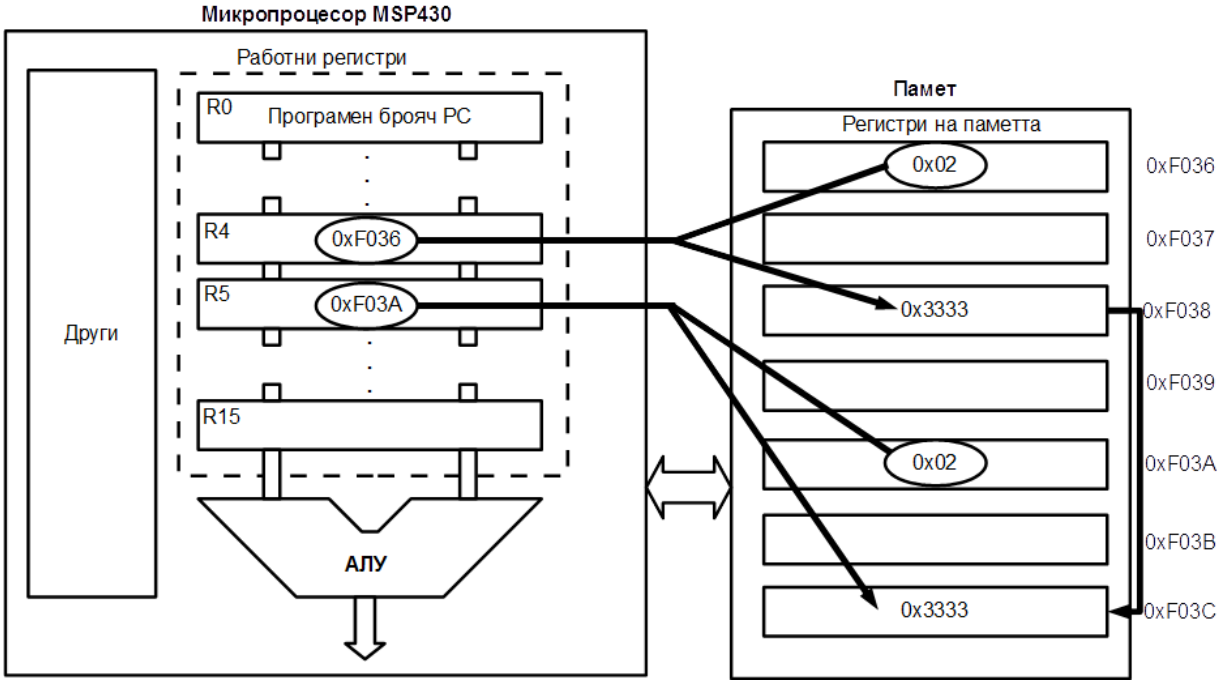
2.4.1 Регистрова адресация (Register) – съдържанието на работен регистър от ядрото е операнд на инструкцията.

Пример: mov.w R4,R5
Преди: R4 = 0xF035, R5 = 0x5555
След: R4 = 0xF035, R5 = 0xF035



2.4.2. Индексна адресация (Indexed) – съдържанието на работен регистър от ядрото + съдържанието на регистър от програмната памет (отместване) указват адреса (някъде в паметта на микроконтролера) на операнда.

Пример: mov.w 0x02(R4), 0x02(R5)
Преди: R4 = 0xF036, R5 = 0xF03A, MEM(0xF038) = 0x3333, MEM(0xF03C) = 0x5555
След: R4 = 0xF036, R5 = 0xF03A, MEM(0xF038) = 0x3333, MEM(0xF03C) = 0x3333

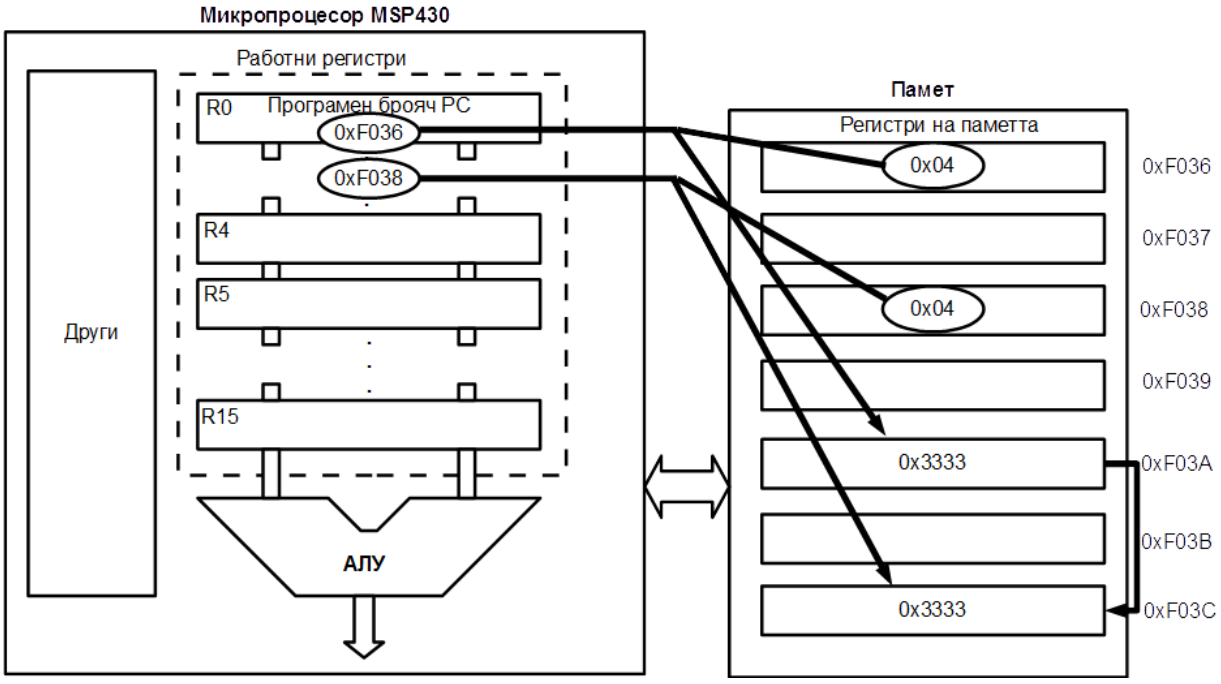


2.4.4. Относителна (Symbolic) адресация – съдържанието на регистър от програмната памет (отместване) + стойността на програмния брояч PC указват адреса на операнда.

Пример: mov.w Label1,Label2 (етикетите в Асемблер са относителни, или още условни адреси, които се заменят с абсолютни след линкването на програмата)

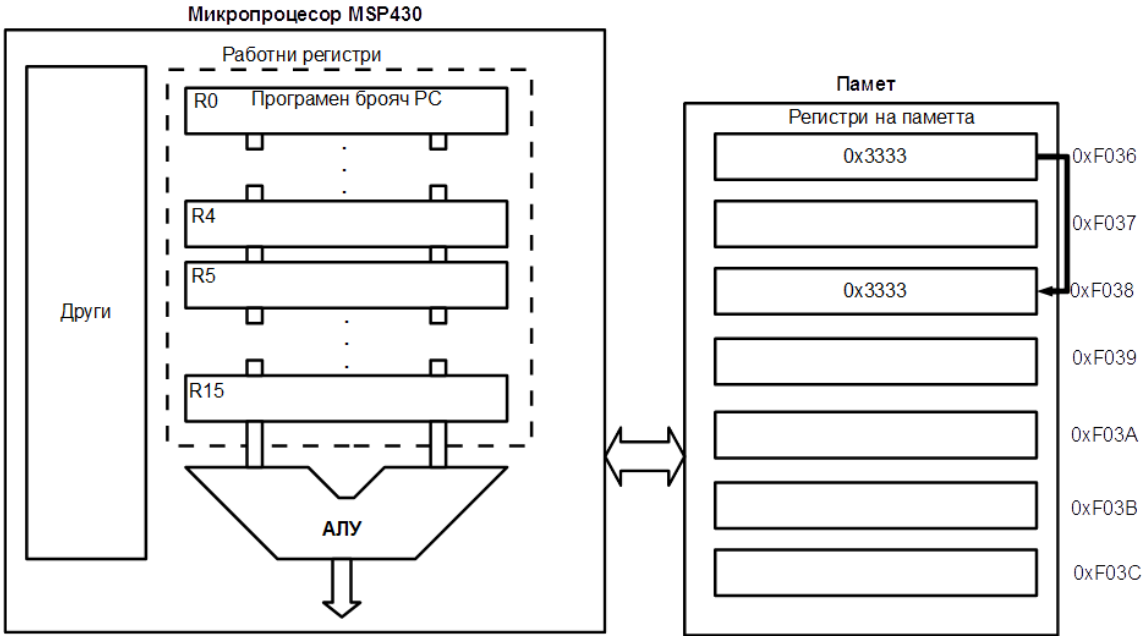
Преди: PC = 0xF036, MEM(0xF03A) = 0x3333, MEM(0xF03C) = 0x5555

След: PC = 0xF03A, MEM(0xF03A) = 0x3333, MEM(0xF03C) = 0x3333



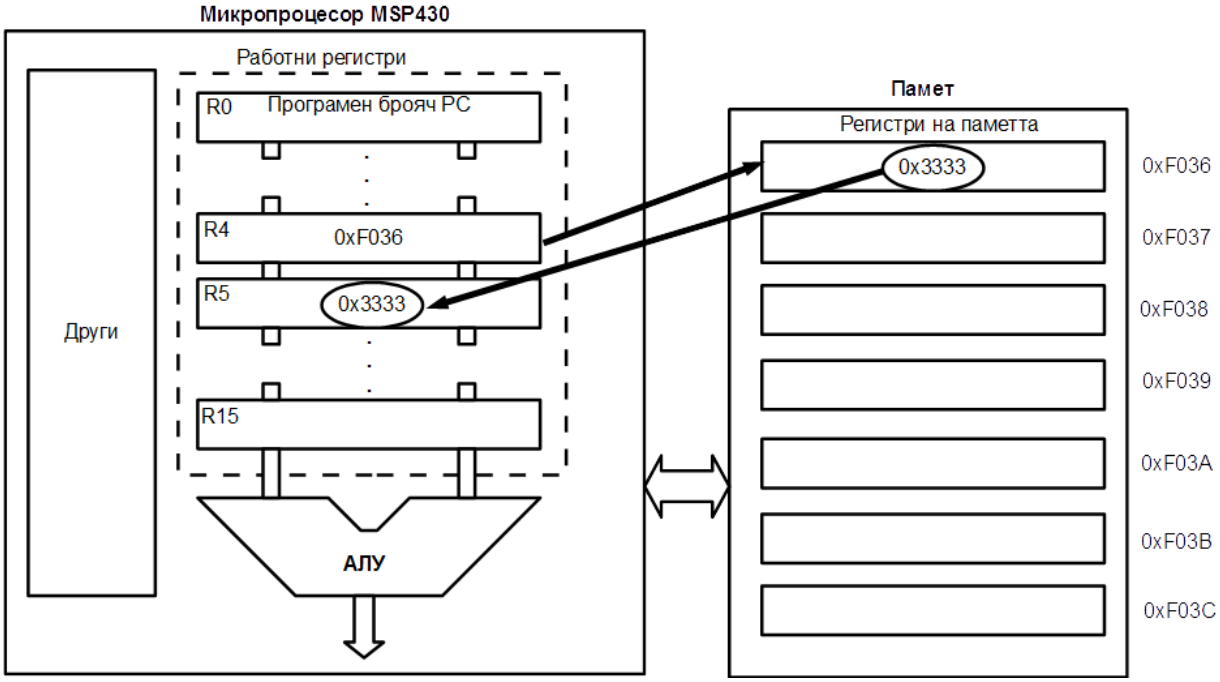
2.4.3. Абсолютна адресация (Absolute) – съдържанието на регистър от програмната памет указва адреса (някъде в паметта на микроконтролера) на операнда.

Пример: mov.w &0xF036, &0xF038 (иди на адрес 0xF036 и копирай каквото има там на адрес 0xF038)
Преди: MEM(0xF036) = 0x3333, MEM(0xF038) = 0x5555
След: MEM(0xF036) = 0x3333, MEM(0xF038) = 0x3333



2.4.5. Индиректна регистрова адресация (Indirect Register) – аналогична на абсолютната. Разликата - съдържанието на работен регистър от ядрото указва абсолютен адрес от паметта, на който се намира операнда.

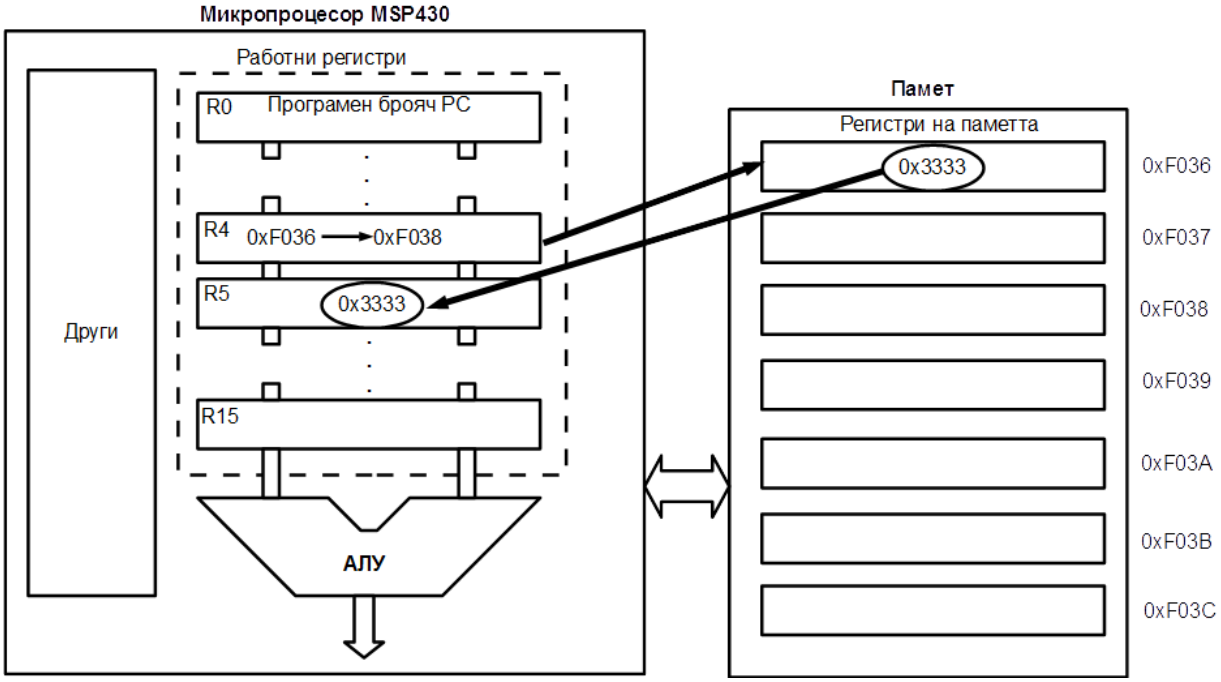
Пример: mov.w @(R4),R5
Преди: MEM(0xF036) = 0x3333, R4 = 0xF036, R5 = 0x5555
След: MEM(0xF036) = 0x3333, R4 = 0xF036, R5 = 0x3333



2.4.6. Индиректна автоинкрементираща адресация (Indirect Autoincrement) – съдържанието на работен регистър от ядрото указва абсолютен адрес от паметта, на който се намира операнда. След изпълнение на инструкцията, съдържанието на работния регистър се увеличава с 1, ако инструкцията е с наставка ".b" или се увеличава с 2, ако инструкцията е с ".w".

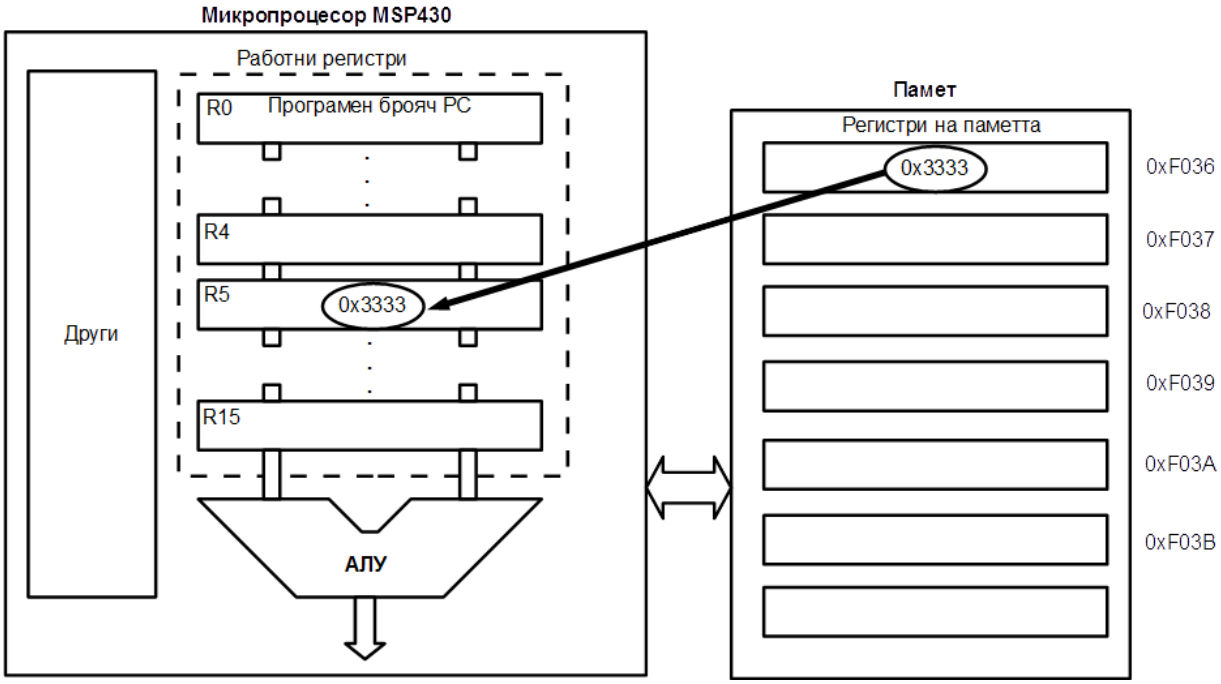
Пример: mov.w @R4+,R5 (R4 се увеличава с 2)
Преди: MEM(0xF036) = 0x3333, R4 = 0xF03**6**, R5 = 0x5555
След: MEM(0xF036) = 0x3333, R4 = 0xF03**8**, R5 = 0x3333

Пример: mov.b @R4+,R5 (R5 се увеличава с 1)
Преди: MEM(0xF036) = 0x3333, R4 = 0xF03**6**, R5 = 0x5555
След: MEM(0xF036) = 0x3333, R4 = 0xF03**7**, R5 = 0x0033 (старшата част се нулира заради „.b" наставката)



2.4.7. Непосредствена адресация (Immediate) – съдържанието на регистър от програмната памет (константа) е операнд на инструкцията.

Пример: mov.w #0x3333,R5
Преди: R5 = 0x5555
След: R5 = 0x3333



2.5. Задачи за изпълнение.

2.5.1. Да се създаде нов проект на Desktop-а в отделна директория и да се въведе програмата на асемблер, която извършва примерни операции, демонстриращи видовете адресации. Използва се само една инструкция – MOV с цел опростяване, НО адресациите са валидни за всички инструкции на MSP430.

2.5.2. Да се постави точка на прекъсване точно след реда:

MOV.W #WDTPW+WDTHOLD,&WDTCTL

Да се изпълнява стъпка по стъпка (трасира), като едновременно с това се попълва таблицата, показана по-долу. Програмата приключва на реда „JMP $“.

1	2	3	4					5		6	7	8	
№	**Инструкция**	**Инструкция в двоичен вид**	**Съдържание на регистрите**					**Вид Адресация** (Reg, IX, Sym, Abs, Ind, Ind+, Imm)		**Брой цикли**	**Брой думи**	**RAM**	
			PC	**R4**	**R5**	**SR**	**SP**	**SRC**	**DST**			**0x1C00**	**0x1C04**
. . .													
			Абсолютен адрес			**Брой цикли общо:**							
			L1		**L2**		**L3**						

Инструкцията в двоичен вид се взима от прозореца Disassembly.

Адресацията се записва за левия (SRC – source) и десния (DST – destination) операнд. Ако има само един операнд се пише само в DST. Ако няма операнди се поставя тире.

Брой цикли и брой думи се взимат след попълване на цялата таблица от приложението на следващата страница. Въпросната информация е взета от документ с описание на микропроцесора MSP430 от фирмата производител Texas Instruments.

Абсолютния адрес на етикетите се взима от прозореца Disassembly. Асемблерът е дал адресите на тези етикети, микропроцесорът работи само с числени стойности. Етикетите се използват за улеснение на програмиста.

Общият брой на циклите се изчислява като се сумират стойностите от всички редове на колонка „Брой цикли“. Това може да помогне на програмиста за оценка времето на изпълнение на програмта му, ако в нея няма условни преходи.

Показване на регистрите на ядрото: View → Register

Показване на регистрите на паметта: View → Memory. Изберете наблюдение на регион 0x1C00 (началото на SRAM).

Показване на дисасемблер View → Disassembly

Table 3-3. Source/Destination Operand Addressing Modes

As/Ad	Addressing Mode	Syntax	Description
00/0	Register mode	Rn	Register contents are operand
01/1	Indexed mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	Symbolic mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	Absolute mode	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/-	Indirect register mode	@Rn	Rn is used as a pointer to the operand.
11/-	Indirect autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/-	Immediate mode	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

В таблиците по-долу TONY и EDE са етикети (условни адреси).
Rn – регистрова; @Rn – индиректна; @Rn+ – индиректна автоинкрементираща; #N – непосредствена, X(Rn) – индексна, EDE – относителна, &EDE – абсолютна.

4.5.1.5.2 Format II (Single-Operand) Instruction Cycles and Lengths

Table 4-9 lists the length and the CPU cycles for all addressing modes of the MSP430 single-operand instructions.

Table 4-9. MSP430 Format II Instruction Cycles and Length

Addressing Mode	No. of Cycles			Length of Instruction	Example
	RRA, RRC SWPB, SXT	PUSH	CALL		
Rn	1	3	4	1	`SWPB R5`
@Rn	3	3	4	1	`RRC @R9`
@Rn+	3	3	4	1	`SWPB @R10+`
#N	N/A	3	4	2	`CALL #LABEL`
X(Rn)	4	4	5	2	`CALL 2(R7)`
EDE	4	4	5	2	`PUSH EDE`
&EDE	4	4	6	2	`SXT &EDE`

Забележка: Колонката „RRA, RRC, SWPB, SXT” трябва да се чете като „Всички инструкции без PUSH и CALL”.

(Формат III)

4.5.1.5.3 Jump Instructions Cycles and Lengths

All jump instructions require one code word and take two CPU cycles to execute, regardless of whether the jump is taken or not.

NOP – 1 дума, 1 такт

4.5.1.5.4 *Format I (Double-Operand) Instruction Cycles and Lengths*

Table 4-10 lists the length and CPU cycles for all addressing modes of the MSP430 Format I instructions.

Table 4-10. MSP430 Format I Instructions Cycles and Length

Addressing Mode		No. of Cycles	Length of Instruction	Example
Source	Destination			
Rn	Rm	1	1	`MOV R5,R8`
	PC	3	1	`BR R9`
	x(Rm)	4(1)	2	`ADD R5,4(R6)`
	EDE	4(1)	2	`XOR R8,EDE`
	&EDE	4(1)	2	`MOV R5,&EDE`
@Rn	Rm	2	1	`AND @R4,R5`
	PC	4	1	`BR @R8`
	x(Rm)	5(1)	2	`XOR @R5,8(R6)`
	EDE	5(1)	2	`MOV @R5,EDE`
	&EDE	5(1)	2	`XOR @R5,&EDE`
@Rn+	Rm	2	1	`ADD @R5+,R6`
	PC	4	1	`BR @R9+`
	x(Rm)	5(1)	2	`XOR @R5,8(R6)`
	EDE	5(1)	2	`MOV @R9+,EDE`
	&EDE	5(1)	2	`MOV @R9+,&EDE`
#N	Rm	2	2	`MOV #20,R9`
	PC	3	2	`BR #2AEh`
	x(Rm)	5(1)	3	`MOV #0300h,0(SP)`
	EDE	5(1)	3	`ADD #33,EDE`
	&EDE	5(1)	3	`ADD #33,&EDE`
x(Rn)	Rm	3	2	`MOV 2(R5),R7`
	PC	5	2	`BR 2(R6)`
	TONI	6(1)	3	`MOV 4(R7),TONI`
	x(Rm)	6(1)	3	`ADD 4(R4),6(R9)`
	&TONI	6(1)	3	`MOV 2(R4),&TONI`
EDE	Rm	3	2	`AND EDE,R6`
	PC	5	2	`BR EDE`
	TONI	6(1)	3	`CMP EDE,TONI`
	x(Rm)	6(1)	3	`MOV EDE,0(SP)`
	&TONI	6(1)	3	`MOV EDE,&TONI`
&EDE	Rm	3	2	`MOV &EDE,R8`
	PC	5	2	`BR &EDE`
	TONI	6(1)	3	`MOV &EDE,TONI`
	x(Rm)	6(1)	3	`MOV &EDE,0(SP)`
	&TONI	6(1)	3	`MOV &EDE,&TONI`

(1) MOV, BIT, and CMP instructions execute in one fewer cycle.

Задача 2.5.1.

```
#include "msp430.h"                     ; #define controlled include file

        NAME    main                    ; module name

        PUBLIC  main                    ; make the main label vissible
                                        ; outside this module
        ORG     0FFFEh
        DC16    init                    ; set reset vector to 'init' label

        RSEG    CSTACK                  ; pre-declaration of segment
        RSEG    CODE                    ; place program in 'CODE' segment

init:   MOV     #SFE(CSTACK), SP        ; set up stack
        MOV.W   #0x00, R4               ; Нулирай работен регистър R5
        MOV.W   #0x00, R5               ; Нулирай работен регистър R6
        MOV.W   #0x0020, R7             ; Зареди числото 32 в R7
        MOV.W   #0x1C00, R8             ; Зареди началния адрес на SRAM в R8
        MOV.W   #0x1C00, R9             ; Зареди началния адрес на SRAM в R9
        MOV.W   #0x0000, &0x1C00        ; Инициализирай първите 2 байта от SRAM

;Нулирай първите 64 байта от SRAM----------------------
zero:   MOV.W   0(R8), 0(R9)
        ADD.W   #2, R9
        DEC.W   R7
        JNZ     zero
;-----------------------------------------------------
        CLR     R2                              ; Нулирай STATUS регистъра

main:   NOP                                     ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL          ; Stop watchdog timer

        MOV.W   #0x5555, R4                     ; Поставете точка на прекъсване на този ред
        MOV.W   R4, R5

        MOV.W   #0x1BFF, R4                     ; Демонстрация на Непосредствена адресация
        MOV.W   #0x1C00, R5
        MOV.W   #0x3412, &0x1C00
        MOV.W   #0x7856, &0x1C04

        MOV.W   1(R4), 4(R5)                     ; Демонстрация на Индексна адресация

        MOV     #0x05, R4
L1:     DEC     R4
        JNZ     L1                              ; Демонстрация на Символна адресация

        NOP

        JMP     L2                              ; Демонстрация на Символна адресация
        NOP                                     ; Обърнете внимание на инструкцията в двоичен вид
        NOP                                     ; и броя на NOP инструкциите
        NOP
        NOP
        NOP

L2:     JMP     L3                              ; Демонстрация на Символна адресация
        NOP                                     ; Обърнете внимание на инструкцията в двоичен вид
        NOP                                     ; и броя на NOP инструкциите
        NOP

L3:     MOV.W   #0xcdab, &0x1C00
        MOV.W   &0x1C00, &0x1C04                ; Демонстрация на Абсолютна адресация

        CLR.W   &0x1C04
        MOV.W   #0x1C00, R4
        MOV.W   @R4, &0x1C04                    ; Демонстрация на Индиректна адресация

        MOV.W   R4, &0x1C04

        MOV.W   @R4+, R5                        ; Демонстрация на Индиректна автоинкрементираща
        MOV.W   @R4+, R5                        ; адресация
        MOV.W   @R4+, R5
        MOV.W   @R4+, R5
```

```
MOV.W     #0x1C00, R4

MOV.B     @R4+, R5      ; Демонстрация на Индиректна автоинкрементираща
MOV.B     @R4+, R5      ; адресация с байтов достъп
MOV.B     @R4+, R5
MOV.B     @R4+, R5

MOV.W     #0x1234, R4
MOV.W     #0xabcd, R5

PUSH.W    R4            ; Демонстрация на PUSH и POP инструкции
PUSH.W    R5            ; Обърнете внимание на реда на извикването им

MOV.W     #0x5555, R4
MOV.W     #0x3333, R5

POP.W     R5
POP.W     R4

JMP       $
END
```

Лабораторно упражнение №3

“Управление на входно-изходните портове на микроконтролера MSP430FR5739”

3.1. Въведение. MSP430FR5739 (RHA40) има 40 извода, от които 32 могат да се използват като логически входове/изходи за общо предназначение (GPIO – General Purpose Input Output). Конфигурацията им се променя в зависимост от фърмуера, който изпълнява микроконтролера. Texas Instruments номерират всеки извод по следната схема:

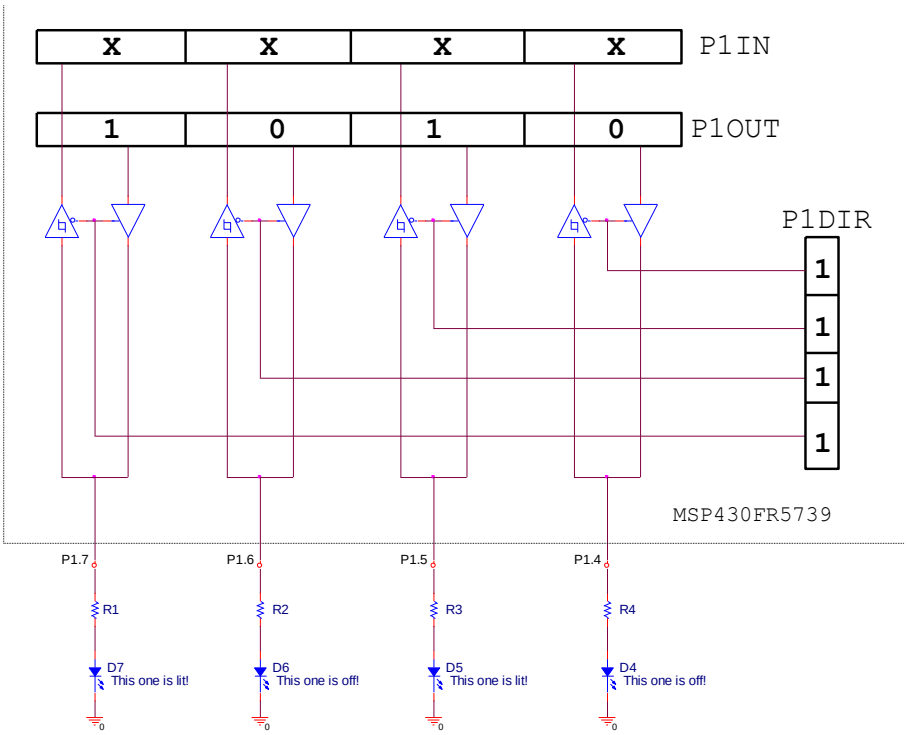
Px.y

където **x** е номера на входно-изходния порт, а **y** – номера на входно-изходния извод.

Портовете се номерират с цифра от 1 ÷ 4, като изключение прави PJ.y, който обозначава специален порт, към който е свързан JTAG интерфейсът. Ако JTAG не се използва, те може да се използват като изводи с общо предназначение. Входовете са с тригер на Шмит. Номерът на извода може да приема стойности P1.(0 ÷ 7), P2.(0 ÷ 7), P3.(0 ÷ 7), P4.(0 ÷ 1) и PJ.(0 ÷ 5).

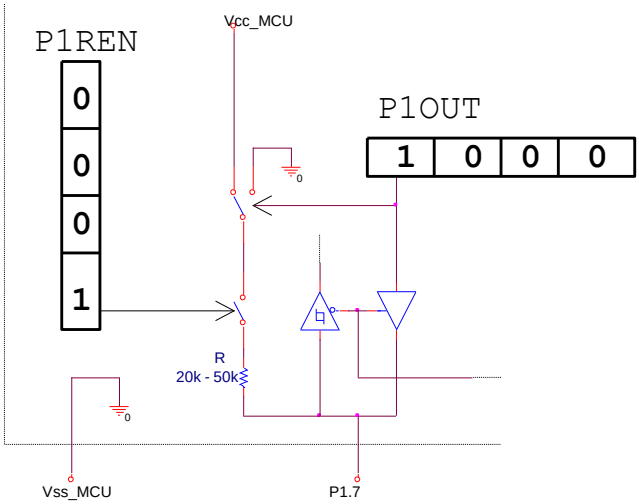
На всеки извод отговаря по един бит от даден регистър в микроконтролера. Най-важните регистри за управлението на портовете са:

- PxDIR – регистър, указващ типа на извода. Избира дали ще е вход или изход.
- PxIN – регистър, отразяващ логическото състояние на изводите, конфигурирани като входове.
- PxOUT – регистър, контролиращ логическото състояние на изводите, конфигурирани като изходи.



Фиг. 3.1

Ако конфигурираме изводите на микроконтролера като изходи и свържем към тях периферни устройства, например светодиоди, то чрез запис в PxOUT регистъра ще можем да управляваме тези устройства посредством логическите нива на изводите. Такъв пример е показан на **фиг.3.1**. За простота е показан само най-старшият квартет от използваните регистри. При запис $10100000_{(b)}$ в P1OUT ще светят само светодиоди D7 и D5. Аналогично при конфигуриране на изводите като входове, ако подадем логическа 1 на P1.7 и P1.5, то бит 7 и 5 от P1IN ще се установят в 1. Тогава програмата може да прочете регистър P1IN, което ще върне резултат $10100000_{(b)}$ и така да разбере на кой извод какво ниво е подадено. Често в микроконтролерите се включват вътрешни резистори, издърпващи даден вход към логическа 0 или 1 (**фиг. 3.2**). Това позволява намаляване броя на външно-включваните елементи. Най-често това се използва при свързване на бутони и схеми с отворен колектор/дрейн.

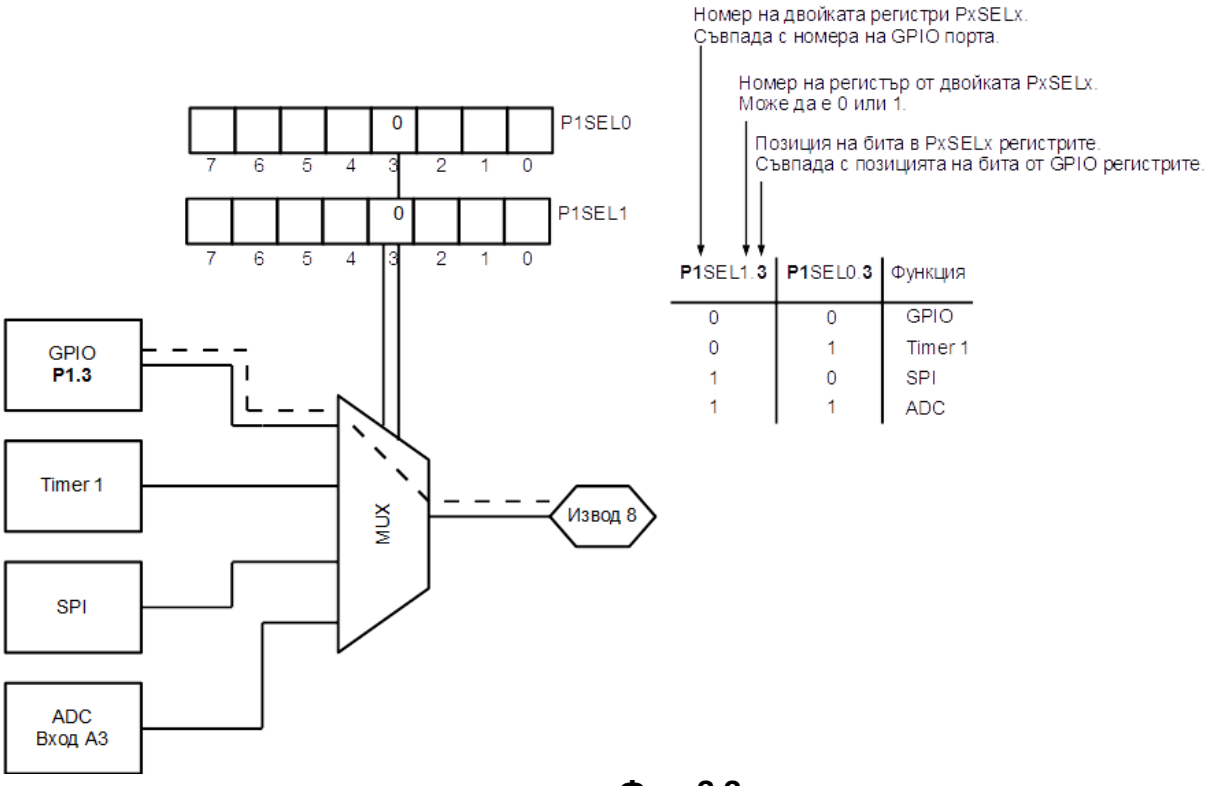


Фиг. 3.2

- PxREN – регистър за включване/изключване на вътрешния издърпващ резистор (1 – включен, 0 – изключен). Когато дадения извод е конфигуриран като вход, PxOUT задава вида на резистора (1 – pull-up, 0 – pull-down). Стойността му е без значение и варира от 20 ÷ 50 k (по каталожни данни).
- PxIE – регистър, разрешаващ прекъсванията (interrupts). Ако даден извод е конфигуриран като вход, той може да се използва за генериране на прекъсване (1 – прекъсването от даден извод е разрешено, 0 – прекъсването е забранено). Прекъсването е процес на спиране изпълнението на главната програма и стартиране изпълнението на специална програма (interrupt handler), разположена на различен адрес в паметта. Този процес е асинхронен спрямо изпълнението на главната програма. Той позволява микроконтролерът свободно да извършва дадени операции и само при настъпване на събитие (прекъсване) да обслужва заявката чрез специален сорс код. Алтернативен метод на работа е с постоянна проверка (polling) за настъпило събитие, но докато тя се извършва, микроконтролерът не може да прави нищо друго.
- P1IFG – регистър с битове, отговарящи на източника на прекъсване. Специалната програма (interrupt handler) в повечето случай е само една за GPIO модула. При работа с два или повече извода, програмата няма как да знае кой от тях е генерирал прекъсването. Затова регистър P1IFG

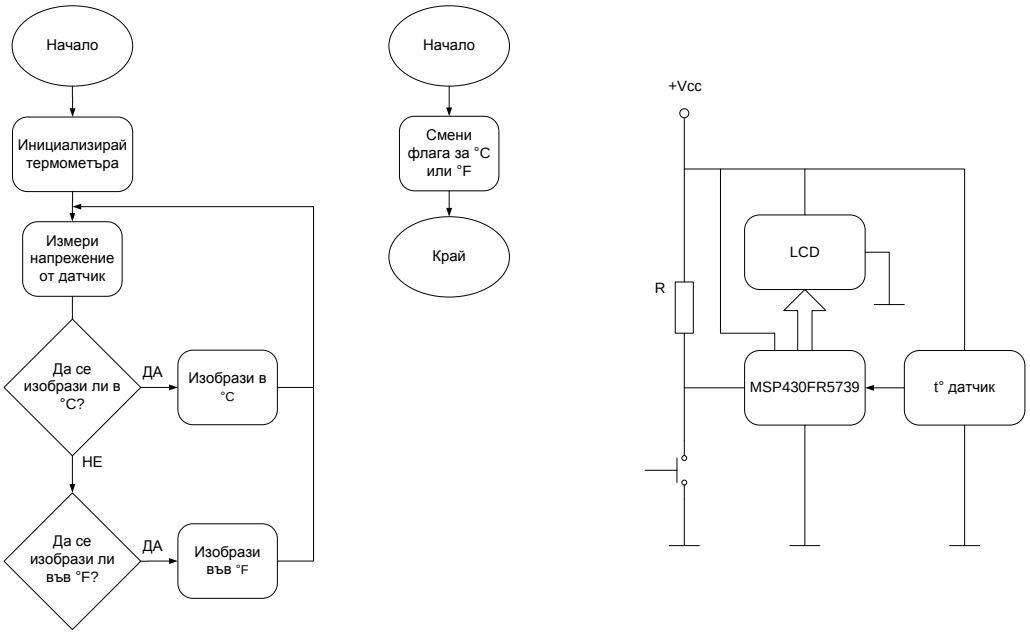
съдържа флагове, осигуряващи ни информацията кой точно извод е генерирал прекъсването. Позицията на флага (бита) носи тази информация.

- PxIES – регистър, определящ активния фронт на прекъсването. Прекъсването от извод, конфигуриран като вход, може да се осъществи по два начина – по нарастващ фронт (0 → 1) или по падащ фронт (1 → 0) на входния сигнал. 0 – конфигурира прекъсване по нарастващ фронт (rising edge), 1 – по падащ (falling edge). Други микроконтролери позволяват прекъсване по четири начина – по фронт(0 → 1 / 1 → 0) и по ниво (1 / 0).
- PxSEL0 и PxSEL1 – регистри, конфигуриращи функцията на извода. При всички микроконтролери даден извод понякога може да извършва повече от една функция. В настоящото упражнение разглеждаме функцията GPIO, но изводите може да се използват и от ADC, компаратор, PWM, SPI и други модули. За да се превключи функцията на извода, производителят на ИС реализира мултиплексор, като всеки негов вход се избира от регистри PxSEL0 и PxSEL1 (за MSP430). Данни за функциите на всеки извод може да се намерят в каталога (datasheet) на микроконтролера. На **фиг. 3.3** е показан извод 8 на MSP430FR5739(RHA40), който може да се използва от GPIO, Timer1, SPI и ADC модула. Както се вижда от таблицата на фигурата, името на регистъра се формира от номера на GPIO порта, свързан към дадения извод. Това значи, че регистри P1SEL0 – P1SEL1 мултиплексират изводи P1.0 – P1.7, P2SEL0 – P2SEL1 мултиплексират P2.0 – P2.7 и т.н. Комбинацията от допълнителни периферни модули, свързани към изводите, зависи от конкретния микроконтролер.



Фиг. 3.3

Пример за програма, използваща прекъсвания е показана на **фиг. 3.4**. Представена е блоковата схема на устройството и управляващия алгоритъм. От тях е видно, че основната програма изпълнява измерването на температурата и опреснява дисплея. Когато потребителят натисне бутона, микропроцесора получава прекъсване и обслужва по-малка програма, която променя начина на изобразяване на температурата. След това изпълнението на главната програма продължава там, откъдето е била прекъсната.



Фиг. 2.4

3.2 Задачи за изпълнение.

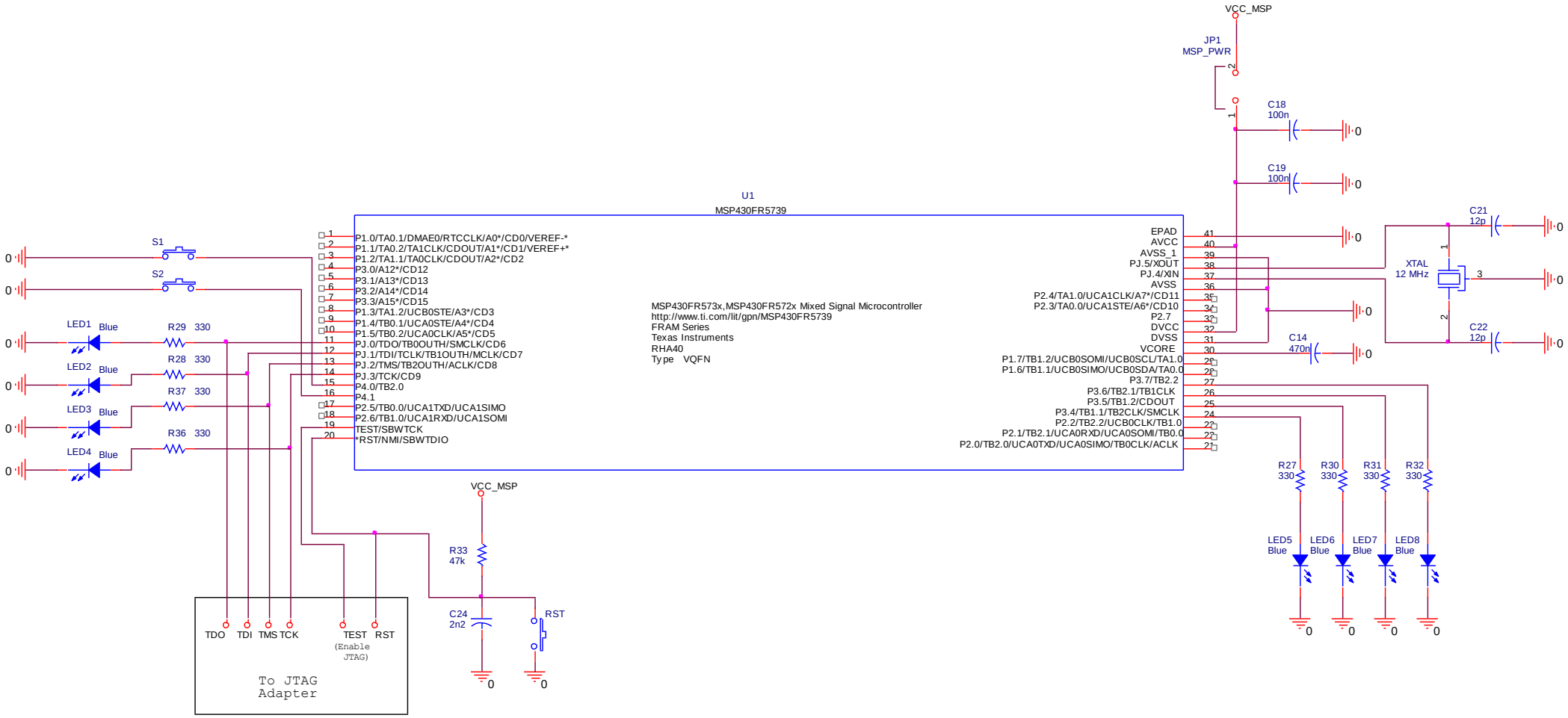
3.2.1. Да се създаде нов проект на Desktop-а в отделна директория с име 03_2_1_Lab, да се копира програмата на асемблер, която включва и изключва светодиод от демо платката MSP-EXP430FR5739. Принципната схема е показана на **фиг. 3.5**. С цел опростяване, част от елементите на макета са пропуснати. Да се трасира програмата стъпка по стъпка. Да се наблюдават регистрите на микропроцесора и изходния порт. Да се постави точка на прекъсване.

3.2.2. Да се създаде нов проект на Desktop-а в отделна директория с име 03_2_2_Lab и да се реализира задача 3.2.1., но с програмния език С.

3.2.3. Да се създаде нов проект на Desktop-а в отделна директория с име 03_2_3_Lab и да се реализира програма на С, която да чете състоянието на бутон S1 и да превключва светодиод LED8 от платката. Да се използва метода “polling”. От View → Register да се наблюдава регистъра P4IN.

3.2.4. Да се създаде нов проект на Desktop-а в отделна директория с име 03_2_4_Lab и да се реализира програма на С, която да чете състоянието на бутон S1 и да превключва светодиод LED8 от платката. Да се използват прекъсвания.

3.2.5. Да се обясни разликата в механизма на действие за задача 3.2.3. и 3.2.4.


U1
MSP430FR5739
MSP430FR573x,MSP430FR572x Mixed Signal Microcontroller
http://www.ti.com/lit/gpn/MSP430FR5739
FRAM Series
Texas Instruments
RHA40
Type VQFN
VCC_MSP
JP1
MSP_PWR
C18
100n
C19
100n
C21
12p
C22
12p
XTAL
12 MHz
C14
470n
S1
S2
LED1 Blue
LED2 Blue
LED3 Blue
LED4 Blue
R29 330
R28 330
R37 330
R36 330
R27 330
R30 330
R31 330
R32 330
LED5 Blue
LED6 Blue
LED7 Blue
LED8 Blue
R33
47k
C24
2n2
RST
TDO
TDI
TMS
TCK
TEST
RST
(Enable JTAG)
To JTAG Adapter
P1.0/TA0.1/DMAE0/RTCCLK/A0*/CD0/VEREF-*
P1.1/TA0.2/TA1CLK/CDOUT/A1*/CD1/VEREF+*
P1.2/TA1.1/TA0CLK/CDOUT/A2*/CD2
P3.0/A12*/CD12
P3.1/A13*/CD13
P3.2/A14*/CD14
P3.3/A15*/CD15
P1.3/TA1.2/UCB0STE/A3*/CD3
P1.4/TB0.1/UCA0STE/A4*/CD4
P1.5/TB0.2/UCA0CLK/A5*/CD5
PJ.0/TDO/TB0OUTH/SMCLK/CD6
PJ.1/TDI/TCLK/TB1OUTH/MCLK/CD7
PJ.2/TMS/TB2OUTH/ACLK/CD8
PJ.3/TCK/CD9
P4.0/TB2.0
P4.1
P2.5/TB0.0/UCA1TXD/UCA1SIMO
P2.6/TB1.0/UCA1RXD/UCA1SOMI
TEST/SBWTCK
*RST/NMI/SBWTDIO
EPAD
AVCC
AVSS_1
PJ.5/XOUT
PJ.4/XIN
AVSS
P2.4/TA1.0/UCA1CLK/A7*/CD11
P2.3/TA0.0/UCA1STE/A6*/CD10
P2.7
DVCC
DVSS
VCORE
P1.7/TB1.2/UCB0SOMI/UCB0SCL/TA1.0
P1.6/TB1.1/UCB0SIMO/UCB0SDA/TA0.0
P3.7/TB2.2
P3.6/TB2.1/TB1CLK
P3.5/TB1.2/CDOUT
P3.4/TB1.1/TB2CLK/SMCLK
P2.2/TB2.2/UCB0CLK/TB1.0
P2.1/TB2.1/UCA0RXD/UCA0SOMI/TB0.0
P2.0/TB2.0/UCA0TXD/UCA0SIMO/TB0CLK/ACLK

Фиг. 3.5

Задача 3.2.1.

```
#include "msp430.h"                                    ; #define controlled include file

            NAME    main                               ; module name

            PUBLIC  main                               ; make the main label vissible
                                                       ; outside this module
            ORG     0FFFEh
            DC16    init                               ; set reset vector to 'init' label

            RSEG    CSTACK                             ; pre-declaration of segment
            RSEG    CODE                               ; place program in 'CODE' segment

init:       MOV     #SFE(CSTACK), SP                   ; set up stack

main:       NOP                                        ; main program
            MOV.W   #WDTPW+WDTHOLD,&WDTCTL             ; Stop watchdog timer
;-------------------------------------------------------------------------------------------------------------------------------
            BIS.B   #???, &P3DIR                       ;Инициализация на порт P3, извод 4 – изход
                                                       ;MSP430FR5739 Userguide → стр. 287
            MOV.B   #???, &P3OUT                       ;Установи всички изводи в логическа 0
                                                       ;MSP430FR5739 Userguide → стр. 287

L1          XOR.B   #???, &P3OUT                       ;Смяна състоянието на извод P3.4
                                                       ;MSP430FR5739 Userguide → стр. 287
            CALL    #Delay                             ;Извикай подпрограма за изчакване
            JMP     L1                                 ;Върни се при етикет L1


Delay:      MOV.W   #65535, R15                        ;Зареди регистър R15 с числото 65535
L2          DEC.W   R15                                ;Намали регистър R15 с 1
            JNZ     L2                                 ;Ако R15 == 0, продължи. Ако R15 != 0,
                                                       ;върни се при етикет L2
            RET                                        ;Върни се в main
;-------------------------------------------------------------------------------------------------------------------------------
            END
```

Задача 3.2.2.

```
#include "io430.h"

void main( void )
{
  unsigned long i;

  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  ??? //Конфигурирай извод P3.5 като изход, MSP430FR5739 Userguide → стр. 287
  ??? //Инициализирай нивото на P3.5 в логическа нула, MSP430FR5739 Userguide → стр. 287

  for( ; ; )
  {
  ???//Реализирай преобръщане на логическото ниво (toggle) чрез логически оператор от C

  ??? //Реализирай софтуерно закъснение чрез празен for( ; ; ){ } цикъл
  }
}
```

Задача 3.2.3.

```
#include "io430.h"

void main( void )
{
  unsigned long i;
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  PJDIR = 0x03; //Конфигурирай изводи PJ.0 и PJ.1 като изходи
  P3DIR = 0x80; //Конфигурирай извод P3.7 като изход
  P3OUT = 0x00; //Инициализирай извод P3.7 с лог. 0

??? //Конфигурирай извод P4.0 като вход, MSP430FR5739 Userguide → стр. 287
???//Включи издърпващ резистор на извод P4.0, MSP430FR5739 Userguide → стр. 287
???//Свържи издърпващия резистор към захранване Vdd, ;MSP430FR5739 Userguide
     //→ стр. 287

  while(1)
  {
    PJOUT = 0x01;
    for(i = 0; i < 40000; i++){}
    PJOUT = 0x02;
    for(i = 0; i < 40000; i++){}

    if((P4IN & 0x01) == 0)
    {
      P3OUT ^= 0x80;
    }
  }
}
```

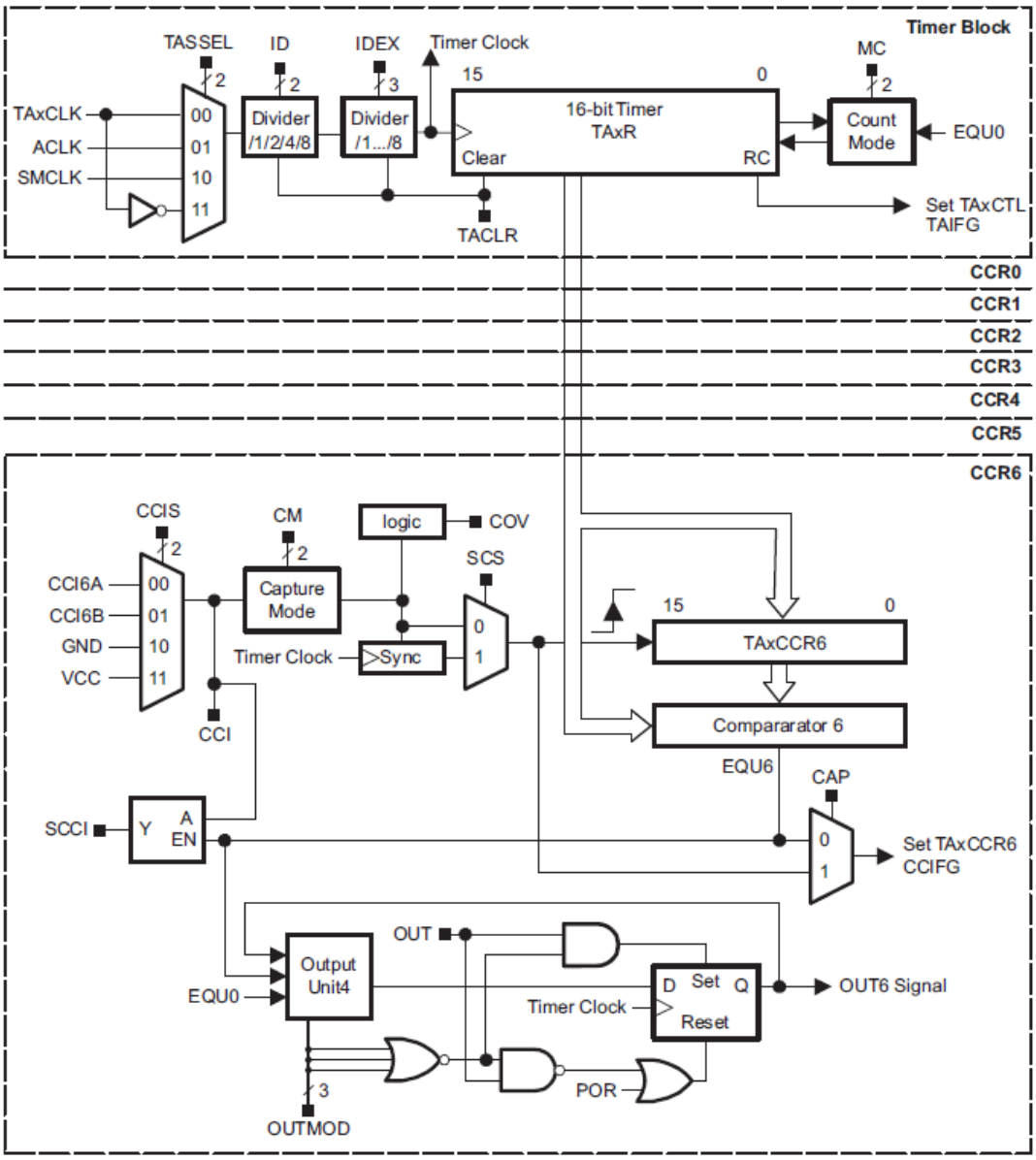
Задача 3.2.4.

```
#include "io430.h"

void main( void )
{
  unsigned long i;
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  PJDIR = 0x03; //Конфигурирай изводи PJ.0 и PJ.1 като изходи.
  P3DIR = 0x80; //Конфигурирай извод P3.7 като изход.
  P3OUT = 0x00; //Инициализирай извод P3.7 с лог. 0.

  ??? //Конфигурирай извод P4.0 като вход, MSP430FR5739 Userguide → стр. 287
  ??? //Включи издърпващ резистор на извод P4.0, MSP430FR5739 Userguide → стр. 287
  ??? //Свържи издърпващия резистор към захранване Vdd,MSP430FR5739 Userguide → стр. 287

  ??? //Избери прекъсване по падащ фронт, MSP430FR5739 Userguide → стр. 286
  ??? //Нулирай всички флагове на прекъсване за порта, MSP430FR5739 Userguide → стр. 286
  ??? //Разреши прекъсването от извод P4.0, MSP430FR5739 Userguide → стр. 286

  __enable_interrupt( ); //Разреши глобално прекъсванията към микропроцесора MSP430.

  while(1)
  {
    PJOUT = 0x01;
    for(i = 0; i < 40000; i++){}
    PJOUT = 0x02;
    for(i = 0; i < 40000; i++){}
  }
}

#pragma vector=PORT4_VECTOR //Хендлер на прекъсване за GPIO модул 4.
__interrupt void port4_handler(void)
{
  unsigned int interrupt_flag;
  unsigned long i;

  interrupt_flag = P4IFG; //Прочети регистъра с флаговете.

  switch(interrupt_flag)
  {
  case 0x01: //Прекъсване от извод P4.0.
    ??? //Преобърни извод P3.7, чрез логически оператор от C
    break;
  case 0x02: //Прекъсване от извод P4.1.
    break;
  default: //Неразпознат източник на прекъсване.
    break;
  }

  for(i = 0; i < 30000; i++){ } //Закъснение срещу притрепване на контакта.
  P4IFG = 0x00; //Нулирай флаговете на прекъсване.
}
```

Лабораторно упражнение №4
“Изследване на таймерен модул на микроконтролер MSP430FR5739”

4.1. Въведение. В практиката се налага микроконтролерните системи да генерират или да отмерват точно определени времеинтервали. Ако програмиста знае тактовата честота на контролера и времето за изпълнение на асемблерните инструкции, то той би могъл да реализира софтуерно времеинтервал. Това обаче почти никога не се използва в практиката, защото тактовата честота може да се изменя динамично, изпълнението на програмата не е последователно (обслужват се прекъсвания, които биха удължили софтуерното закъснение), съществуват инструкции с условие, които променят хода на програмата. Затова във всички микроконтролери се вграждат хардуерни модули, които са конфигурират от ядрото, но броячите им се увеличават/намаляват независимо от него. При препълване броячът се нулира и сигнализира за това на ядрото чрез прекъсване. Когато увеличаването/намаляването на брояча става в интервал от време, определен от извод на микроконтролера се казва, че таймерът работи в режим Capture. Когато текущата стойност на брояча се сравнява с друг регистър и при съвпадение на двете стойности се генерира прекъсване се казва, че таймерът работи в режим Compare. Когато стойността на брояча се сравнява с други два регистъра и автоматично се запуска отначало след достигане на съвпадение се казва, че таймера работи в режим на генерация на ШИМ (Широчинно Импулсна Модулация) сигнал. В последния случай единия регистър определя период, а другия – коефициента на запълване. Таймерите понякога се наричат Capture/Compare/PWM или CCP модули.

4.2. Таймерен модул на MSP430FR5739. На **фиг. 4.1** е показана блокова схема на таймерния модул. От нея се вижда, че 16-битовия брояч за един таймерен модул е само един, докато CCP регистрите може да достигнат до 7 броя. За конкретния модел (5739) те са 3. Всеки един CCP регистър е поместен заедно с допълнителна логика в таймерен подмодул. От друга страна микроконтролера има два таймерни модули (Timer_A и Timer_B) с общо 2 x 3 = 6 CCP регистъра. Тактовата честота, която се подава на брояча идва от един от източниците TAxCLK, ACLK, SMCLK. Следват два делителя на честота, които позволяват гъвкава настройка на времеинтервалите. Бит CAP от съответния TA(x)CCTL(n) регистър определя работата на всеки един подмодул в режим на Capture или Compare.


Timer Block
TASSEL
ID
IDEX
Timer Clock
MC
TAxCLK
ACLK
SMCLK
00
01
10
11
Divider /1/2/4/8
Divider /1.../8
15
0
16-bit Timer TAxR
Clear
RC
Count Mode
EQU0
Set TAxCTL TAIFG
TACLR
CCR0
CCR1
CCR2
CCR3
CCR4
CCR5
CCR6
CCIS
CM
logic
COV
SCS
CCI6A
CCI6B
GND
VCC
Capture Mode
Timer Clock
Sync
CCI
TAxCCR6
Compararator 6
EQU6
CAP
SCCI
Y
A
EN
Set TAxCCR6 CCIFG
OUT
Output Unit4
EQU0
D
Set
Q
Reset
Timer Clock
OUT6 Signal
POR
OUTMOD

Фиг. 4.1

4.3. Задачи за изпълнение.

4.3.1. Да се създаде нов проект на Desktop-а в отделна директория и да се въведе програмата на C, която мига светодиод LED1 от демо платката с помощта на таймерния модул. Задайте 100 милисекунди времеинтервал. Тактовата честота на ядрото и всички тактови сигнали на микроконтролера да се конфигурират на 8 MHz.

4.3.2. Да се създаде нов проект на Desktop-а в отделна директория и да се въведе програмата на C, която генерира PWM сигнал на извод P1.4. Да се свърже осцилоскоп на този извод.

4.3.3. Да се зададе коефициент на запълване 25 %, 50 % и 75 %. Да се снемат осцилограмите.

4.3.4. Да се промени честотата на PWM сигнала.

4.3.5. Да се промени полярността на PWM сигнала.

Задача 4.3.1.

```
#include "io430.h"

// Timer A0 interrupt service routine
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  TA0CTL &= ~TAIFG;
  PJOUT ^= BIT0;
}

void main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  PJDIR |= 0x01;

  // Тактов генератор DCO = 8MHz
  CSCTL0_H = 0xA5;
  CSCTL1 &= ~DCORSEL;                    //DCORSEL = 0
  CSCTL1 |= DCOFSEL0 | DCOFSEL1;         //DCOFSEL = 11(b)
  CSCTL2 = SELA_3 | SELS_3 | SELM_3;     //ACLK = DCO; SMCLK = DCO; MCLK = DCO
  CSCTL3 = DIVA_0 | DIVS_0 | DIVM_0;     // Всички делители на тактовите сигнали = 1

  TA0CCTL0 |= ????; // Прекъсване при съвпадение с CCR регистъра, Userguide -> стр. 318
  TA0CCR0 = ????;    // Compare стойност за сравнение с рег. на таймера, Userguide -> стр. 319
  TA0CTL = TASSEL_2 | ID_3 | ????; //SMCLK,делител=8, режим UP/DOWN, Userguide -> стр. 317

  __enable_interrupt( );

  while(1){ }
}
```

Задача 4.3.2.

```
#include "io430.h"

void main( void )
{
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer to prevent time out reset

  // Тактов генератор DCO = 8MHz
  CSCTL0_H = 0xA5;
  CSCTL1 &= ~DCORSEL;                    //DCORSEL = 0
  CSCTL1 |= DCOFSEL0 | DCOFSEL1;   // DCOFSEL = 11(b)
  CSCTL2 = SELA_3 | SELS_3 | SELM_3;// Сис. такт ACLK = DCO; SMCLK = DCO; MCLK = DCO
  CSCTL3 = DIVA_0 | DIVS_0 | DIVM_0;  // Всички делители на тактовите сигнали = 1

  P1DIR |= BIT4;                     // P1.4 изход
  P1SEL1 &= ~BIT4;                  // P1.4 изход на
  P1SEL0 |= BIT4;                    // таймерния модул TB0

  TB0CCR0 = ????;    // PWM период, Userguide -> стр. 340
  TB0CCTL1 = ????;  // Полярност на изходния PWM сигнал, Userguide -> стр. 339
  TB0CCR1 = ????;    // PWM коефициент на запълване, MSP430FR5739 Userguide -> стр. 340

  TB0CTL = TBSSEL_2 | MC_1 | TBCLR;     // SMCLK, сумиращ брояч (up mode), изчисти TAR

  while(1){}
}
```

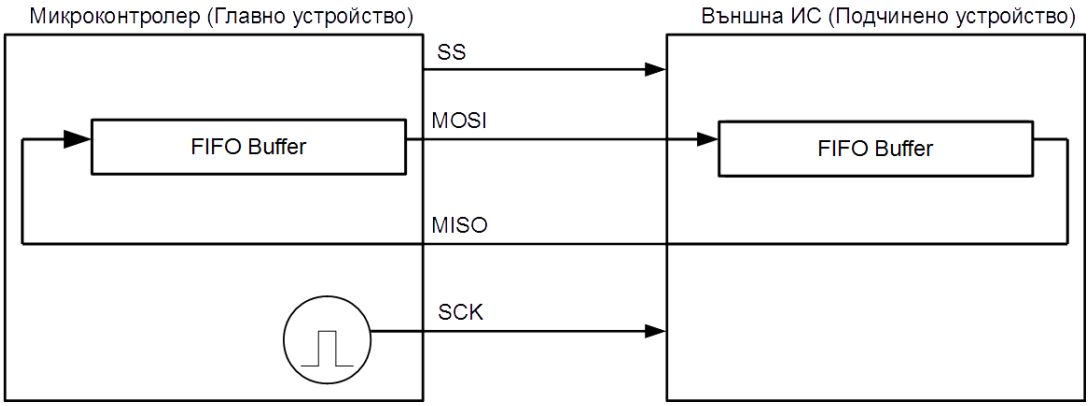
Лабораторно упражнение №5

“Изследване на сериен интерфейс SPI на микроконтролера MSP430FR5739”

5.1. Въведение. Често използван интерфейс за комуникация между две и повече ИС в рамките на една вградена система е SPI интерфейсът, създаден от фирмата Motorola. Това е сериен,синхронен интерфейс,използващ до 4 проводника за обмен на данни. Те са:

- MOSI (Master output / slave input) – изход на главното устройство/вход на подчиненото.
- MISO (Master input / slave output) – вход на главното устройство/изход на подчиненото.
- SCK (Slave Clock) – синхронизиращ тактов сигнал, изработван от главното и подаван към подчиненото устройство. Този сигнал определя интерфейса като „синхронен“.
- SS (Slave Select) – сигнал за избор на подчинено устройство.

При SPI главното (или още–master) устройство задава синхронизиращия тактов сигнал, спрямо който се предават данните. Със всеки тактов импулс се предава един бит информация в даден момент от времето. Това определя интерфейса като „сериен“. Подчиненото устройство (или още–slave) приема правилно данните благодарение на този сигнал. Генератор на такт има само в главното устройство. На **фиг.5.1** е дадено едно типично свързване на две ИС по SPI интерфейс. От нея се вижда, че предаването на данни става посредством преместващи регистри свързани в една обща верига (chain). Зареждането на данните в преместващия регистър се извършва от микропроцесора и е паралелно.



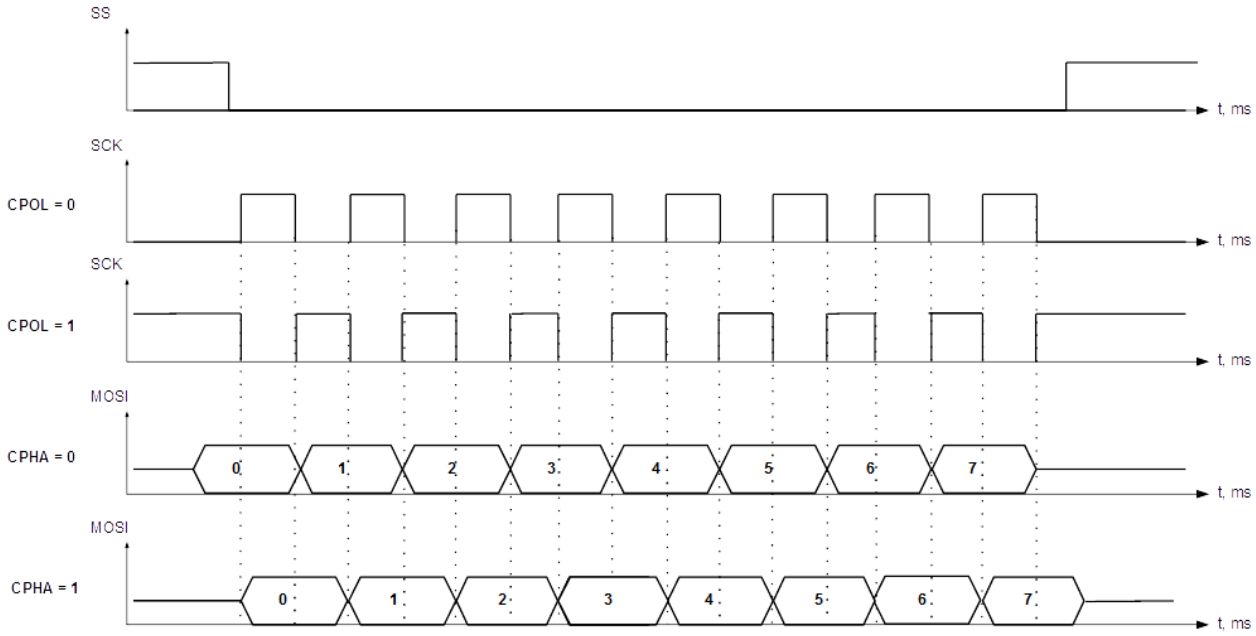
Фиг. 5.1

Протоколът е сравнително прост – реализира 4 режима (SPI modes) на обмен на данни. В таблица 1 са дадени тези режими в зависимост от полярността (CPOL) и фронта на тактовия сигнал (или още – фазата на данните спрямо тактовия сигнал, CPHA).

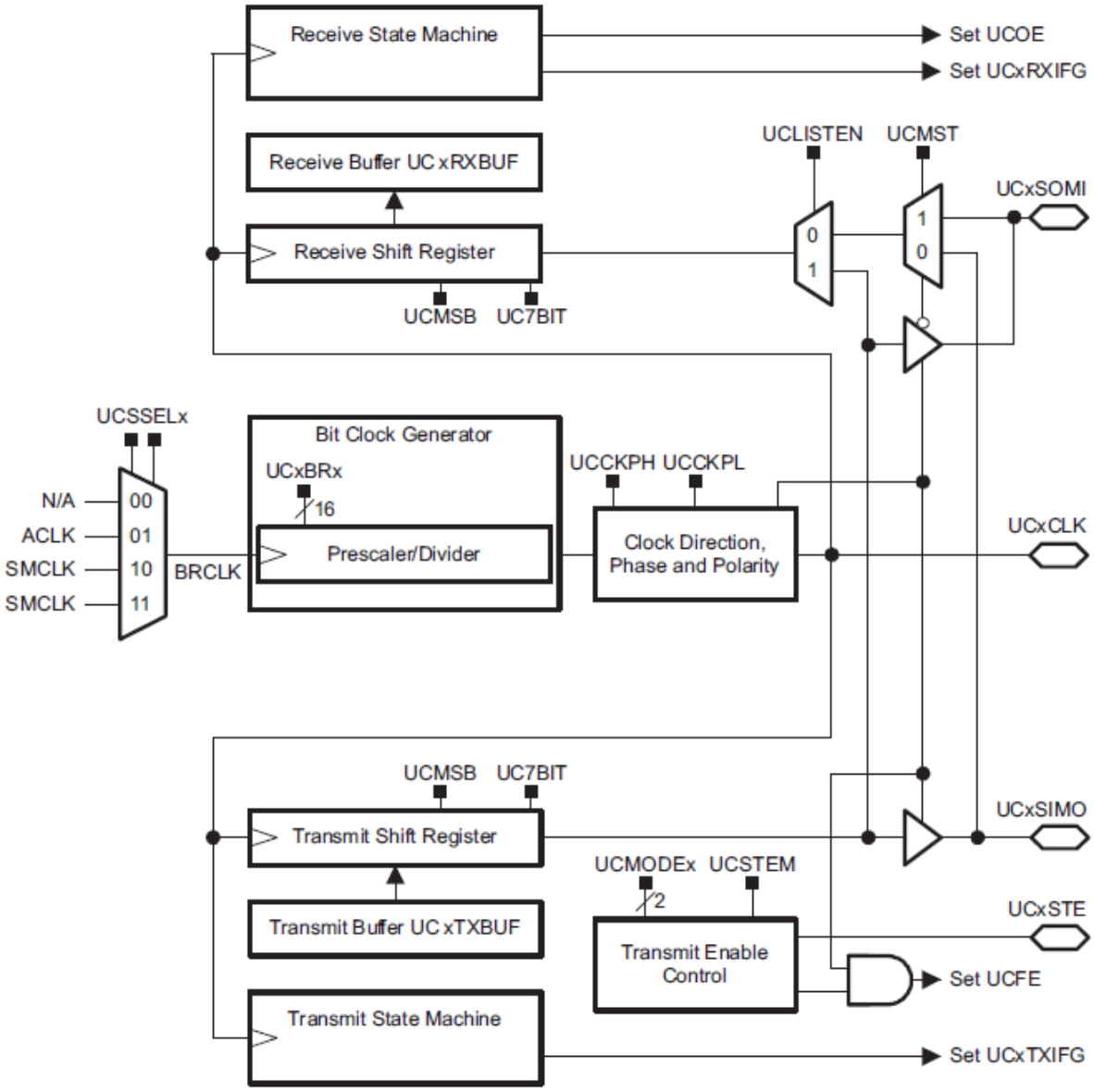
CPOL	CPHA	Режим
0	0	0
0	1	1
1	0	2
1	1	3

Таблица 1

CPOL – определя логическото състояние на проводника, осигуряващ тактов сигнал (SCK),когато по интерфейса няма обмен на данни.
CPHA – определя фронта на тактовия сигнал,по който ще се предават данните.
Оригинално тези означения идват от имената на битовете на конфигуриращи регистри за SPI модул на микроконтролера MC68HC11 на Motorola.



5.2. SPI модул на MSP430FR5739. На **фиг. 5.2** е дадена блокова схема на SPI модула**.** От нея е видно, че данните се записват от микропроцесора в паралелния регистър UCxTXBUF, след което се прехвърлят в преместващия регистър „Transmit Shift Register”. Тогава данните се изместват бит по бит, синхронно спрямо сигнала UCxCLK, на UCxSIMO (или още MOSI). Тактовият сигнал на модула и на интерфейса може да се вземе от системните тактови сигнали ACLK или SMCLK. Той се подава на делител на честота. Приемането става на извод UCxSOMI (или още MISO), на който е свързан входа на приемащия преместващ регистър „Receive Shift Register”. Той прехвърля приетите данни в паралелния регистър UCxRXBUF, който е достъпен за четене от микропроцесора. Интересен факт за MSP430FR5739 е, че SPI, I2C и UART всъщност са един модул (обобщен с названието eUSCI – enhanced Universal Serial Communication Interface, от англ. ез. усложнен универсален сериен комуникационен интерфейс), който може да се конфигурира според нуждите на един от трите интерфейса.


Receive State Machine
Set UCOE
Set UCxRXIFG
Receive Buffer UCxRXBUF
Receive Shift Register
UCMSB
UC7BIT
UCLISTEN
UCMST
UCxSOMI
0
1
1
0
UCSSELx
N/A
ACLK
SMCLK
SMCLK
00
01
10
11
BRCLK
Bit Clock Generator
UCxBRx
16
Prescaler/Divider
UCCKPH UCCKPL
Clock Direction,
Phase and Polarity
UCxCLK
UCMSB
UC7BIT
Transmit Shift Register
UCxSIMO
UCMODEx
UCSTEM
2
Transmit Buffer UCxTXBUF
Transmit Enable
Control
UCxSTE
Set UCFE
Transmit State Machine
Set UCxTXIFG

Фиг. 5.2

5.3. Задачи за изпълнение.

5.3.1. Да се създаде нов проект на Desktop-а в отделна директория и да се въведе програмата на C, която изпраща числото 0x75 по SPI интерефейса в режим CPOL = 0 и CPHA = 0. Да се снемат осцилограмите на изводи MOSI (P2.0), SCK (P1.5) и SS (P1.4).

Забележка: SPI модулът на MSP430FR5739 притежава SS извод, който обаче е наименуван STE. Неговото активно ниво може да се променя с битово поле UCMODEx от UCAxCTLW0.

5.3.2. Да се създаде нов проект на Desktop-а в отделна директория и да се въведе програмата на C, която изпраща числото 0x75 по SPI интерефейса в режим CPOL = 0 и CPHA = 1. Да се снемат осцилограмите на изводи MOSI (P2.0), SCK (P1.5) и SS (P1.4).

5.3.3. Да се създаде нов проект на Desktop-а в отделна директория и да се въведе програмата на C, която изпраща числото 0x75 по SPI интерефейса в режим CPOL = 1 и CPHA = 0. Да се снемат осцилограмите на изводи MOSI (P2.0), SCK (P1.5) и SS (P1.4).

5.3.4. Да се създаде нов проект на Desktop-а в отделна директория и да се въведе програмата на C, която изпраща числото 0x75 по SPI интерефейса в режим CPOL = 1 и CPHA = 1. Да се снемат осцилограмите на изводи MOSI (P2.0), SCK (P1.5) и SS (P1.4).

Задача 5.3.1. / 5.3.2. / 5.3.3. / 5.3.4.

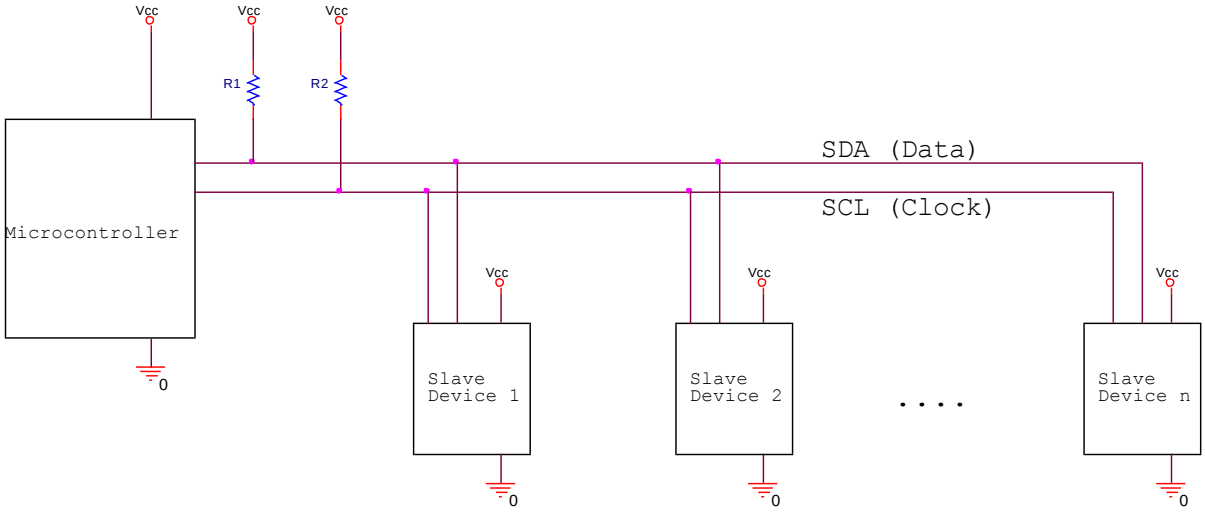
```
#include "io430.h"

void main( void )
{
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer to prevent time out reset

  //Конфигурирай мултиплексора за работа с извод P1.4 като UCA0STE (SS)
  P1SEL1 |= BIT4;
  P1SEL0 &= ~BIT4;

  //Конфигурирай мултиплексора за работа с извод P1.5 като UCA0CLK (SCK)
  P1SEL1 |= BIT5;
  P1SEL0 &= ~BIT5;

  //Конфигурирай мултиплексора за работа с извод P2.0 като UCA0SIMO (MOSI)
  P2SEL1 |= BIT0;
  P2SEL0 &= ~BIT0;

  UCA0CTLW0 |= UCSWRST;              //Задръж SPI модула в ресет
  // 4-изводен SPI (MISO не се използва в л. упр.), 8-bit трансфери, SPI master
  // Полярност на такт - лог.1 , формат на данните MSB първи, такт на SPI модул
  // от ACLK
  UCA0CTLW0 |= UCMST | UCSYNC | UCMSB | UCMODE_2 | UCSTEM | UCSSEL_1;

  UCA0BR0 = 0x02;                    // Раздели честотата на 2
  UCA0BR1 = 0;                       // Раздели честотата на 1

  UCA0CTLW0 ????= ????;              //Задай полярност на SCK
                                     //стр. 456 от MSP430FR5739 user guide
  UCA0CTLW0 ????= ????;              //Задай фаза на MOSI
                                     //стр. 456 от MSP430FR5739 user guide

  UCA0CTLW0 &= ~UCSWRST;             //Пусни SPI модула от ресет

  while(1){
    ???? = ????;                     //Запиши числото, което трябва да се изпрати (0x75)
                                     //в съответния изходен (TX) регистър
                                     //стр. 458 от MSP430ФР5739 user guide
    __delay_cycles(6000);
  }
}
```

Лабораторно упражнение №6
“Изследване на сериен интерфейс I²C на микроконтролера MSP430FR5739”

6.1 Въведение. I²C е синхронен сериен интерфейс, който използва два (+ 1 маса) проводника за предаване на информация между два и повече чипа. Името му идва от IIC – Inter-Integrated Circuit и е създаден от Philips през 1982. От 2006 година използването на протокола за този интерфейс е безплатно. Платено е единствено запазването на уникален адрес за чип от даден вид, използващ I²C, но това засяга само фирмите производителки на чипа.

Връзката между два чипа се реализира на принципа master-slave, т.е. има една главна интегрална схема (master) и една подчинена (slave) такава. Обикновено master чипа е микроконтролерът в системата, а slave е периферен чип, с който ще се осъществява връзката. Съществува и режим multi-master, при който към I²C интерфейса са свързани два или повече главни (master) чипа.

На **фиг. 6.1** е показана блокова схема на система, използваща I²C интер-



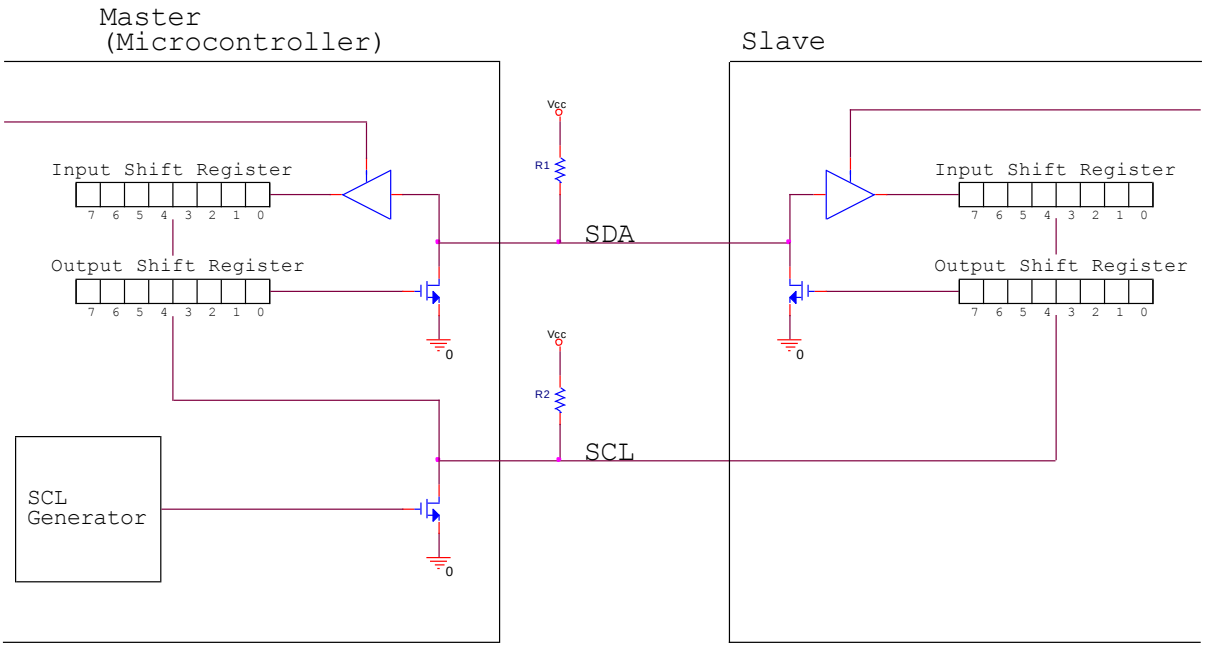
Фиг. 6.1

фейса. Предаването на информацията се осъществява посредством проводниците:

- SDA – данните, предавани серийно по този проводник.
- SCL – тактовият сигнал, по който е синхронизирано изпращането бит по бит на данните.

Към двата проводника са свързани изтеглящи (pull-up) резистори към захранването на системата Vcc. Те са необходими, защото предаването на информация е двупосочно – от микроконтролера към подчинената схема и от подчинената схема към микроконтролера. Това означава, че микроконтролерът ще използва изводът си, свързан към SDA, веднъж като вход и веднъж като изход. Аналогично – подчиненото устройство ще използва SDA като вход, когато иска да приеме информация и след това като изход, когато иска да предаде информация (**фиг. 6.2**). Тази конфигурация води до една теоретична конфликтна ситуация, в която ако и двата чипа са установили изводите си SDA като изходи и единият е в логическа 1, а другият – в логическа 0, то ще се получи късо съединение между захранващия извод Vcc и масата GND. За да се избегне тази възможност се използват изходи с отворен дрейн (или колектор),

които изискват режимен pull-up резистор. SCL също включва такъв резистор, защото протокола I²C дефинира ситуация, в която някой от чиповете задържа тактовия сигнал в логическа 0.



Фиг. 6.2

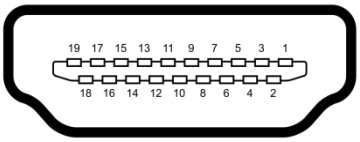
Аналогични на I²C са интерфейсите:

- SMBus – създаден от Intel през 1995. Дефинира по-строго изискванията за ниво и времеви параметри (timings) на сигнала.
- I²S – използва се за предаване на цифров звуков сигнал. Съдържа един допълнителен проводник (Word Select), указващ изпращаният байт на кой канал принадлежи (при стерео предаване – ляв или десен).

I²C интерфейса се характеризира със следните скорости на предаване:

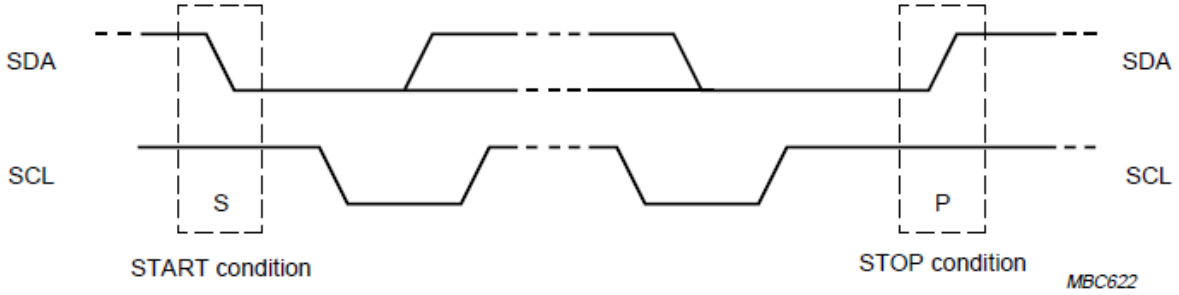
- Оригинална версия от 1982 – 100 kbit/s.
- Версия 1 (1992) – 400 kbit/s (Fast mode).
- Версия 2 (1998) – 3.4 Mbit/s (High-speed mode).
- Версия 3 (2007) – 1 Mbit/s (Fast mode plus).
- Версия 4 (2012) – 5 Mbit/s (Ultra Fast mode).

I²C интерфейса се включва в интегрални схеми с различни функции във вградените системи. АЦП, ЦАП, EEPROM, цифрови потенциометри, температурни датчици, хол датчици и т.н. са само малка част от приложението му. Въпреки че този интерфейс се използва главно за комуникация между ИС в рамките на една вградена система, то не липсват примери за приложения и в комуникацията между две отделни устройства. Например всеки персонален компютър използва I²C като част от HDMI интерфейса **фиг. 6.3**. С негова помощ се „опознават" поддържаните видео формати от изобразяващото устройство (монитор, телевизор, прожектор и други).



Фиг. 6.3 - HDMI куплунг. I²C на извод 15 – SCL и извод 16 – SDA.

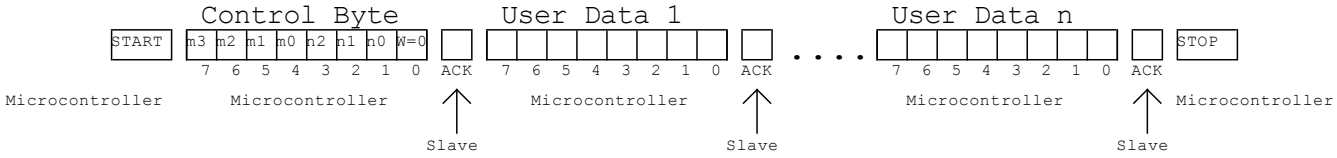
I²C комуникацията се осъществява по стандартизиран протокол. Най-общо казано обменът на данни започва с условие СТАРТ, продължава с трансфер на данни и завършва с условие СТОП. Условията старт и стоп са специални събития, реализирани чрез комбинации на логически нива и фронтове на SCL и SDA линиите, които са уникални и се различават от данните. На **фиг. 6.4** са показани едно старт и стоп условие. Старт условие се генерира, когато SDA



Фиг. 6.4

линията премине в ниско ниво, докато SCL линията е във високо. Стоп условие се генерира, когато SDA линията премине във високо ниво, докато SCL линията е във високо. Между старт и стоп условията се изпращат потребителските данни. Всеки бит от тях трябва да запази своето състояние, докато SCL е във високо ниво (в противен случай ще се генерира старт или стоп условие и трансферът ще се наруши), т.е. трансферът на данни се осъществява по ниво.

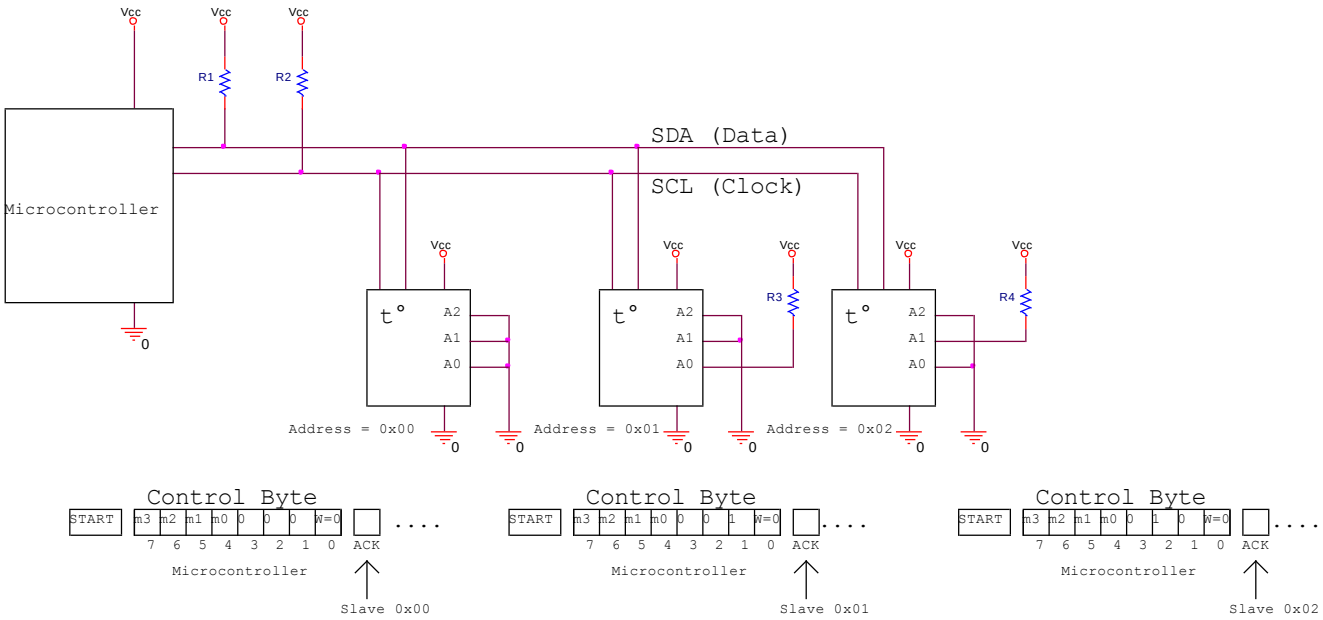
Форматът на данните при **запис** от микроконтролер в подчинена ИС е показан на **фиг. 6.5**.



Фиг. 6.5 – Запис в подчинена ИС.

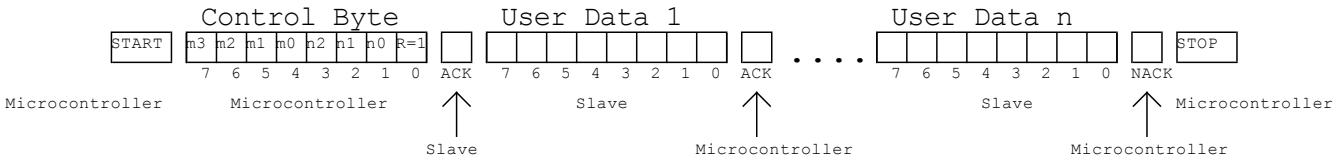
Микроконтролерът генерира СТАРТ условие, след което изпраща контролен байт. Този байт достига до всички подчинени устройства. Старшата тетрада (битове m0 ÷ m3) съдържа контролен код, част от 7-битов адрес, който е различен за различните подчинени чипове – трябва да се провери datasheet-а им. Следват три бита, които указват младшата част на 7-битовия адрес на подчиненото устройство (битове n0 ÷ n2). Благодарение на тях към една I²C шина могат да се свържат повече от един чипове от един и същи вид (на **фиг. 6.6** е показано свързване на три еднакви температурни датчика, чиито адреси са зададени хардуерно с изводи 1, 2 и 3). Последният бит от контролния байт е

четене/запис (R/W) и при запис трябва да е 0. Ако на I²C шината има устройство със зададения адрес (битове n0 ÷ n2), то трябва да отговори с изпращане на един бит, наречен ACK (acknowledge) или още - потвърждение. Неговото ниво е логическа 0. След това се изпращат потребителските данни User Data 1 ... User Data n и най-накрая се генерира условие стоп, с което трансферът приключва.



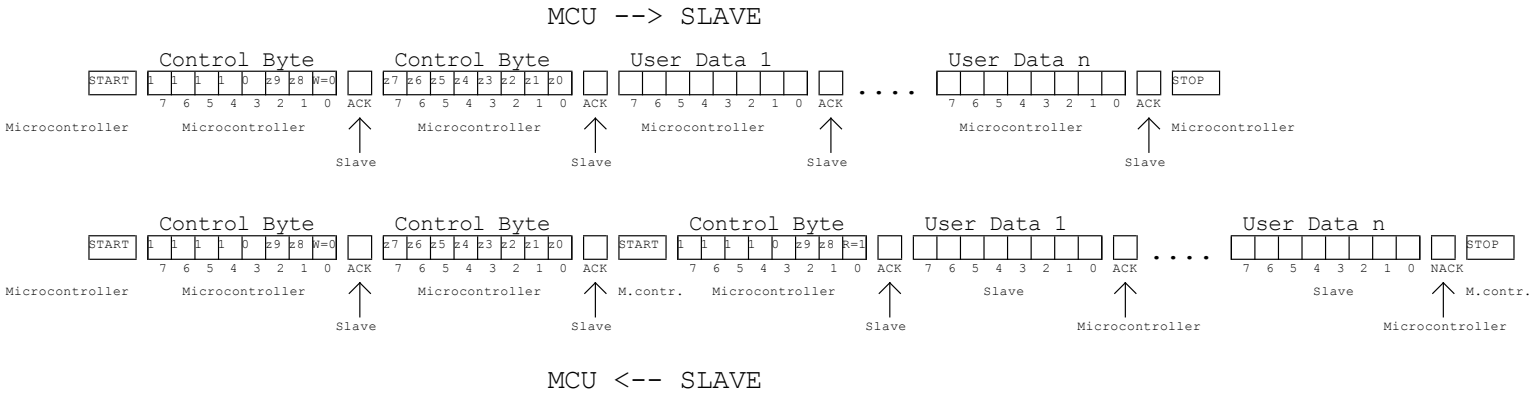
Фиг. 6.6

Трансферът на данните от подчинена ИС към микроконтролера се определя като **четене** и форматът е показан на **фиг. 6.7**. Да се обърне внимание на предавателя и приемника под всеки байт на **фиг. 6.5** и **фиг. 6.7**! Ако микроконтролерът иска да спре приемането на данни, той трябва да генерира NACK бит (No Acknowledge, логическа 1) и след това условие СТОП.



Фиг. 6.7

Отделените седем бита m0 ÷ m3 и n0 ÷ n2 позволяват на една I²C шина да се свържат до 2^7 = 128 чипа (минус един броудкаст адрес и няколко за бъдещо ползване). Версия 1 дефинира някои леки промени в протокола, които позволяват адресиране на 2^{10} = 1024 (т.е. 10-битов адрес) чипа. Те са показани на **фиг. 6.8**. Новото тук е фиксираното число 11110 в контролния байт. То указва, че ще се използва 10-битово адресиране. Следват битове z9 ÷ z0, които са 10-битовия адрес на подчиненото устройство. Да се обърне внимание на двойното изпращане на условие СТАРТ и редуването на W=0 и R=1 при четене (MCU ← SLAVE).

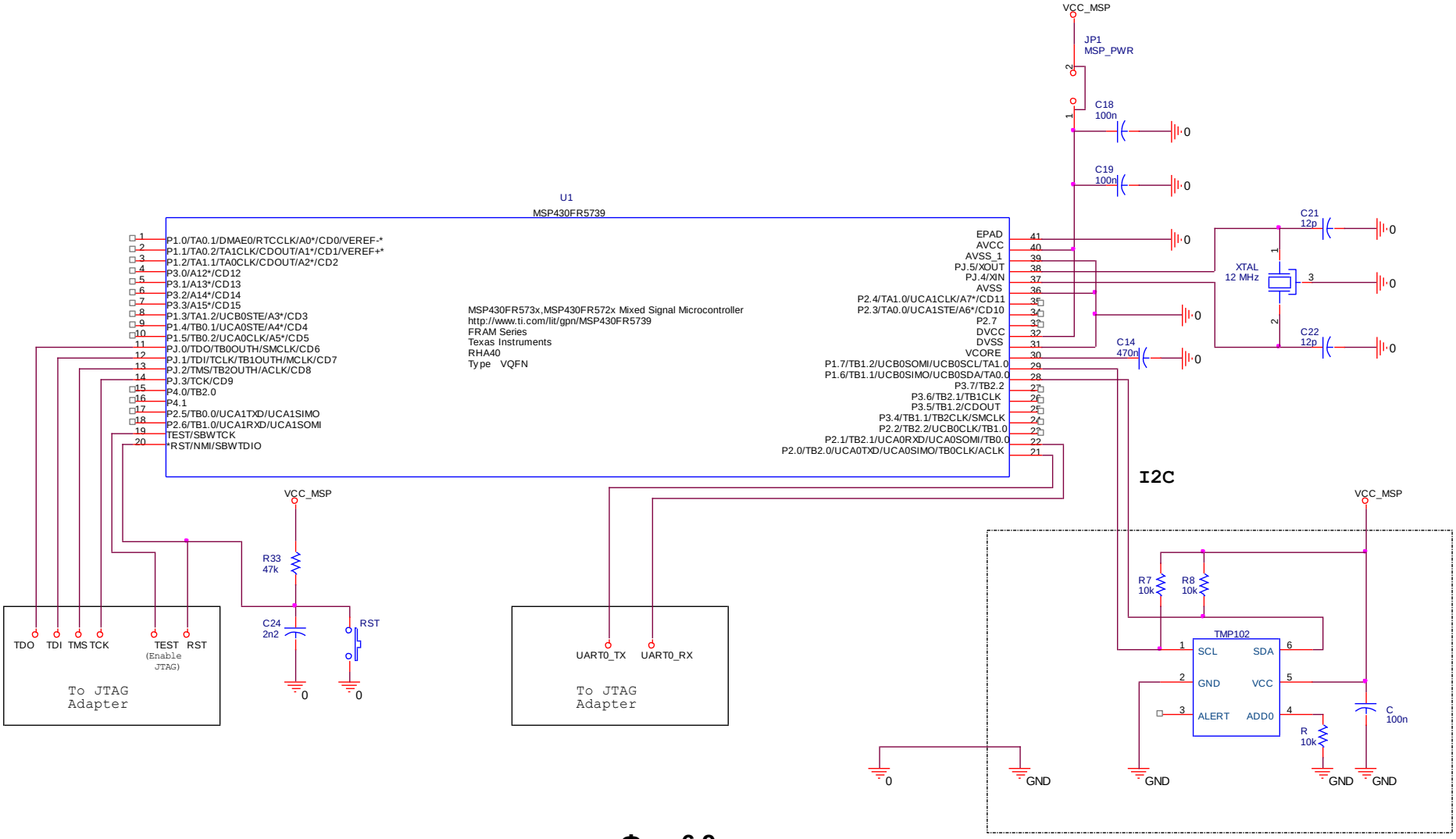


Фиг. 6.8

6.2 Задачи за изпълнение.

6.2.1. Да се създаде нов C проект на Desktop-а в отделна директория с име 03_2_1_Lab и да се напише програма, която чете данни от температурния датчик TMP102 посредством I^2C интерфейс. Принципната схема е показана на **фиг. 6.9**. Да се наблюдават SCL и SDA линиите с осцилоскоп и да се свалят осцилограмите.

6.2.2. Да се създаде нов C проект на Desktop-а в отделна директория с име 03_2_2_Lab, която реализира запис по I^2C интерфейса. За целта да се запише произволно число в един от регистрите T_{LOW} или T_{HIGH} на температурния датчик TMP102. Да се снемат осцилограмите. Да се сравнят с осцилограмите от предишната задача. Да се обяснят разликите.


VCC_MSP
JP1
MSP_PWR
C18
100n
C19
100n
U1
MSP430FR5739
MSP430FR573x,MSP430FR572x Mixed Signal Microcontroller
http://www.ti.com/lit/gpn/MSP430FR5739
FRAM Series
Texas Instruments
RHA40
Type VQFN
P1.0/TA0.1/DMAE0/RTCCLK/A0*/CD0/VEREF-*
P1.1/TA0.2/TA1CLK/CDOUT/A1*/CD1/VEREF+*
P1.2/TA1.1/TA0CLK/CDOUT/A2*/CD2
P3.0/A12*/CD12
P3.1/A13*/CD13
P3.2/A14*/CD14
P3.3/A15*/CD15
P1.3/TA1.2/UCB0STE/A3*/CD3
P1.4/TB0.1/UCA0STE/A4*/CD4
P1.5/TB0.2/UCA0CLK/A5*/CD5
PJ.0/TDO/TB0OUTH/SMCLK/CD6
PJ.1/TDI/TCLK/TB1OUTH/MCLK/CD7
PJ.2/TMS/TB2OUTH/ACLK/CD8
PJ.3/TCK/CD9
P4.0/TB2.0
P4.1
P2.5/TB0.0/UCA1TXD/UCA1SIMO
P2.6/TB1.0/UCA1RXD/UCA1SOMI
TEST/SBWTCK
*RST/NMI/SBWTDIO
EPAD
AVCC
AVSS_1
PJ.5/XOUT
PJ.4/XIN
AVSS
P2.4/TA1.0/UCA1CLK/A7*/CD11
P2.3/TA0.0/UCA1STE/A6*/CD10
P2.7
DVCC
DVSS
VCORE
P1.7/TB1.2/UCB0SOMI/UCB0SCL/TA1.0
P1.6/TB1.1/UCB0SIMO/UCB0SDA/TA0.0
P3.7/TB2.2
P3.6/TB2.1/TB1CLK
P3.5/TB1.2/CDOUT
P3.4/TB1.1/TB2CLK/SMCLK
P2.2/TB2.2/UCB0CLK/TB1.0
P2.1/TB2.1/UCA0RXD/UCA0SOMI/TB0.0
P2.0/TB2.0/UCA0TXD/UCA0SIMO/TB0CLK/ACLK
C21
12p
C22
12p
XTAL
12 MHz
C14
470n
I2C
VCC_MSP
R33
47k
C24
2n2
RST
TDO
TDI
TMS
TCK
TEST
(Enable
JTAG)
RST
To JTAG
Adapter
UART0_TX
UART0_RX
To JTAG
Adapter
VCC_MSP
R7
10k
R8
10k
TMP102
SCL
SDA
GND
VCC
ALERT
ADD0
R
10k
C
100n
GND
GND
GND
GND
0

Фиг. 6.9

Задача 6.2.1

```
#include <stdint.h>
#include "io430.h"

#define TMP102ADDR ??? //Дефиниране на TMP102 адреса, Datasheet на TMP102 → стр. 10

// Set DCO to 8 MHz
void init()
{
  CSCTL0_H = 0xA5;
  CSCTL1 &= ~DCORSEL; //DCORSEL = 0
  CSCTL1 |= DCOFSEL0 + DCOFSEL1;     //DCOFSEL0 = 1, DCOFSEL1 = 1
  CSCTL2 = SELA_3 + SELS_3 + SELM_3; // set ACLK = DCO; SMCLK = DCO; MCLK = DCO
  CSCTL3 = DIVA_0 + DIVS_0 + DIVM_0; // set all dividers to 1
}

void init_I2C()
{
P1SEL0&=???;//Превключи мултиплексора към I2C модула, MSP430FR5739 Datasheet → стр. 73
P1SEL1 |= ???;//Превключи мултиплексора към I2C модула, MSP430FR5739 Datasheet → стр. 73


  UCB0CTLW0 |= UCSWRST; //Задръж I2C модула в reset, докато е конфигуриран
  UCB0CTLW0 |= (UCMODE_3 | UCMST); //Избери режим I2C, роля - master
  UCB0CTLW0 |= UCSSEL_3; //Източник на тактов сигнал за I2C модула е SMCLK
  UCB0BRW = 0x50; //SMCLK / 80 = 100 kbps
  UCB0I2CSA = TMP102ADDR; //Укажи адреса на TMP102 датчика (slave устройството)
  UCB0CTLW0 &= ~UCSWRST; //Изведи модула от reset
}

void init_TMP102()
{
  UCB0CTLW0 |= (UCTXSTT | UCTR); //Генерирай start условие, режим предавател
  __delay_cycles(500); //Изчакай предаването на start условието
  UCB0TXBUF = 0x00; //Запиши данните, които ще се изпращат
  while(!(UCB0IFG & UCTXIFG0)){ } //Изчакай данните от UCB0TXBUF да се изпратят
  UCB0CTLW0 |= UCTXSTP; //Генерирай stop условие
  __delay_cycles(1000);
}

void main( void )
{
  uint8_t lsb;
  uint16_t msb, result;

  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  init();
  init_I2C();
  init_TMP102();

  while(1)
  {
    UCB0CTLW0 &= ~????; //Режим приемник, MSP430FR5739 Userguide → стр. 483

    UCB0CTLW0 |= UCTXSTT; //Генерирай start условие
    while(!(UCB0IFG & UCRXIFG0)){ } //Изчакай първия байт да пристигне в UCB0RXBUF
    lsb = UCB0RXBUF;
```

```
    while(!(UCB0IFG & UCRXIFG0)){ } //Изчакай втория байт да пристигне в UCB0RXBUF
    msb = UCB0RXBUF;
    UCB0CTLW0 |= UCTXSTP; //Генерирай stop условие

    result = (msb<<8) | lsb; //Обедини младшата и старшата част на резултата в една променлива
    result >>= 4; //Младшите 4 бита са винаги 0, затова ги премахни
    __delay_cycles(8000);
  }
}
```

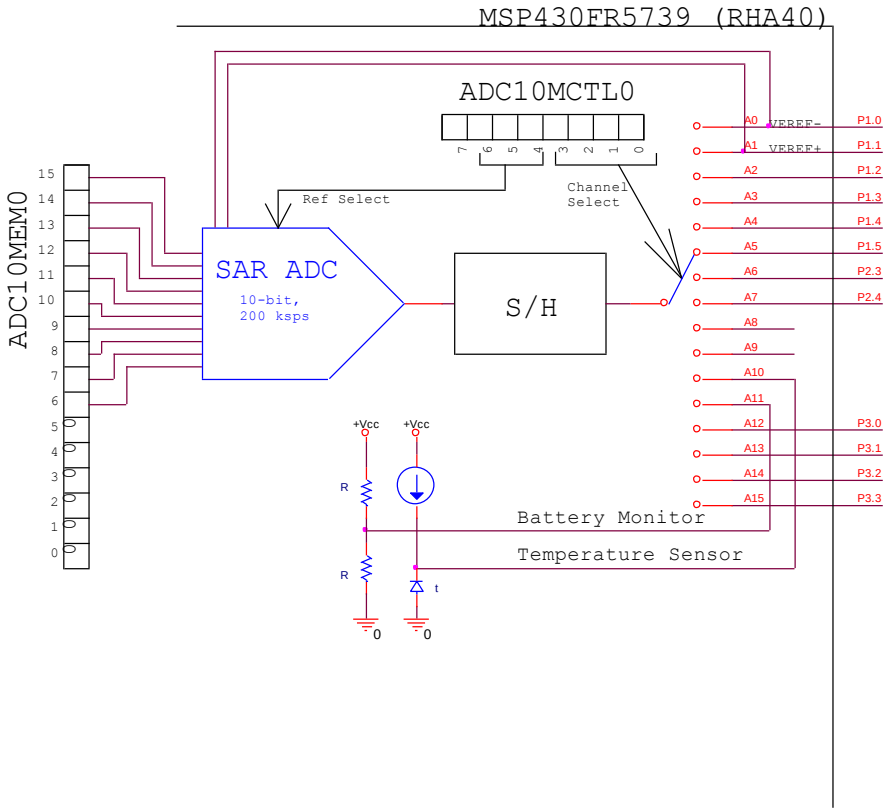
Задача 6.2.2

```
#include <stdint.h>
#include "io430.h"

#define TMP102ADDR ??? //Дефиниране на TMP102 адреса, Datasheet на TMP102 → стр. 10

// Set DCO to 8 MHz
void init()
{
  CSCTL0_H = 0xA5;
  CSCTL1 &= ~DCORSEL; //DCORSEL = 0
  CSCTL1 |= DCOFSEL0 + DCOFSEL1;     //DCOFSEL0 = 1, DCOFSEL1 = 1
  CSCTL2 = SELA_3 + SELS_3 + SELM_3; // set ACLK = DCO; SMCLK = DCO; MCLK = DCO
  CSCTL3 = DIVA_0 + DIVS_0 + DIVM_0; // set all dividers to 1
}

void init_I2C()
{
P1SEL0&=???;//Превключи мултиплексора към I2C модула, MSP430FR5739 Datasheet → стр. 73
P1SEL1 |= ???;//Превключи мултиплексора към I2C модула, MSP430FR5739 Datasheet → стр. 73

  UCB0CTLW0 |= UCSWRST; //Задръж I2C модула в reset, докато е конфигуриран
  UCB0CTLW0 |= (UCMODE_3 | ????); //Избери режим I2C, роля – master, MSP430FR5739
                                  //Userguide → стр. 483
  UCB0CTLW0 |= UCSSEL_3; //Източник на тактов сигнал за I2C модула е SMCLK
  UCB0BRW = 0x50; //SMCLK / 80 = 100 kbps
  UCB0I2CSA = TMP102ADDR; //Укажи адреса на TMP102 датчика (slave устройството)
  UCB0CTLW0 &= ~UCSWRST; //Изведи модула от reset
}

void init_TMP102()
{
  UCB0CTLW0 |= (UCTXSTT | UCTR); //Генерирай start условие, режим предавател
  __delay_cycles(500); //Изчакай предаването на start условието
  UCB0TXBUF = 0x03; //Указател към регистър Thigh на датчика TMP102
  while(!(UCB0IFG & UCTXIFG0)){ } //Изчакай данните от UCB0TXBUF да се изпратят
  UCB0CTLW0 |= UCTXSTP; //Генерирай stop условие
}

void main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;
  init();
  init_I2C();
  init_TMP102();
```

```
  while(1)
  {
    UCB0CTLW0 |= (UCTXSTT | ????); //Генерирай start условие, режим предавател, MSP430FR5739
                                   //Userguide → стр. 483
    __delay_cycles(500); //Изчакай предаването на start условието
    UCB0TXBUF = 0x55;
    while(!(UCB0IFG & UCTXIFG0)){ } //Изчакай данните от UCB0TXBUF да се изпратят
    UCB0TXBUF = 0xF0;
    while(!(UCB0IFG & UCTXIFG0)){ } //Изчакай данните от UCB0TXBUF да се изпратят
    UCB0TXBUF = 0x0F;
    while(!(UCB0IFG & UCTXIFG0)){ } //Изчакай данните от UCB0TXBUF да се изпратят

    UCB0CTLW0 |= UCTXSTP; //Генерирай stop условие
    __delay_cycles(8000);
  }
}
```

Лабораторно упражнение №7
“Аналогово-цифров преобразувател на MSP430FR5739 и комуникационен модул UART (RS-232)”

7.1. Въведение. Микроконтролерът MSP430FR5739 притежава вграден аналогово-цифров SAR преобразувател с 10-битова разредност, бързодействие 200 ksps и 16 входни канала (**фиг. 7.1**).



Фиг. 7.1

Потребителят може да измерва напрежение на 12 извода. Другите 4 са за измерване температурата на чипа (с помощта на интегриран диод), за измерване на захранващото напрежение и два не се използват. При включване на външни еталонни източници за горен и долен праг на измерването потребителят ще може да измерва напрежение само на 10 извода.

Преди даден извод да бъде използван от АЦП-то, трябва да се конфигурира *мултиплексорът*, превключващ извода измежду различните периферни модули (всеки извод може да бъде използван от много периферни модули, но в даден момент само един е включен на него). Това става чрез регистър PxSEL0 и PxSEL1.

АЦП може да работи в *4 режима*, задавани от битовете ADC10CONSEQx на регистъра ADC10CTL1:

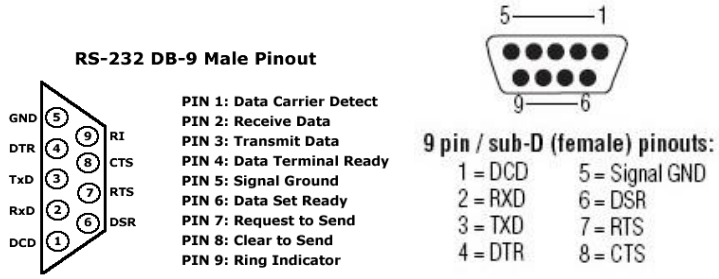
- един канал, едно измерване;
- много канали, по едно измерване на всеки;
- един канал, много измервания;
- много канали, много измервания на всеки.

АЦП-то може да използва *еталонния източник* на напрежение, вграден в микроконтролера. Неговата стойност може да се избере от стойностите: 1.5 V, 2.0 V, 2.5 V. Това може да стане, като се установи бит 0 от REFCTL0 в единица и с REFVSEL битовете се избере една от трите стойности. Да се разгледа MSP430FR57XX User's Guide-а за желаната комбинация на REFVSEL.

Тактовата честота на модула е един от източниците SMCLK, MCLK, ACLK, MODOSC и може да се избере чрез битовете ADC10SSELx от регистъра ADC10CTL1 (нека да е еднаква с честотата на микропроцесора).

Тази честота допълнително може да бъде *разделена* 1 ÷ 512 пъти чрез ADC10DIVx битове от регистъра ADC10CTL1 и ADC10PDIVx битове от регистъра ADC10CTL2 (нека да е разделена на 1).

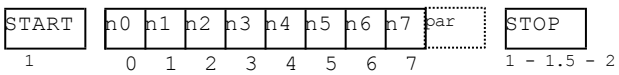
В настоящото упражнение се използва RS232 интерфейс, наречен още UART, за осъществяване на връзка между микроконтролера и персонален компютър. Това е напрежителен, сериен, асинхронен интерфейс. На **фиг. 4.2** са показани най-често използваните куплунги с 9 извода (има и с 25 извода) - мъжки и женски DB9.



Фиг. 7.2

В повечето приложения вградените системи използват само три проводника от този куплунг – това са TxD (извод 3) за предаване, RxD (извод 2) за приемане и маса (извод 5). Логическите нива се представят с отрицателни и положителни напрежения за разлика от TTL нивата, където са само положителни. Логическата 0 (наричана Space) е дефинирана като ≥ +5 V, а логическата 1 (наричана Mark) като ≤ -5 V. Максималните нива на напреженията трябва да са в границите ± 15 V, но чиповете, използващи този интерфейс трябва да са способни да издържат до ±25 V. Приемниците трябва да регистрират и затихнали сигнали с амплитуда не по-малка от +3 V и не по-голяма от -3 V (т.е. имат минимален 5 – 3 = 2 V запас за шумоустойчивост).

Предаването на данни се осъществява байт по байт, всеки от който съдържа един стартов бит, даннови битове, бит проверка за грешки и един или повече стопови бита (**фиг. 7.3**). Изпращането на данните се извършва от LSB към MSB. Повечето инженери са свикнали да представят байтовете в MSB към LSB формат, затова може да се каже, че при наблюдение на осцилоскопа данните от UART изглеждат обърнати. Когато не се предават данни линиите се държат в логическа 1.



Фиг. 7.3

Битът „*проверка за грешки*” може да бъде 4 вида:

- проверка по четност – битът се установява в 1 или 0, така че сумата от всички единици в изпращания байт да е четно число;
- проверка по нечетност – битът се установява в 1 или 0, така че сумата от всички единици в изпращания байт да е нечетно число;
- проверка с единица (mark parity) – битът е винаги единица;
- проверка с нула (space parity) – битът е винаги нула.

Пример: ако се изпраща числото 0001 0010 с проверка по четност, деветият бит ще е нула. Аналогично при проверката по нечетност числото 0011 0011 ще има единица за деветия бит.

Стоповите битове може да са :

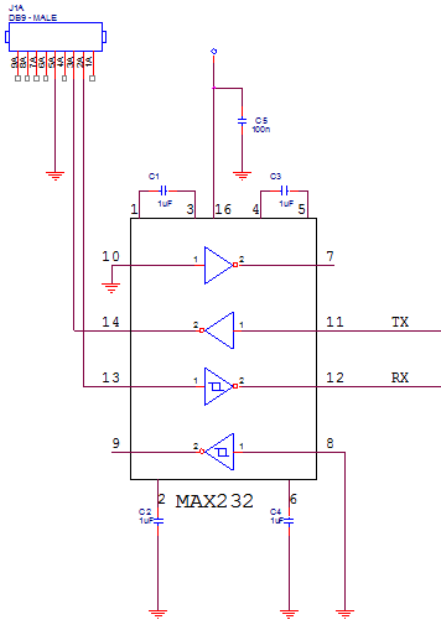
- един;
- един и половина;
- два.

Един и половина и два стоп бита се използват при по-бавни приемащи устройства, които имат нужда от малко по-дълъг (във времето) стоп бит, за да обработят приетите данни.

Данновите битове може да са 5 ÷ 8. Изпращащото устройство предава данните на всеки свой нарастващ фронт на вътрешния му генератор, а приемащото устройство регистрира тези данни на всеки падащ фронт на вътрешния му генератор. Синхронизацията на вътрешните генератори на приемника и предавателя се осъществява по спадащия фронт на старт бита (т.е. при първия преход -15 V → +15 V). Така интерфейсът не се нуждае от допълнителен проводник за тактов сигнал и затова наричаме UART-а асинхронен интерфейс.

Скоростта на предаване се дава в бодове и 1 бод = 1 бит информация. В практиката се използват стандартизирани скорости, някои от които – 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200. По-високи скорости на обмен се осъществяват с USB-UART емулатори. Максималната дължина на кабела между две устройства е 10 m.

Понеже микроконтролерите се захранват с напрежения до 5 V, то те не могат да изработят сигнали с амплитуда ± 5 V ÷ ±15 V. Също така не биха могли да приемат такива сигнали без да се повредят. Затова се използват специални транслаторни схеми, които изработват необходимите сигнали и за интерфейса, и за микроконтролера. Една примерна такава ИС е показана на **фиг. 7.4**. Използван е MAX232, а на схемата е дадена и вътрешната му структура. Инверторите са всъщност специални и осигуряват транслацията на нивата. Използват се charge-pump постояннотокови преобразуватели за повишаване и изработване на отрицателни напрежения, като за целта се свързват кондензаторите C1, C2, C3 и C4.



Фиг. 7.4

7.2 Задачи за изпълнение.

7.2.1. Да се създаде нов C проект на Desktop-а в отделна директория с име 07_2_1_Lab. Да се напише програма, която изпраща „Hello, World!" съобщение по UART (RS232) интерфейса.

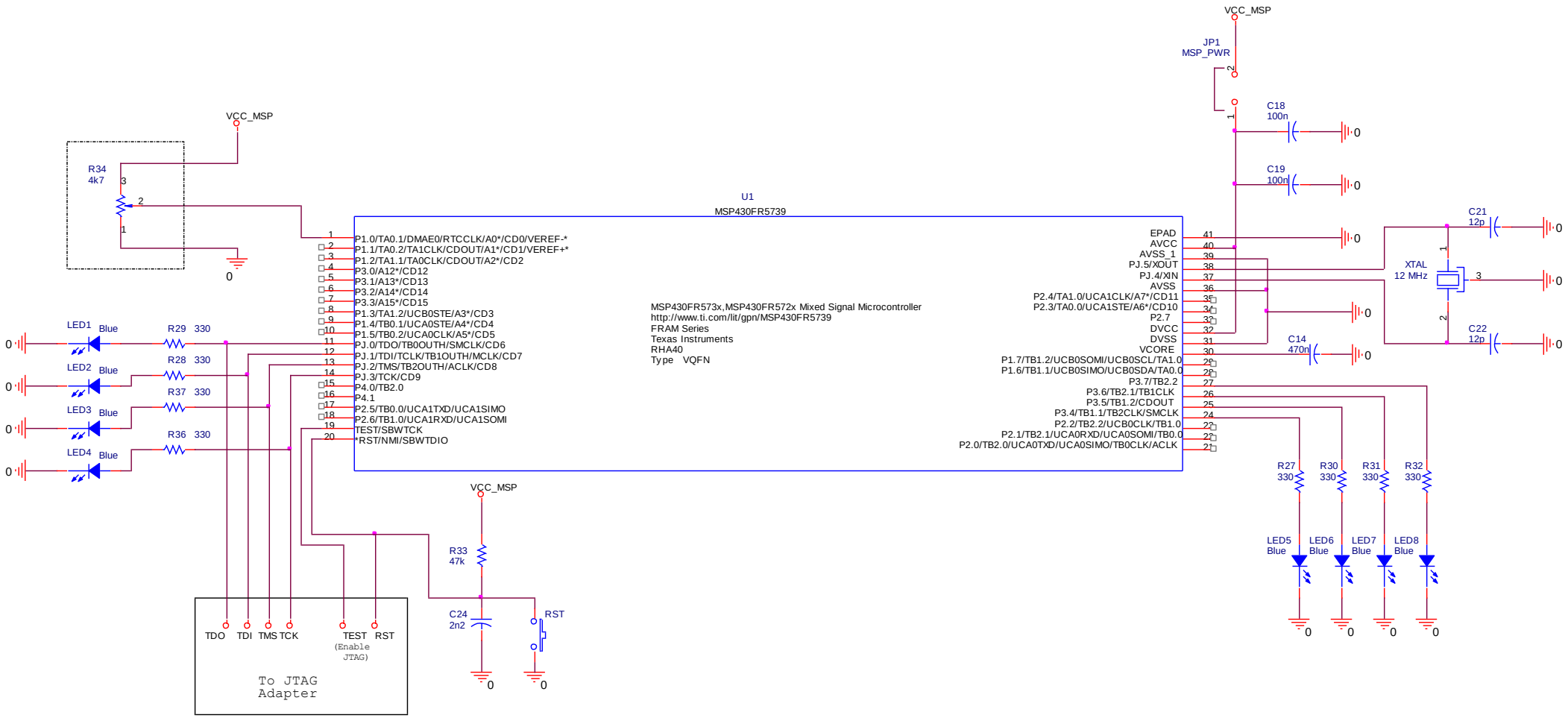
За целта трябва да се копират файловете uart.c, uart.h, uartstdio.c и uartstdio.h (предоставени от ръководителя на упражнението) в директорията на проекта. След това в IAR → десен бутон върху top-level директорията → Add Files → указва се пътя до току-що добавените файлове → Open. В main.c се включва хедърния файл uartstdio.h. Използвайки API функциите от този файл реализирайте програмата си. За да видите резултата стартирайте терминална програма (Hyper Terminal, Tera Term, Real Term и т.н.) и настройте връзка 9600 бода, 8 даннови бита, без проверка по четност, 1 стопов бит.

»RS232 в това упражнение е виртуален и емулиран от USB.

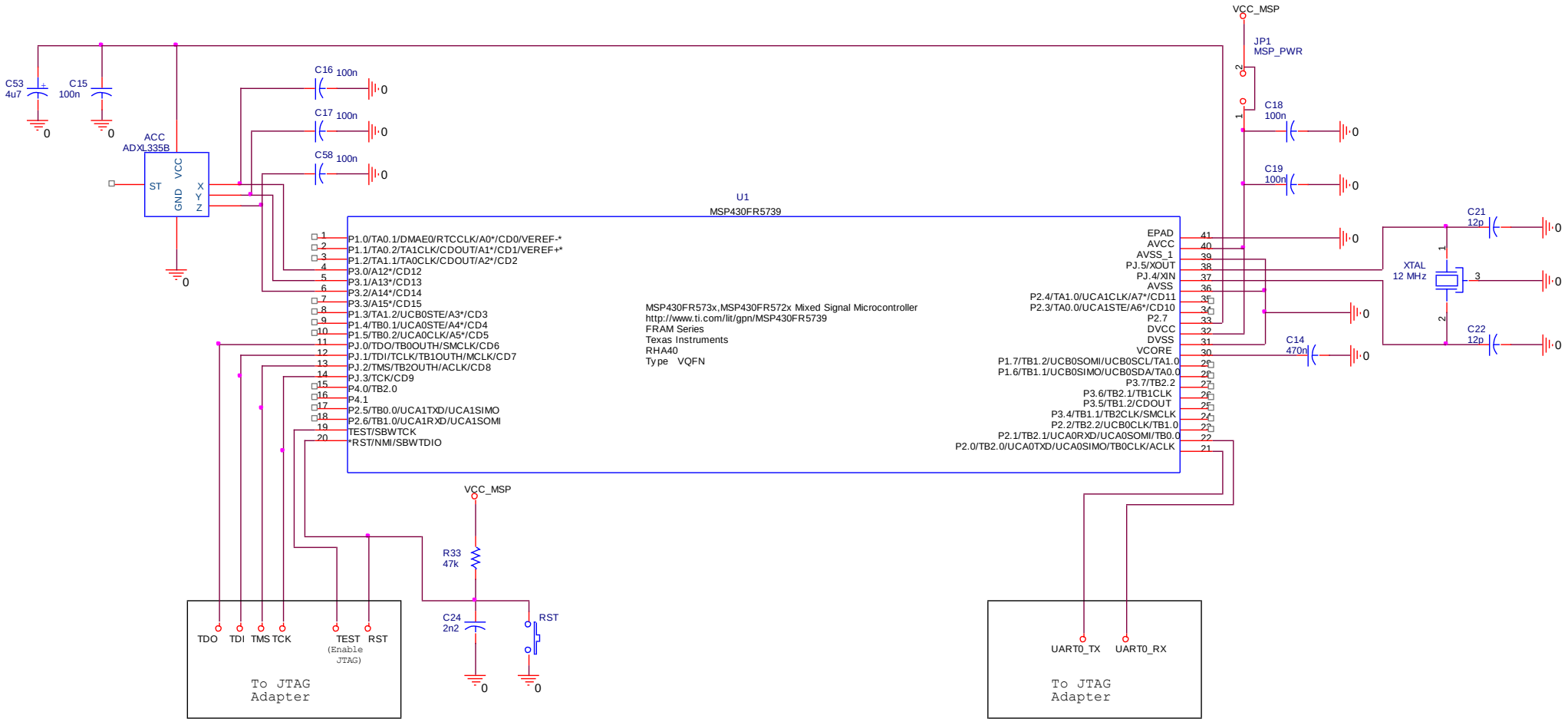
7.2.2. Да се създаде нов C проект на Desktop-а в отделна директория с име 07_2_2_Lab. Да се свърже потенциометър към демо платката (**фиг.7.5**). Да се напише програма, която измерва напрежението в средната точка на потенциометъра. Да се наблюдава резултата от измерването посредством JTAG дебъгера.

7.3.3. Да се създаде нов C проект на Desktop-а в отделна директория с име 03_2_3_Lab и да се добави UARTStdio, както беше направено в задача 7.2.1. Да се напише програма, която използва акселерометъра на демо платката (**фиг. 7.6**). Данните се изпращат от микроконтролера към компютъра по виртуалния UART. Данните от измерванията на трите канала се представят в необработен вид като цели числа в интервала 0 ÷ 1023.

За по-лесно тестване на програмата на **фиг. 7.7** е дадена схема от каталога на акселерометъра ADXL335, изобразяваща положението на чипа и очакваните (нормализирани) напрежения в изхода.


VCC_MSP
JP1
MSP_PWR
R34
4k7
U1
MSP430FR5739
MSP430FR573x,MSP430FR572x Mixed Signal Microcontroller
http://www.ti.com/lit/gpn/MSP430FR5739
FRAM Series
Texas Instruments
RHA40
Type VQFN
P1.0/TA0.1/DMAE0/RTCCLK/A0*/CD0/VEREF-*
P1.1/TA0.2/TA1CLK/CDOUT/A1*/CD1/VEREF+*
P1.2/TA1.1/TA0CLK/CDOUT/A2*/CD2
P3.0/A12*/CD12
P3.1/A13*/CD13
P3.2/A14*/CD14
P3.3/A15*/CD15
P1.3/TA1.2/UCB0STE/A3*/CD3
P1.4/TB0.1/UCA0STE/A4*/CD4
P1.5/TB0.2/UCA0CLK/A5*/CD5
PJ.0/TDO/TB0OUTH/SMCLK/CD6
PJ.1/TDI/TCLK/TB1OUTH/MCLK/CD7
PJ.2/TMS/TB2OUTH/ACLK/CD8
PJ.3/TCK/CD9
P4.0/TB2.0
P4.1
P2.5/TB0.0/UCA1TXD/UCA1SIMO
P2.6/TB1.0/UCA1RXD/UCA1SOMI
TEST/SBWTCK
*RST/NMI/SBWTDIO
EPAD
AVCC
AVSS_1
PJ.5/XOUT
PJ.4/XIN
AVSS
P2.4/TA1.0/UCA1CLK/A7*/CD11
P2.3/TA0.0/UCA1STE/A6*/CD10
P2.7
DVCC
DVSS
VCORE
P1.7/TB1.2/UCB0SOMI/UCB0SCL/TA1.0
P1.6/TB1.1/UCB0SIMO/UCB0SDA/TA0.0
P3.7/TB2.2
P3.6/TB2.1/TB1CLK
P3.5/TB1.2/CDOUT
P3.4/TB1.1/TB2CLK/SMCLK
P2.2/TB2.2/UCB0CLK/TB1.0
P2.1/TB2.1/UCA0RXD/UCA0SOMI/TB0.0
P2.0/TB2.0/UCA0TXD/UCA0SIMO/TB0CLK/ACLK
C18
100n
C19
100n
C14
470n
C21
12p
C22
12p
XTAL
12 MHz
LED1
LED2
LED3
LED4
Blue
R29 330
R28 330
R37 330
R36 330
R27 330
R30 330
R31 330
R32 330
LED5
LED6
LED7
LED8
R33
47k
C24
2n2
RST
TDO
TDI
TMS
TCK
TEST
(Enable
JTAG)
To JTAG
Adapter

Фиг. 7.5


C53
4u7
C15
100n
C16 100n
C17 100n
C58 100n
ACC
ADXL335B
ST
VCC
GND
X
Y
Z
U1
MSP430FR5739
MSP430FR573x,MSP430FR572x Mixed Signal Microcontroller
http://www.ti.com/lit/gpn/MSP430FR5739
FRAM Series
Texas Instruments
RHA40
Type VQFN
P1.0/TA0.1/DMAE0/RTCCLK/A0*/CD0/VEREF-*
P1.1/TA0.2/TA1CLK/CDOUT/A1*/CD1/VEREF+*
P1.2/TA1.1/TA0CLK/CDOUT/A2*/CD2
P3.0/A12*/CD12
P3.1/A13*/CD13
P3.2/A14*/CD14
P3.3/A15*/CD15
P1.3/TA1.2/UCB0STE/A3*/CD3
P1.4/TB0.1/UCA0STE/A4*/CD4
P1.5/TB0.2/UCA0CLK/A5*/CD5
PJ.0/TDO/TB0OUTH/SMCLK/CD6
PJ.1/TDI/TCLK/TB1OUTH/MCLK/CD7
PJ.2/TMS/TB2OUTH/ACLK/CD8
PJ.3/TCK/CD9
P4.0/TB2.0
P4.1
P2.5/TB0.0/UCA1TXD/UCA1SIMO
P2.6/TB1.0/UCA1RXD/UCA1SOMI
TEST/SBWTCK
*RST/NMI/SBWTDIO
EPAD
AVCC
AVSS_1
PJ.5/XOUT
PJ.4/XIN
AVSS
P2.4/TA1.0/UCA1CLK/A7*/CD11
P2.3/TA0.0/UCA1STE/A6*/CD10
P2.7
DVCC
DVSS
VCORE
P1.7/TB1.2/UCB0SOMI/UCB0SCL/TA1.0
P1.6/TB1.1/UCB0SIMO/UCB0SDA/TA0.0
P3.7/TB2.2
P3.6/TB2.1/TB1CLK
P3.5/TB1.2/CDOUT
P3.4/TB1.1/TB2CLK/SMCLK
P2.2/TB2.2/UCB0CLK/TB1.0
P2.1/TB2.1/UCA0RXD/UCA0SOMI/TB0.0
P2.0/TB2.0/UCA0TXD/UCA0SIMO/TB0CLK/ACLK
VCC_MSP
JP1
MSP_PWR
C18
100n
C19
100n
C14
470n
C21
12p
C22
12p
XTAL
12 MHz
R33
47k
C24
2n2
RST
TDO
TDI
TMS
TCK
TEST
(Enable
JTAG)
RST
To JTAG
Adapter
UART0_TX
UART0_RX
To JTAG
Adapter

Фиг. 7.6

The output noise is not ratiometric but is absolute in volts; therefore, the noise density decreases as the supply voltage increases. This is because the scale factor (mV/*g*) increases while the noise voltage remains constant. At V_S = 3.6 V, the X-axis and Y-axis noise density is typically 120 µg/√Hz, whereas at V_S = 2 V, the X-axis and Y-axis noise density is typically 270 µg/√Hz.

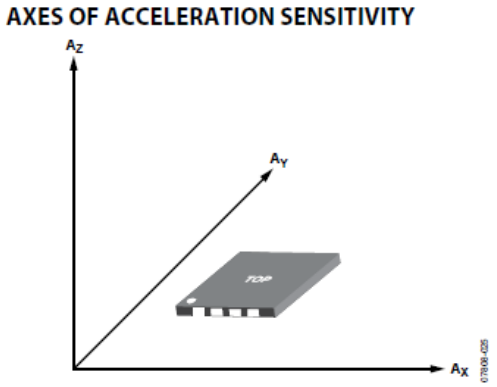
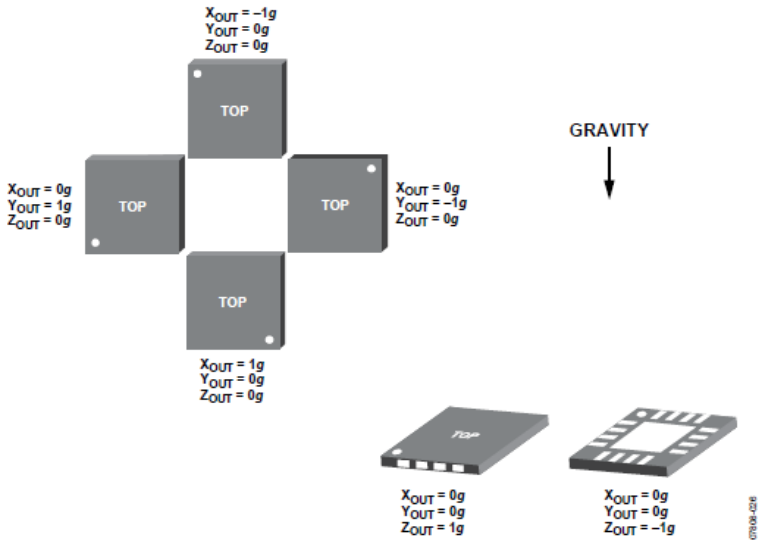


Figure 23. Axes of Acceleration Sensitivity; Corresponding Output Voltage Increases When Accelerated Along the Sensitive Axis.



Фиг. 7.7

Задача 7.2.1

```
#include "io430.h"
??? //Включи хедърния файл на UARTStdio, който сте добавили към проекта

void main( void )
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  ??? //Инициализирай UARTStdio функцията → отвори хедърния файл и виж прототипите на
        //достъпните функции

  while(1)
  {
    ??? //Извикай printf за MSP430 с произволен стринг → отвори хедърния файл и виж
          // прототипите на достъпните функции, извикайте printf( ) функцията, както бихте я
          //извикали в консозолна С програма на вашия PC

    __delay_cycles(24000000);
  }
}
```

Задача 7.2.2

```
#include "io430.h"

void initADC( )
{
  //Избери функцията на извод P1.0 като вход на АЦП
  P1SEL0 = 0x01;
  P1SEL1 = 0x01;

  //ENC = 0
  ADC10CTL0 &= ~ADC10ENC;

  //Включи АЦП
  //Избери Sample-and-hold време = 1024 такта на АЦП-то
  ADC10CTL0 |= ADC10ON | ADC10SHT0 | ADC10SHT1 | ADC10SHT2 | ADC10SHT3;

  //Избери MCLK за тактов сигнал
  //и използвай sampling таймера
  ADC10CTL1 &= ~ADC10SSEL0;
  ADC10CTL1 |= ADC10SSEL1 | ADC10SHP;

  //Избери режим "един канал - едно измерване"
  //Раздели тактовия сигнал на 1
  //Не инвертирай sample-and-hold сигнала
  //Използвай ADC10SC бита като сигнал за начало на sample-and-hold
  ADC10CTL1 &= ~(ADC10CONSEQ0 | ADC10CONSEQ1 | ADC10DIV0 | ADC10DIV1 | ADC10DIV2 |
ADC10ISSH | ADC10SHS0 | ADC10SHS1);

  //Избери товароносимост на буфера на еталонния източник за 200 kbps
  //Формат на резултата - прав код
  //Раздели тактовия сигнал на 1
  ADC10CTL2 &= ~(ADC10SR | ADC10DF | ADC10PDIV0 | ADC10PDIV1);

  //Избери 10-битово преобразуване
  ADC10CTL2 |= ???; //MSP430FR5739 User Guide → стр. 404
```

```
  //Използвай Vdd за горен праг, Vss - за долен.
  ADC10MCTL0 &=  ~(ADC10SREF0 | ADC10SREF1 | ADC10SREF2);
  //Избери P1.0 за входен канал
  ADC10MCTL0 |= ADC10INCH_0;

  //Установи ENC в 1
  ADC10CTL0 |= ADC10ENC;
}

void main( void )
{
  //Тук ще се съхранява резултата от измерването
  volatile unsigned int ADCResult;

  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  //Инициализирай АЦП-то
  initADC();

  while(1)
  {
    //Започни измерване
    ADC10CTL0 |= ???; //MSP430FR5739 User Guide → стр. 402

   //Изчакай, докато приключи измерването
    while(ADC10CTL1 & ???){ } //MSP430FR5739 User Guide → стр. 403
   //Запиши резултата в ADCResult
    ADCResult =  ADC10MEM0;
   }
}
```

Задача 7.2.3

```
#include "io430.h"
#include "uartstdio.h"

void initADC( )
{
  //Избери функцията на изводи P3.0, P3.1, P3.2 като вход на АЦП
  P3SEL0 = ???; //MSP430FR5739 Datasheet → стр. 79
  P3SEL1 = ???; //MSP430FR5739 Datasheet → стр. 79

  //ENC = 0
  ADC10CTL0 &= ~ADC10ENC;

  //Включи АЦП
  //Избери Sample-and-hold време = 32 такта на АЦП-то
  //Ако Fadc = 4 MHz, Tsample = 32*(1/4MHz) = 8 us
  ADC10CTL0 |= (ADC10ON | ADC10SHT0 | ADC10SHT1);
  ADC10CTL0 &= ~(ADC10SHT2 | ADC10SHT3);

  //Избери MCLK за тактов сигнал
  //и използвай sampling таймера
  ADC10CTL1 &= ~ADC10SSEL0;
  ADC10CTL1 |= ADC10SSEL1 | ADC10SHP;

  //Избери режим "много канали, по едно измерване на всеки"
  //Раздели тактовия сигнал на 6 (24 MHz/6 = 4 MHz)
  //Не инвертирай sample-and-hold сигнала
  //Използвай ADC10SC бита като сигнал за начало на sample-and-hold
```

```
  ADC10CTL1 |= (??? | ADC10DIV0 | ADC10DIV2); //MSP430FR5739 User Guide → стр. 403
 //MSP430FR5739 User Guide → стр. 403
  ADC10CTL1 &= ~(??? | ADC10DIV1 | ADC10ISSH | ADC10SHS0 | ADC10SHS1);

  //Избери товароносимост на буфера на еталонния източник за 200 kbps
  //Формат на резултата - прав код
  //Раздели тактовия сигнал на 1
  ADC10CTL2 &= ~(ADC10SR | ADC10DF | ADC10PDIV0 | ADC10PDIV1);

  //Избери 10-битово преобразуване
  ADC10CTL2 |= ???; //MSP430FR5739 User Guide → стр. 404

  //Използвай Vdd за горен праг, Vss - за долен
  ADC10MCTL0 &=  ~(ADC10SREF0 | ADC10SREF1 | ADC10SREF2);
  //Пусни АЦП да мери всички канали от A15 надолу.
  ADC10MCTL0 |= ADC10INCH_15;
  //Установи ENC в 1
  ADC10CTL0 |= ADC10ENC;
}

void initADXL()
{
  //Включи ADXL335 с логическа 1 на извод P2.7
  P2DIR |= 0x80;
  P2OUT |= 0x80;
}

void main( void )
{
  signed int i;
  //Тук ще се съхраняват резултатите от измерванията
  unsigned long arr[15];

  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;

  //Инициализирай АЦП-то
  initADC();
  //Инициализирай ADXL335
  initADXL();
  //Инициализирай UART модула и UARTprintf функцията
  UARTStdioInit();

  while(1)
  {
    for(i = 15; i >= 0; i--)
    {
    //Започни измерване
      ADC10CTL0 |= ???; //MSP430FR5739 User Guide → стр. 404
      while(!(ADC10IFG & ADC10IFG0)){ }
      ADC10IFG = 0x00;
      arr[i] =  ADC10MEM0;
    }

    UARTprintf("X: %04d Y: %04d Z: %04d \r", arr[12], arr[13], arr[14]);

     __delay_cycles(2400000);
  }
}
```

Лабораторно упражнение №8

"Безжичен LAN (WLAN) модул CC3000 и MSP430FR5739. Създаване на Web сървър."

8.1. Въведение. Към демо платката MSP-EXP430FR5739 може да се свърже комуникационна демо платка SimpleLink (произведена от фирмата Texas Instruments), позволяваща реализиране на безжична LAN мрежа (или още Wireless LAN, WLAN, но да не се бърка с WAN) по стандарта IEEE 802.11 b/g. Такъв вид мрежа се поддържа от безжичните домашни рутери, което позволява да се създаде вградена система на базата на MSP430FR5739 с връзка към Интернет (**фиг. 8.1**). Ядрото на SimpleLink е мулти-чипният модул CC3000 на фирмата LS Research. Към него трябва да се свърже единствено външна антена и няколко кондензатора. Връзката MSP430FR5739 ←→ CC3000 се осъществява чрез SPI интерфейс. SimpleLink използва SMD антена (погледнете я на платката - отбелязана е като ANT1), която реализира безжичния интерфейс CC3000 ←→ Wifi рутер. Максималната скорост на Интернет връзката е 54 Mbps (на носеща честота 2.4 GHz), което е много над възможностите на MSP430FR5739 (макс. тактова честота 24 MHz) и реално обменът на данни става на по-ниска скорост.

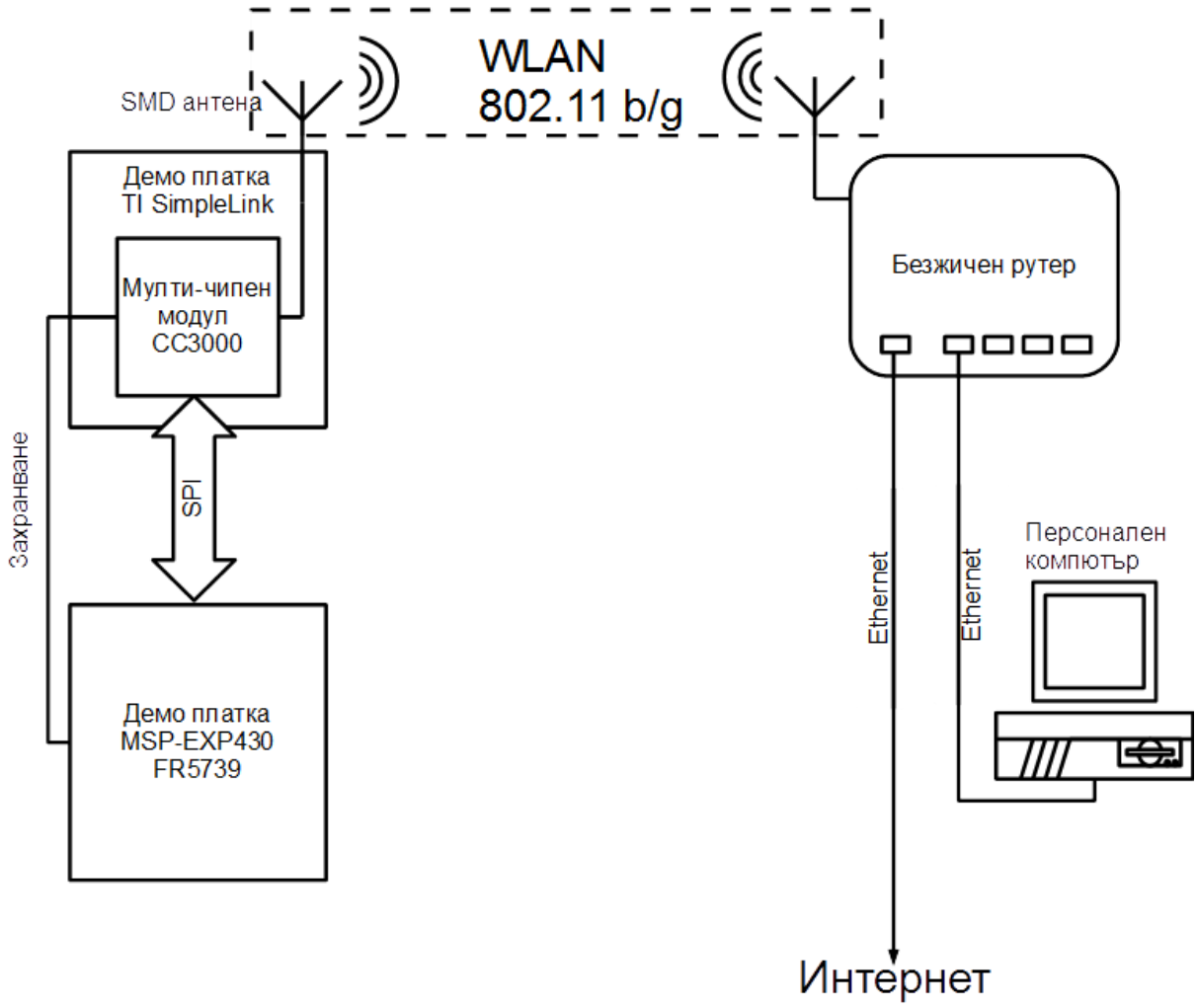
8.2. Основни понятия за Интернет. Някои от по-важните термини за Интернет мрежата са:

- **LAN** (Local Area Network) - мрежа, обхващаща различен брой компютри, разположени в област не по-голяма от 10 km. Броят на компютрите варира много и зависи от мрежовата структура и използваните кабели. LAN може да се нарече мрежа от три компютъра в три съседни блока, както и 1000 компютъра в един квартал.

- **WLAN** (Wireless Local Area Network) - класическа LAN мрежа, но използваща ефира за преносна среда. Комуникацията се осъществява в определен радиоканал.

- **WAN** (Wide Area Network) – международни мрежи, обхващащи голяма географска област. Наи-известният пример е Интернет. Друг пример са корпоративните мрежи (Интранет) на фирми, имащи клонове по цял свят и които са си изградили WAN за спомагане дейността на фирмата. Такива мрежи осигуряват обмен от около 1 ÷ 6 Mbit/s, което е значително по-малко от скоростите, използвани в LAN.

- **Gateway** (на български се използва термина „шлюз") - активни мрежови устройства, наречени още рутери (router), които представляват компютри или специализирани вградени системи, конфигурирани да управляват маршрута на данните. За да се осъществи връзка от една LAN в друга LAN трябва да се премине през gateway.



Фиг. 8.1

- **IP адрес** – уникално 32-битово цяло число, присвоено на дадено устройство в Интернет. Такъв адрес е необходим при предаването на информация между две и повече устройства. Изобразява се в десетичен вид, като всеки байт е разделен с точки (например 192.168.1.10, 10.1.1.23, 81.150.230.33 и други).

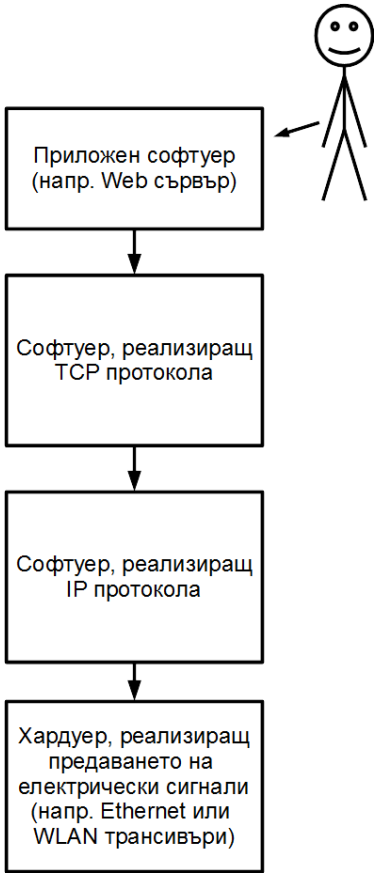
С 32-битово число може да се адресират до 2^{32} = 4 294 967 296 устройства. Поради широката употреба на световната мрежа Интернет този брой се оказва недостатъчен затова съществуват и 128-битови адреси.

- **IP протокол** (Internet Protocol) – стандартизиран механизъм за предаване на информация в световната мрежа Интернет. Той дефинира комуникация, която се осъществява чрез разделяне на данните на пакети в подателя и събирането им в получателя. Всеки пакет освен полезните данни съдържа и IP адреса на подателя/получателя, заедно с различни служебни данни. При този протокол доставката на информация не се гарантира.

- **DHCP протокол** (Dynamic Host Configuration Protocol) – аналогичен на IP протокола, но разликата е, че при първоначалното

включване на устройството параметрите на IP протокола (IP адрес, Mask, Gateway и др.) се получават автоматично от DHCP сървър. Това означава, че потребителят ще получи автоматично, без негова намеса, всички необходими настройки, за да се свърже към Интернет. Характерното за този протокол е, че при всяко стартиране на компютъра си, потребителят получава IP адрес, който може да не съвпада с IP адреса от предишна сесия (предишното включване) на компютъра. Нещо повече – този IP адрес може да се промени по време на настоящата сесия.

- **TCP протокол** (Transmission Control Protocol) – протоколът осигурява пренасянето на непрекъсната поредица от данни, като за разлика от повечето Интернет протоколи, работи с изграждане на връзка и гарантиране на доставката на данните. Софтуерът, реализиращ този протокол е на най-ниското ниво на абстракция спрямо другите протоколи (**фиг. 8.2**).



Фиг. 8.2

8.3. Връзка клиент-сървър. Връзката между две устройства в Интернет се основава на принципа клиент-сървър, при която:

- **клиент** се нарича компютърна програма, изискваща създаването на връзка и използване на някакъв компютърен ресурс;
- **сървърно приложение** е компютърна програма, която предоставя даден ресурс на една или повече клиентски програми. Сървърното приложение „слуша“ за идващи заявки от клиенти, обслужва ги и продължава да „слуша“ за следващи клиенти.

Типичен пример за комуникация клиент-сървър е качването на файл в Google Drive. Тук Интернет браузърът (Mozilla, Opera, Google Chrome) се нарича клиент, а Google Drive – сървърно приложение. Предоставеният ресурс е дисково пространство.

За да може клиент да се свърже към сървър е необходимо да се знаят поне два параметъра на сървъра:

- IP адрес;
- Порт, на който сървърното приложение очаква заявки от клиентите.

Под порт се разбира софтуерен механизъм за връзка, а не физически порт (куплунг). Софтуерните портове са необходими, за да се реализира достъп до сървъра от много потребители към много сървърни програми. Благодарение на портовете един и същ сървър може да се използва например за уеб-страница и дисково пространство за документи едновременно. По подразбиране уеб-страниците се обслужват на порт 80, т.е. когато напишем в браузъра www.mysite.com, всъщност ще се преведе като www.mysite.com:80. С двуеточие се указва номера на порта. Друг пример е програма, осигуряваща достъп до файловата система на сървър – SSH. Връзката на SSH се осъществява на порт 22. Валидни числа за портове са $0 \div 65535$.

От гледна точка на програмиста, връзката клиент-сървър се осъществява посредством т.нар. сокети (socket), които наподобяват файлове. Сокетът е крайна точка за комуникация (endpoint) в една компютърна мрежа. За да започне комуникацията, потребителската програма трябва да отвори сокет, след което да извършва всички операции по комуникацията с помощта на указател към този сокет (точно както се използват файлови указатели).

В настоящото упражнение се използва библиотека на Texas Instruments, която предлага множество API функции за работа със сокети. Най-често използваните от тях са показани в **Таблица 8.1**.

Име на функцията	Кратко описание
socket( )	Създава крайна точка за комуникация, сокет, и връща указател към нея.
bind( )	Присвоява на сокета статичния IP адрес на мрежовата карта. Ако тя използва DHCP, като адрес на функцията трябва да се подаде нула. Присвоява на сокета порт, на който ще се осъществяват връзките.
listen( )	Използва се от сървърни програми. Отваря се присвоеният порт на присвоения IP адрес. От този момент нататък системата очаква („слуша“) за пристигащи връзки от клиенти.
accept( )	Приема връзка с клиент. Тази функция връща указател към сокета на клиента (с негова помощ може да разберем например IP адреса на клиента).
connect( )	Използва се от клиентски програми. На тази функция се предава IP адреса на сървъра и неговия порт, към който ще се свързваме.
recv( )	Връща приетите байтове от сървъра/клиента.
send( )	Изпраща определен брой байтове към сървъра/клиента.

Таблица 8.1

8.4. Развойна среда Code Composer Studio. Фирмата Texas Instruments разработва своя развойна среда за микроконтролерите си, въпреки че много от тях могат да се програмират със софтуер с отворен код (от компилатори до програми за зареждане във флаш паметта). При инсталацията на средата потребителят може да избере комерсиалните компилатори на TI (със същото име) и/или отворени такива (GCC). След това при създаването на нов проект може да се избере един от двата.

CCS е базирана на отворената развойна среда Eclipse. Визуално двете среди си приличат много. Вляво на екрана е разположено дървото на проекта, централно - текстов редактор, долу – терминал на компилатора. При дебъгване на програма допълнително се отварят прозорци за дисасембли на програмата и организацията на стека. Преминаването от редактиращ в дебъгващ изглед става с два бутона горе, вдясно на екрана.

8.5. Задачи за изпълнение.

8.5.1. Да се разучи създаването на TCP/IP връзка. За целта се стартират два прозореца на програмата Sock, намираща се в MSHT_LAB/tools. В първия прозорец се избира:

- Socket type → Server
- В полето Server Name → 127.0.0.1
- В полето Server Port → 80 (пример, може и друго число)
- Натиска се бутон Listen.

Във втория прозорец се избира:

- Socket type → Client
- В полето Server Name → 127.0.0.1
- В полето Server Port → 80 (пример, може и друго число)
- Натиска се бутон Connect
- В полето Message се въвежда произволно текстово съобщение
- Натиска се бутон Send.

Ако всичко е минало наред, в прозореца на сървъра трябва да се покаже същото съобщение.

Пояснение: с този експеримент тествахме TCP/IP комуникацията по „виртуалната LAN карта“ на персоналния компютър. Linux и Windows операционните системи поддържат такъв виртуален интерфейс, чийто виртуален IP адрес е 127.0.0.1

Допълнителен експеримент: научете IP адреса на вашия компютър, като въведете в терминал (за Windows 7 → Start menu → Search → cmd) командата ipconfig. Попитайте ваши колеги за тяхното IP и опитайте да изпратите текстово съобщение до техния компютър.

8.5.2. Да се тества работата на MSP430FR5739 и CC3000 с помощта на демо програмата CC3000_FRAM_Sensor_Application. За целта се стартира средата Code Composer Studio на Texas Instruments. От главното меню се избира File → Switch workspace → Other → Browse → указва се пътя до работното поле (workspace) CC3000_workspace → OK → OK. Вляво на средата, в дървото с директории, се намира директорията на проекта SensorApplication. Щраква се два пъти върху нея, за да се покажат C файловете. От тях се щраква два пъти върху demo.c, където се намира main( ) функцията на проекта. Избира се Project → Build Project. Компилацията на проекта трябва да премине без грешки. След това се зарежда фърмуерът в микроконтролера чрез Run → Debug. В CCS ще се отвори Debug сесия, което е нормално. Изпълнението на програмата ще спре в началото на main( ).

Следва да се отвори терминална програма за RS232 интерфейс (Hyper Terminal, TeraTerm, Real Term, Conect или др.) с 9600 бода, 8-N-1 и COMx, където x е номера на виртуалния COM порт на MSP430FR5739 платката.

Когато се направи това, обратно в CCS се натиска бутон Resume или клавиш F8 от клавиатурата. В рамките на няколко секунди трябва да излезе меню в терминалната програма, изглеждащо по следния начин:

```
-------Press ENTER for terminal prompt--------
Sensor App Version: v1.0.12
Attempting to connect using
SSID: LAB1362
Pass: arm-cortex
```

Фърмуерът на MSP430FR5739 реализира сървър, към който ще се свързва клиентът (компютърната програма) CC3000_FRAM_Sensor_Application. Сървърът използва безжична LAN връзка, затова преди да стартираме клиента трябва да свържем MSP430FR5739 към безжичен рутер. Това става като в терминала въведем командата:

assoc [име на Wi-Fi мрежа] [парола]

където необходимите данни ще са предоставени от ръководителя на упражнението.

!Важно → Wi-Fi мрежата трябва да използва WPA2 (криптиран) протокол.

Щом демо бордът получи динамично IP от рутера, светодиоди 1 ÷ 4 ще светнат, а в терминала ще се изпише Waiting for Clients. Вече може да се стартира клиентското приложение.

Клиента CC3000_FRAM_Sensor_Application се намира в MSHT_LAB/tools/ CC3000_FRAM_Sensor_Application. За да се стартира трябва да се влезе в директорията със същото име и да се щракне два пъти върху batch скрипта:

run_gui.bat

което ще покаже GUI програма, разработена от Texas Instruments за тест на Wi-Fi връзката.

В първия прозорец, който ще се покаже (с име Configure Connection Type) се натиска бутона Manual, който ще отвори нов прозорец (с име Network Interface Selection), в който трябва да изберем LAN картата на компютъра, която го свързва към рутера. Така ще реализираме сценария, показан на **фиг. 5.1**. Натиска се OK.

Ако всичко е минало без грешки, ще се отвори прозорец със слънчева система и планетата Сатурн. Позицията на пръстените на Сатурн всъщност се контролира от акселерометъра на демо платката. Температурата се измерва от NTC също на демо платката. Цветът на Сатурн се изменя в червено или синьо, в зависимост от околната температура. Светодиод 5 на демо борда показва осъществена връзка с клиент, а светодиод 6 (мигащ) показва изпращането на съобщение към клиента (всяко съобщение съдържа X, Y, Z кординати, стойността Vdd на демо платката и околната температура).

8.5.3. Да се реализира изпращане на низ от символи от демо платката с MSP430FR5739 към програмата Sock. За целта да се отвори файлът main.c от проекта sendstr в работното поле, отворено по-рано (CC3000_workspace). Да се разучи С кода в този файл. Под коментара:

//Send string in TCP/IP terminal

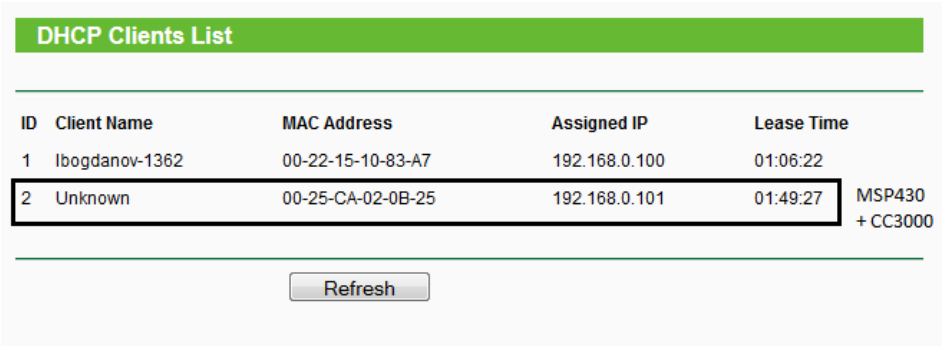
да се извика функцията send( ) с необходимите параметри. Описание на функциите от библиотеката за CC3000 може да бъде намерено в MSHT_LAB/Documents/CC3000_Doxygen_API_swrc266/html/index.html → след което се избира Modules → Socket_api → функция send( ).

Да се компилира и зареди фърмуерът. Ако свързването с рутера е минало успешно в терминала на виртуалния COM порт трябва да се видят съобщенията:

```
Initializing CC3000 WLAN ...done!
Connected!
IP assigned or reassigned!
----INFO----
   Connected to: LAB1362
             IP: 192.168.0.101
           Mask: 255.255.255.0
Default Gateway: 192.168.0.1
            DNS: 192.168.0.1
    DHCP server: 192.168.0.1
CC3000 MAC addr: 0:25:ca:2:b:25
Waiting for a client...
```

Фиг. 8.3

Ръководителят на упражнението може да се логне на рутера и да потвърди успешното свързване на MSP430 демо платката:



След това се стартира програмата Sock в режим client и се въвежда IP-то, което току-що е дадено на платката (на **фиг. 8.3** това е 192.168.0.101). За порт се избира числото, което е записано в променливата port (виж main.c). Натиска се бутона Connect. Въвежда се произволно съобщение в полето Message и се натиска бутон Send. В отговор MSP430FR5739 ще покаже съобщението във виртуалния COM порт и ще изпрати обратно ново съобщение с функцията send( ), което трябва да пристигне в полето Received на Sock.

8.5.4. Да се реализира уеб сървър, съдържащ елементарна уеб страница, която изобразява текущите стойности на трите оси на акселерометъра. Проектът се казва websrv. Да се разучи файлът html_axl.h. Забележете как в код на C се вгражда HTML – чрез низ от символи (в случая char *index_axl_html). За да е по-прегледен кодът, низът е разположен на няколко реда в хедърния файл. Към HTML кода са

добавени символи за нов ред \n и преди кавичките е сложен символът \ (което е изискване на C-низовете). Тоест HTML кодът трябва да претърпи известна трансформация, преди да бъде зареден в контролер, на който няма файлова система. В практиката се използват програми, които генерират хедърен (.h) C файл от HTML (.html) файл.

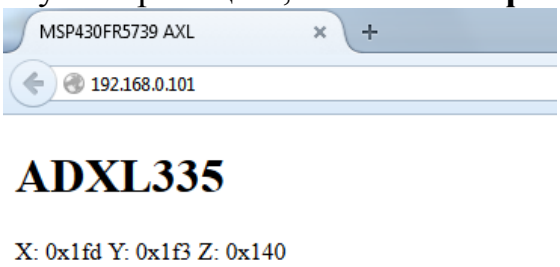
Да се изпрати информацията, сочена от указателя *index_axl_html, подобно на предходното упражнение. Разликата е, че този път изпращаните данни са много и трябва да се разделят на порции, т.е. не може да се изпратят по следния начин:

send(client_socket, index_axl_html, **strlen(**index_axl_html**)**, 0**);** //НЕ!

а трябва да се извика функцията send( ) за всяка една порция. Да се разучи програмата и да се помести функцията send( ) на правилното място, с правилните аргументи.

Помощ: мястото на функцията се намира под коментара //Send partial message over TCP/IP.

Да се компилира и зареди програмата. Да се отвори уеб-браузър и в полето за име на сайт да се напише IP-то на демо платката. Натиснете Enter. Трябва да се появи уеб-страницата, показана на **фиг. 8.4**.



Фиг. 8.4

Пояснение: HTML кодът на програмата е следния:

```
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML>
  <HEAD>
    <TITLE>
      MSP430FR5739 AXL
    </TITLE>
  </HEAD>
<BODY>
  <meta http-equiv="refresh" content="2" />
  <H1>ADXL335</H1>
  <P>X: 0x000 Y: 0x000 Z: 0x000</P>
</BODY>
</HTML>
```

Приложение

П1. ASCII Таблица

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡			
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±			
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥			
131	â	147	ô	163	ú	179	|	195	├	211	╙	227	π	243	≤			
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠			
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡			
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷			
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈			
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°			
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙			
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·			
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√			
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ			
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²			
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■			
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255				

Source: www.LookupTables.com

П2. Приоритети на операторите в С

Operator	Description	Associativity
( ) [ ] . -> ++ --	Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement (see Note 2)	left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (change *type*) Dereference Address Determine size in bytes	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
?:	Ternary conditional	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right

				Status Bits			
				V	N	Z	C
*	ADC(.B)	dst	dst + C → dst	x	x	x	x
	ADD(.B)	src,dst	src + dst → dst	x	x	x	x
	ADDC(.B)	src,dst	src + dst + C → dst	x	x	x	x
	AND(.B)	src,dst	src .and. dst → dst	0	x	x	x
	BIC(.B)	src,dst	.not.src .and. dst → dst	-	-	-	-
	BIS(.B)	src,dst	src .or. dst → dst	-	-	-	-
	BIT(.B)	src,dst	src .and. dst	0	x	x	x
*	BR	dst	Branch to	-	-	-	-
	CALL	dst	PC+2 → stack, dst → PC	-	-	-	-
*	CLR(.B)	dst	Clear destination	-	-	-	-
*	CLRC		Clear carry bit	-	-	-	0
*	CLRN		Clear negative bit	-	0	-	-
*	CLRZ		Clear zero bit	-	-	0	-
	CMP(.B)	src,dst	dst - src	x	x	x	x
*	DADC(.B)	dst	dst + C → dst (decimal)	x	x	x	x
	DADD(.B)	src,dst	src + dst + C → dst (decimal)	x	x	x	x
*	DEC(.B)	dst	dst - 1 → dst	x	x	x	x
*	DECD(.B)	dst	dst - 2 → dst	x	x	x	x
*	DINT		Disable interrupt	-	-	-	-
*	EINT		Enable interrupt	-	-	-	-
*	INC(.B)	dst	Increment destination, dst +1 → dst	x	x	x	x
*	INCD(.B)	dst	Double-Increment destination, dst+2→dst	x	x	x	x
*	INV(.B)	dst	Invert destination	x	x	x	x
	JC/JHS	Label	Jump to Label if Carry-bit is set	-	-	-	-
	JEQ/JZ	Label	Jump to Label if Zero-bit is set	-	-	-	-
	JGE	Label	Jump to Label if (N .XOR. V) = 0	-	-	-	-
	JL	Label	Jump to Label if (N .XOR. V) = 1	-	-	-	-
	JMP	Label	Jump to Label unconditionally	-	-	-	-
	JN	Label	Jump to Label if Negative-bit is set	-	-	-	-

Legend:
0 Status bit always cleared
1 Status bit always set
x Status bit cleared or set on results
- Status bit not affected
* Emulated Instructions

				Status Bits			
				V	N	Z	C
	JNC/JLO	Label	Jump to Label if Carry-bit is reset	-	-	-	-
	JNE/JNZ	Label	Jump to Label if Zero-bit is reset	-	-	-	-
	MOV(.B)	src,dst	src → dst	-	-	-	-
*	NOP		No operation	-	-	-	-
*	POP(.B)	dst	Item from stack, SP+2 → SP	-	-	-	-
	PUSH(.B)	src	SP - 2 → SP, src → @SP	-	-	-	-
	RETI		Return from interrupt TOS → SR, SP + 2 → SP TOS → PC, SP + 2 → SZP	x	x	x	x
*	RET		Return from subroutine TOS → PC, SP + 2 → SP	-	-	-	-
*	RLA(.B)	dst	Rotate left arithmetically	x	x	x	x
*	RLC(.B)	dst	Rotate left through carry	x	x	x	x
	RRA(.B)	dst	MSB → MSB ....LSB → C	0	x	x	x
	RRC(.B)	dst	C → MSB .....LSB → C	x	x	x	x
*	SBC(.B)	dst	Subtract carry from destination	x	x	x	x
*	SETC		Set carry bit	-	-	-	1
*	SETN		Set negative bit	-	1	-	-
*	SETZ		Set zero bit	-	-	1	-
	SUB(.B)	src,dst	dst + .not.src + 1 → dst	x	x	x	x
	SUBC(.B)	src,dst	dst + .not.src + C → dst	x	x	x	x
	SWPB	dst	swap bytes	-	-	-	-
	SXT	dst	Bit7 → Bit8 ..... Bit15	0	x	x	x
*	TST(.B)	dst	Test destination	x	x	x	x
	XOR(.B)	src,dst	src .xor. dst → dst	x	x	x	x

Legend:

0	The Status Bit is cleared	1	The Status Bit is set
x	The Status Bit is affected	-	The Status Bit is not affected
*	Emulated Instructions		

П4. Intrinsic функции на MSP430

MSP430 Intrinsics

Table 6-5 lists all of the intrinsic operators in the MSP430 C/C++ compiler. A function-like prototype is presented for each intrinsic that shows the expected type for each parameter. If the argument type does not match the parameter, type conversions are performed on the argument.

For more information on the resulting assembly language mnemonics, see the *MSP430x1xx Family User's Guide*, the *MSP430x3xx Family User's Guide*, and the *MSP430x4xx Family User's Guide*.

Table 6-5. MSP430 Intrinsics

Intrinsic		Generated Assembly
unsigned short	__bcd_add_short(unsigned short *op1*, unsigned short *op2*);	MOV *op1*, *dst* CLRC DADD *op2*, *dst*

unsigned long	__bcd_add_long(unsigned long *op1*, unsigned long *op2*);	MOV *op1_low*, *dst_low*
		MOV *op1_hi*, *dst_hi*
		CLRC
		DADD *op2_low*, *dst_low*
		DADD *op2_hi*, *dst_hi*

unsigned short	__bic_SR_register(unsigned short *mask*);	BIC *mask*, SR
unsigned short	__bic_SR_register_on_exit(unsigned short *mask*);	BIC *mask*, saved_SR
unsigned short	__bis_SR_register(unsigned short *mask*);	BIS *mask*, SR
unsigned short	__bis_SR_register_on_exit(unsigned short *mask*);	BIS *mask*, saved_SR
unsigned long	__data16_read_addr(unsigned short *addr*);	MOV.W addr, Rx MOVA 0(Rx), dst
void	__data16_write_addr (unsigned short *addr*, unsigned long *src*);	MOV.W addr, Rx MOVA src, 0(Rx)
unsigned char	__data20_read_char(unsigned long *addr*);(1)	MOVA *addr*, Rx MOVX.B 0(Rx), *dst*
unsigned long	__data20_read_long(unsigned long *addr*);(1)	MOVA *addr*, Rx MOVX.W 0(Rx), *dst.lo* MOVX.W 2(Rx), *dst.hi*
unsigned short	__data20_read_short(unsigned long *addr*);(1)	MOVA *addr*, Rx MOVX.W 0(Rx), *dst*
void	__data20_write_char(unsigned long *addr*, unsigned char *src*);(1)	MOVA *addr*, Rx MOVX.B *src*, 0(Rx)
void	__data20_write_long(unsigned long *addr*, unsigned long *src*);(1)	MOVA *addr*, Rx MOVX.W *src.lo*, 0(Rx) MOVX.W *src.hi*, 2(Rx)
void	__data20_write_short(unsigned long *addr*, unsigned short *src*);(1)	MOVA *addr*, Rx MOVX.W *src*, 0(Rx)
void	__delay_cycles(unsigned long);	See Section 6.8.2.

void	__disable_interrupt(void);
	OR
	_disable_interrupts(void);
		DINT

void	__enable_interrupt(void);
	OR
	_enable_interrupt(void);
	OR
	_enable_interrupts(void);
		EINT

unsigned int	__even_in_range(unsigned int, unsigned int);	See Section 5.8.13.
unsigned short	__get_interrupt_state(void);	MOV SR, *dst*
unsigned short	__get_R4_register(void);	MOV.W R4, dst
unsigned short	__get_R5_register(void);	MOV.W R5, dst
unsigned short	__get_SP_register(void);	MOV SP, *dst*
unsigned short	__get_SR_register(void);	MOV SR, *dst*
unsigned short	__get_SR_register_on_exit(void);	MOV saved_SR, *dst*
void	__low_power_mode_0(void);	BIS.W #0x18, SR
void	__low_power_mode_1(void);	BIS.W #0x58, SR
void	__low_power_mode_2(void);	BIS.W #0x98, SR
void	__low_power_mode_3(void);	BIS.W #0xD8, SR
void	__low_power_mode_4(void);	BIS.W #0xF8, SR
void	__low_power_mode_off_on_exit(void);	BIC.W #0xF0, saved_SR
void	__never_executed(void);	See Section 6.8.3.
void	__no_operation(void);	NOP
void	__op_code(unsigned short);	Encodes whatever instruction corresponds to the argument.
void	__set_interrupt_state(unsigned short *src*);	MOV *src*, SR
void	__set_R4_register(unsigned short *src*);	MOV.W *src*, R4
void	__set_R5_register(unsigned short *src*);	MOV.W *src*, R5
void	__set_SP_register(unsigned short *src*);	MOV *src*, SP
unsigned short	__swap_bytes(unsigned short *src*);	MOV *src*, *dst* SWPB *dst*