Марин Христов, Росен Радонов,
Благомир Дончев

Системи за проектиране в микроелектрониката

София, 2004

Учебникът по "Системи за проектиране в микроелектрониката" е предназначен за студентите от специалността "Електроника" на Технически университет – София при подготовката им по дисциплините "Системи за проектиране в микроелектрониката" – за образователно-квалификационна степен "магистър", "Автоматизация на инженерния труд в микроелектрониката" и "Проектиране на интегрални цифрови схеми и системи" за образователно-квалификационна степен "бакалавър". Той може да бъде използван и от студентите от други сходни специалности, докторанти, инженери, проектанти и др., интересуващи се от работата и приложението на индустриални CAD системи.

У В О Д

Проектирането на едно микроелектронно изделие е сложен творчески процес. Успехът на проекта се определя главно от квалификацията и способностите на проектанта, независимо колко “интелигентна” и усъвършенствана е използваната компютърна система за автоматизирано проектиране. От друга страна, не е възможно “ръчното” проектиране на интегрална схема или система, съдържаща няколко стотин милиона елемента, интегрирани в един чип.

Трудно се поддава на класификация и описание огромното разнообразие от системи за проектиране. Освен предлаганите комерсиализирани пакети, обикновено всяка значима компания за проектиране и производство на микроелектронни изделия използва собствено разработени софтуерни продукти. Въпреки това, през последните години в света като индустриален стандарт се наложиха няколко големи интегрирани среди – CADENCE, SYNOPSYS, MENTOR GRAPHICS.

Учебникът по “Системи за проектиране в микроелектрониката” е опит за първи път в България да се опише структурата и функционалните възможности на някои от най-разпространените системи за проектиране. Представеният материал е много синтезиран, кратък и в някои случаи схематичен, тъй като разглежданите софтуерни среди за проектиране са изключително сложни, постоянно се развиват и усъвършенстват, така че конкретните методи, алгоритми и начин на работа са вече “остарели” в момента на описването им. Надяваме се, че описаните принципи и подходи при проектирането на микроелектронни изделия ще са валидни още известно време.

Материалът в този учебник може да бъде възприет по-лесно, ако при разглеждането му се работи практически с описвания програмен пакет.

Настоящата книга обобщава десетгодишния опит на колектива на ECAD лабораторията при Факултета по Електронна техника и технологии в Техническия университет – София. Изразяваме нашата голяма благодарност и признателност за приноса и помощта на многобройните сътрудници на лабораторията, голяма част от които в момента работят в други фирми и организации в България и чужбина.

Ще приемем с внимание и интерес всички мнения, препоръки, предложения, целящи подобряването и осъвременяването на настоящия учебник.

София, 2004г. *АВТОРИТЕ*

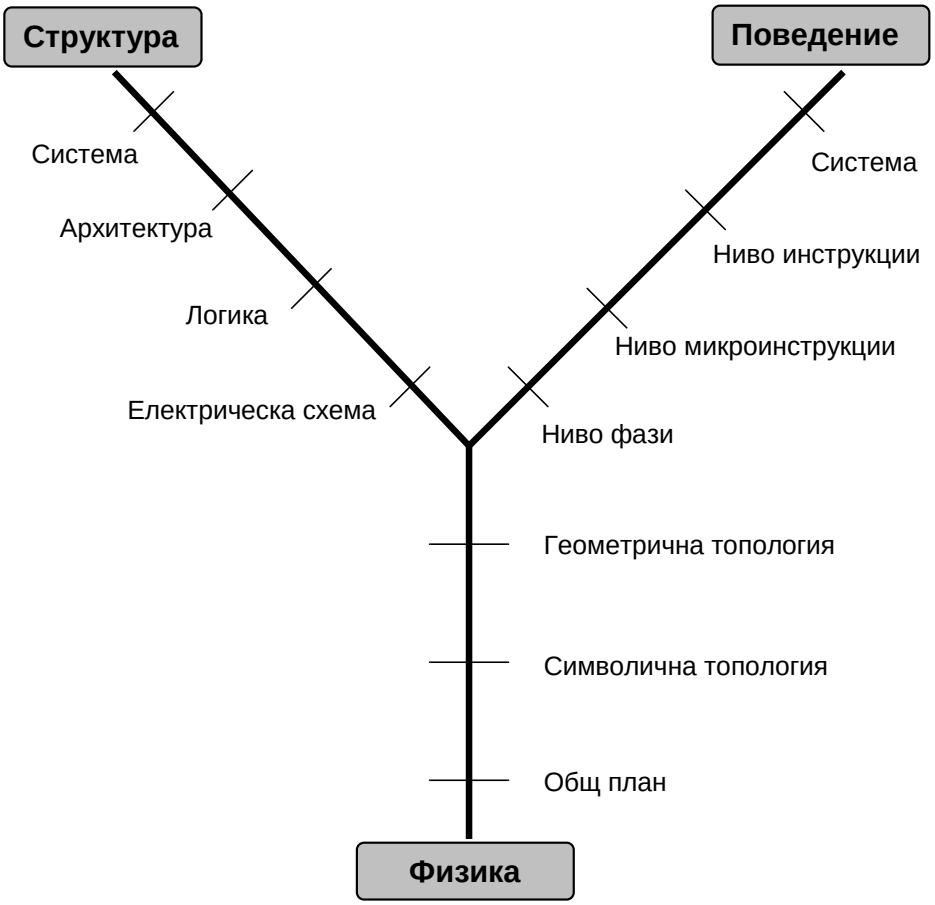
ГЛАВА 1

Методи за автоматизирано проектиране в електрониката

1.1. Начини за представяне на интегрални схеми

Всяка интегрална схема може да бъде представена чрез три различни описания - нива на абстракция: поведенческо, структурно и физическо. Поведенческото ниво позволява да се опише функционирането на схемата. Необходимата архитектура, осигуряваща желаните свойства се представя от структурното описание. Физическото описание показва материалната реализация на схемата на ниво топология. Всяко ниво на абстракция може да се представи с различни по вид описания от по-долното ниво. Така например различни структурни схеми могат да съответстват на един и същ поведенчески модел.

Както е показано на фиг. 1.1 трите основни нива на абстракция могат да бъдат представени графично като клоните на Y. Според това представяне проектирането на схема по методологията "от горе на долу" ("top-down") предлага изборът на път от възможно най-високото ниво на поведенческо описание до най-ниското физическо ниво, преминавайки през структурните нива. Този подход е валиден не само при проектирането на интегрални схеми, но и при реализирането на



Фиг. 1.1. Диаграма на Гайски

програмни системи. Например при компилаторът, който преобразува програмен език от високо ниво до асемблер също трябва да се преминава от едно ниво на абстракция към друго.

Не трябва да се смесват нивата на абстракция с йерархичното описание на схемата. Йерархията е въведена, за да се избегне повтарящото се описание на голям брой елементи от едно и също ниво на абстракция. По-лесно е да се анализира една схема или топология, ако различните инвертори, тригери и др. са организирани като клетки, клетките като модули и т. н. Важно е да се разбере, че йерархичното описание се използва само за едно и също ниво на абстракция. Необходимо е, освен това, да се запази една и съща йерархична структура за различните нива на абстракция. Така, на една клетка съдържаща инвертори, на логическо ниво съответства същата клетка, съставена от правоъгълници на топологично ниво. Това е т. нар. йерархична система модели (описания).

1.2. Етапи при проектирането на микроелектронни изделия

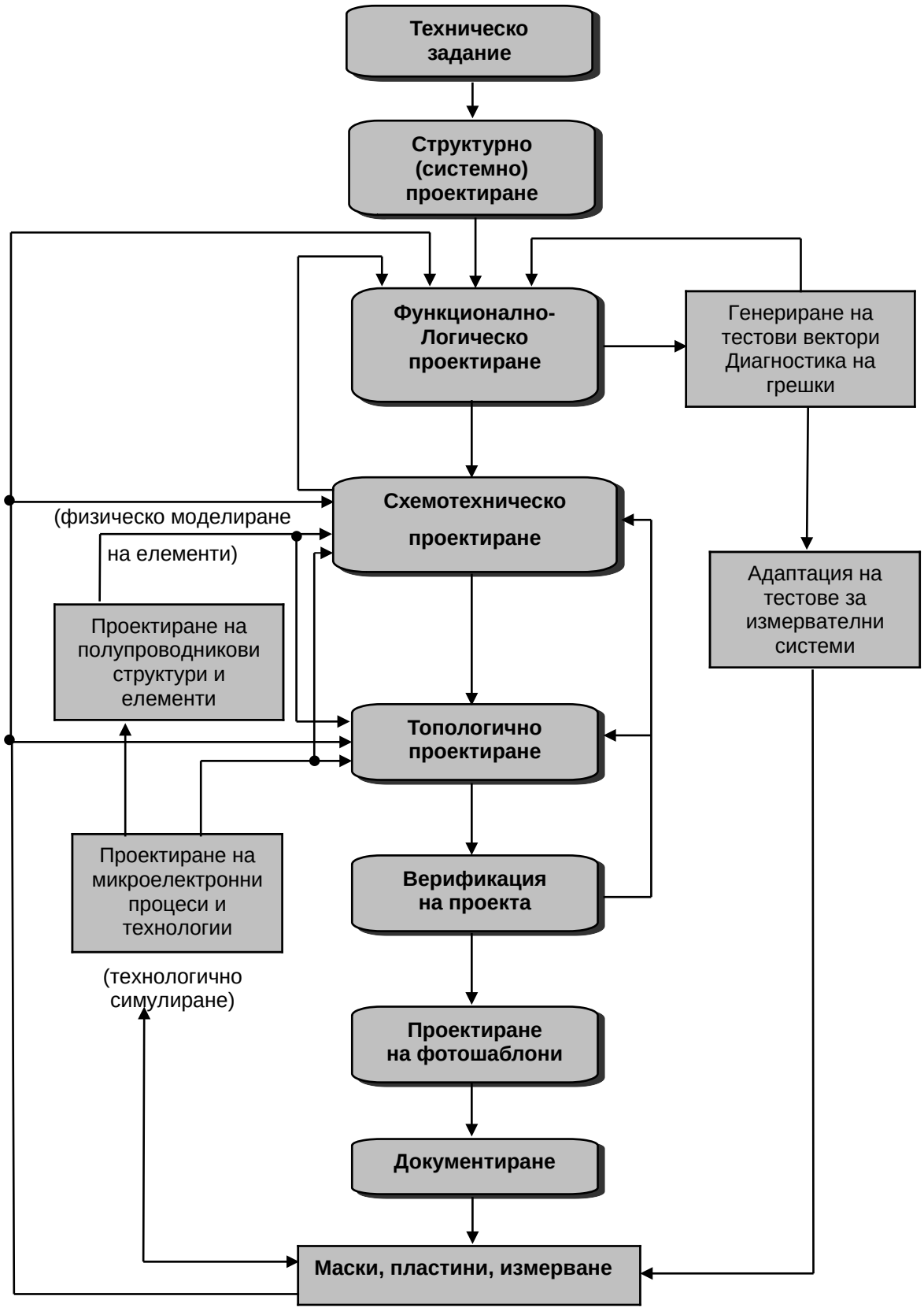
Линейно представяне на основните етапи при проектирането на интегрални схеми е дадено на фиг. 1.2. Не трябва да се забравя, че проектирането е част от цялостния процес проектиране-производство-тестване (фиг. 1.3).

Един от основните етапи, определящи крайния успех на процеса проектиране-производство-реализация е формулирането на *техническо задание.* Тук се специфицират целта и задачите, за които е предназначена проектираната схема (система). Задават се и се обосновават основните системни технико-икономически характеристики и параметри, като технология, бързодействие, захранване, цена и др.

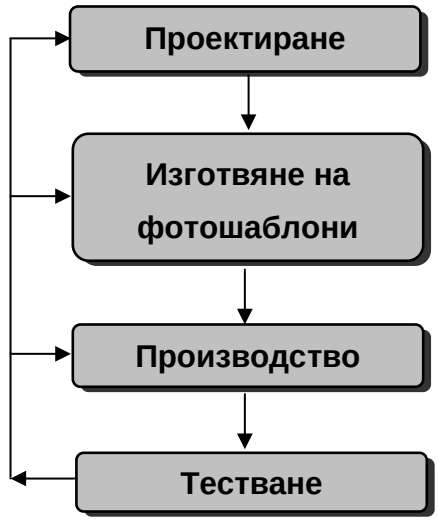
Трябва да се отбележи, че етапът *структурно (системно) проектиране* понякога се дефинира като архитектурно, поведенческо ниво. Основното внимание на проектанта при архитектурното ниво на абстракция е насочено към общата структура на системата, описвана обикновено чрез примитиви от високо ниво, като процесори, памети, входно-изходни блокове и т. н., характеризирани чрез параметри като обем памет, време за достъп, дължина на думите, протоколи за обмен на данни и др. На този етап на проектиране се изследва (синтезира) общата структура на системата, разпределението на потоците информация между отделните блокове, определя се подходящата система команди и сигнали, създават се алгоритмите за функциониране на подсистемите, общата система се разделя на апаратна и програмна част.

Функционално-логическата схема на проектираната система се построява на етапа на *функционално-логическото проектиране* като се отчита библиотеката базови фрагменти и елементи, които са разработени за избраната технология. Правилното функциониране на

синтезираната логическа схема се проверява чрез симулиране на логическото поведение на схемата и анализ на временните съотношения.



Фиг. 1.2. Основни етапи при проектирането на ИС



Фиг. 1.3. Проектиране-производство-тестване

При констатирани проблеми (грешки) при логиката на функциониране или временните закъснения на отделните сигнали се извършва редакция и оптимизация на първоначалната схема до получаването на задоволително решение. По такъв начин се извършва оптимизация на вътрешната структура на проектираната схема (система). На този етап се генерират *контролни и диагностични тестове* за проверка на прототипа. Тестовете са предназначени да откриват неизправности, въведени по време на производството на интегралната схема, при предположение, че проектирането е без грешки. За да се осъществят на практика измерванията, съставените тестове се адаптират за конкретна измервателна тестерна система.

На етапа на *схемотехническото проектиране* се синтезират и симулират електрическите схеми на отделните възли и блокове на проектираната схема. В зависимост от конкретните изисквания се извършват един или няколко от следните видове анализи: постояннотоков, честотен, преходен, температурен, радиационен, шумов, анализ на чувствителност, малосигнален, голямосигнален, смесен цифрово-аналогов и др. За постигане на задоволителни резултати от проектирането е предвидено използването на модели на електронните компоненти с различни нива на точност и сложност, възможност за оптимизиране по един или няколko параметъра и др.

Физическа реализация на проекта във вид на топология се синтезира на етапа на *топологичното проектиране.* Тук се решават следните задачи: компановка, разполагане на елементите, трасировка на междуелементните съединения. Като резултат от топологичното проектиране се получава проект на комплектът фотошаблони, необходими за изготвяне на интегралната схема.

Много важно за успешното изпълнение на схемотехническото и топологично проектиране, както и цялостното проектиране на микроелектронни елементи и интегрални схеми са етапите *"Проектиране на микроелектронни процеси и технологии* (технологично симулиране)" и *"Проектиране на полупроводникови структури и елементи* (физическо моделирани на елементи)". На базата на детайлно дву-тридименсионално решение на математическите уравнения, описващи технологичните процеси на използваната технология и физическите процеси в полупроводниковите елементи се осъществява многопараметрично

оптимизационно проектиране на базовата микроелектронна технология и основните градивни елементи. По този начин се осъществява интегрална връзка и единство между и физическите процеси в полупроводниковите интегрални структури, технологичните процеси и методи за изготвянето им, топологичното, схемотехничното и функционално-логическото им описание.

Процесът *верификация на проекта* се състои от следните подетапи:

- **Проверка на нормите и правилата за топологично проектиране.** Проверява се формалното изпълнение на всички допуски, норми и правила на използуваната технология, като реализирани минимални размери, минимални разстояния между елементи, елементи и проводящи шини, минимална широчина на проводящи шини, припокриване на области от елементи, елементи и шини, минимални и максимални стойности на периметри и площи на области и т. н.
- **Възстановяване на електрическата схема** от топологичния проект и проверка на индентичността й с първоначалната схема.
- **Ресимулация.** Въз основа на създадената топология се извлича реалната електрическа еквивалентна схема чрез добавяне на паразитните елементи. Извършва се отново схемотехническа или логическа симулация за определяне критичния път за предаване на сигналите, оценка на времената за закъснение и общото функциониране на схемата.

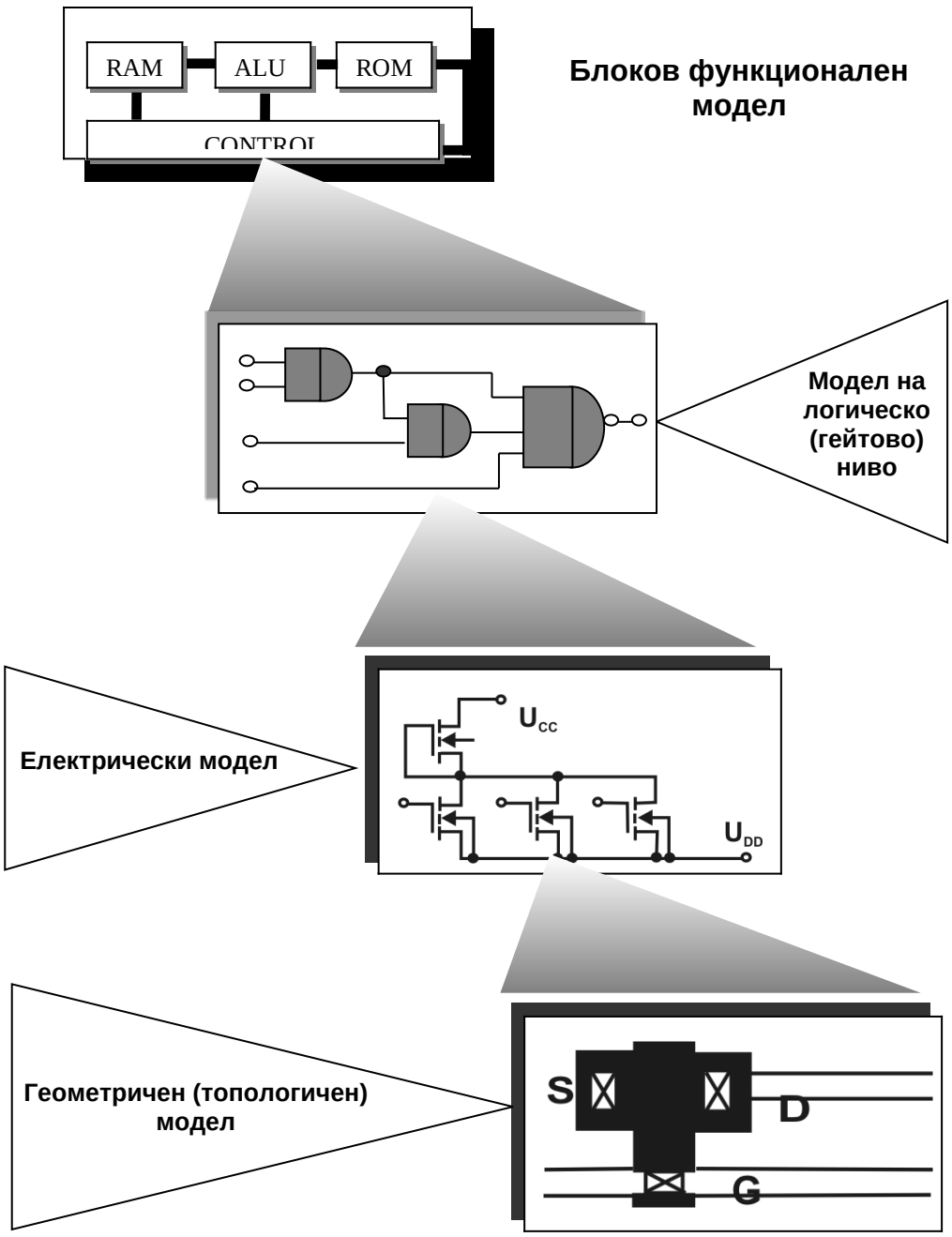
При успешното изпълнение на верификацията на проекта се пристъпва към *проектиране на фотошаблоните* и *изготвяне* на необходимата документация за произвеждане на проектираната схема.

Процесът на проектиране е итерационен процес, при които всеки етап се изпълнява многократно до получаването на задоволително решение. При достигане до проблемна ситуация или грешка процесът на проектиране се връща с един или няколко етапа по-назад и се изпълнява повторно до успешното преодоляване на проблема.

Особеност на този процес е силната взаимна връзка между отделните етапи при проектирането и типа на проектираната схема. Така например електрическият анализ на схемата, проведен без отчитане на паразитните елементи, екстрахирани от топологията, може да се счита само като груба прогноза за действието й. От друга страна, основните геометрични размери на елементите се определят на основата на детайлен анализ и проектиране на топологията за достигане на зададени електрически параметри и характеристики. Следователно задоволително крайно решение може да бъде получено само чрез последователни приближения.

Важна особеност на съвременните системи за автоматизирано проектиране е използването на вградена йерархична система модели,

използвани при различните нива за описание на проекта. На фиг. 1.4 е показана примерна система йерархични модели.



Фиг. 1.4. Системна йерархия на модела

1.3. Основни изисквания към системите за автоматизирано проектиране

Системите за автоматизирано проектиране имат следните видове осигуряване:

- **Методическо осигуряване** - Тук влизат материалите, в които е изложена теорията, методите, способите, математическите модели, алгоритмите, специалните алгоритмични езици за описание на обекти, терминологията, нормативите и другите

данни обезпечаващи методологията на автоматизираното проектиране.

- **Математическо осигуряване** - съвкупност от математически методи, алгоритми и модели, необходими за осъществяване на проектирането.
- **Лингвистично осигуряване** - проблемно ориентирани езици, термини, определения.
- **Програмно осигуряване** - съвкупност от текстове на програми, програми на машинни носители, експлоатационна документация. Програмното осигуряване се разделя на общосистемно (операционни системи, транслатори и т. н.) и приложно (пакети програми с конкретни предназначения).
- **Техническо осигуряване** - устройства на изчислителната и организационната техника, средства за съхраняване и предаване на информация, въвеждане и извеждане на информация, интерактивна връзка човек-компютър и т. н.
- **Информационно осигуряване** - документи с описание на стандартни проектни процедури, типови проектни решения, типови елементи, материали и други данни, файлове и бази данни на машинни носители и др. Компонентите на информационното осигуряване трябва да обезпечат възможност за ефективен достъп до исканата информация, съхраняване, обновяване, редактиране, системи за нива на достъп, защита на данните и т. н.
- **Организационно осигуряване** - съвкупност от документи, правила, заповеди, регламентиращи права, задължения и функции на всеки участник в разработката, поддържането и експлоатацията на система за автоматизирано проектиране.

Към системите за автоматизирано проектиране се предявяват следните основни изисквания:

- **Правилен избор на платформата и комплектоването на използвания хардуер** - Поради трудностите за пълна формализация на процеса на проектиране успехът за реално приложение на интегрирана система за автоматизирано проектиране се определя от правилното "разпределение на функциите" между проектанта и компютъра. Компютърът поема цялата техническа, поддаваща се на формализация и самообучение проектанска дейност. Всички творчески решения и операции, такива като синтез на нови топологични решения, окончателна оценка на резултатите от проектирането трябва да се изпълняват от човека.
- **Интерактивност** - Във връзка с необходимостта от постоянно взаимодействие на проектанта с компютъра едно от основните изисквания се явява оперативност и нагледно изобразяване на междинните и крайни резултати в графична и

цифрови форми, както и удобство за въвеждане и извеждане на голяма по обем информация (обикновено графична).

- **Възможност за поетапно въвеждане на части от системата**- Това увеличава ефективността на системата за автоматизирано проектиране (САПР), тъй като сроковете за разработка на математическото осигуряване и трудностите за това обикновено са големи. Към това изискване се отнася и възможността за нарастване на системата, т. е. въвеждане на нови модули за проектиране, използващи нови математически методи. От тук произтича необходимостта от модулна организация на САПР.
- **Съвместимост на ръчни и автоматизирани методи** за проектиране. В системите за автоматизирано проектиране трябва да бъде предвидена възможността за ръчно изпълнение на всяка операция от цикъла на проектирането. Това повишава универсалността на системата, тъй като винаги се срещат схеми, отделни модули или елементи, за които по едни или други причини не могат да бъдат приложени интегрираните в системата средства.
- **Максимална адаптивност на системата** към промяна на технологията и типа проектирана интегрална схема. Поради постоянното усъвършенстване (промяна) на технологията е необходима бърза адаптация на използваното математическо осигуряване. Това обезпечава продължителното използване на системата за автоматизирано проектиране.

Адаптивността на системата към типа проектирана схема се състои във възможността за промяна на функционалните свойства на САПР в зависимост от стойностите на параметрите на схемата и достигнатото на междинни етапи качество на проектирането:

- **Възможност за едновременно въвеждане и работа по няколко проекта** - Обикновено една система за автоматизирано проектиране се използва от екип проектанти, работещи едновременно по няколко проекта на интегрални схеми. Това налага да се работи паралелно по няколко разнотипни проекта, като всеки един може да е на различен етап от проектирането.
- **Наличие на архив, възможност за лесно въвеждане на корекции**. Тези възможности позволяват не само да се разработва конструкторска документация, но и тя лесно да се съхранява. Наличието на архив позволява да се изготвя документация за по-рано разработвана схема, за нейното усъвършенстване, използване на части от нея и др.
- **Интензивен Контрол на междинни и крайни резултати от проектирането**. Контролните операции след всеки етап от проектирането като правило могат да бъдат формализирани и осъществени от компютъра. Бързото откриване и отстраняване на

грешки при проектирането и оценка на различни варианти решения рязко съкращават времето за проектиране и намалява разходите за редовно производство.

- **Оптимална зависимост на изразходваното машинно време от големината на проектираната интегрална схема**. Универсалност на системата за автоматизирано проектиране по отношение на параметрите на проектираната схема, размери на кристала, брой и тип градивни елементи, технологични ограничения и др.

1.4. Обобщение

Проектирането на едно микроелектронно изделие е сложен неоднозначен творчески процес, изискващ използването на система за автоматизирано проектиране. Успехът на този процес се определя главно от квалификацията и качествата на проектанта, колкото и усъвършенствана и "интелигентна" да е прилаганата САПР. От друга страна, обаче, не е възможно проектирането и производството на съвременна интегрална схема без ефективното използване на подходяща система за автоматизирано проектиране.

ГЛАВА 2

Въведение в CADENCE

CADENCE Design Framework II е общ интерфейс към пълния набор от програмни продукти за проектиране на интегрални схеми в системата CADENCE OPUS. Използването на общ потребителски интерфейс и обща база данни позволява лесно преминаване между различни етапи при проектирането - например визуализация на принципната електрическа схема, резултати от симулации, топология за даден проект и др.

За работата си Design Framework II (DFII) изисква X Windows (CDE/Openwindows върху Sun SPARCstation).

CADENCE е автоматизирана система за т. нар. top-down проектиране на интегрални схеми, т. е. от описание на най-високо ниво до генериране на топологията на интегралната схема, чрез използване на метода за проектиране със стандартни клетки. На фиг. 2.1 и фиг. 2.2 са дадени последователностите на работа при използване Semi-custom и Full-custom подходи на проектиране

2.1. Организация на базата данни в DFII

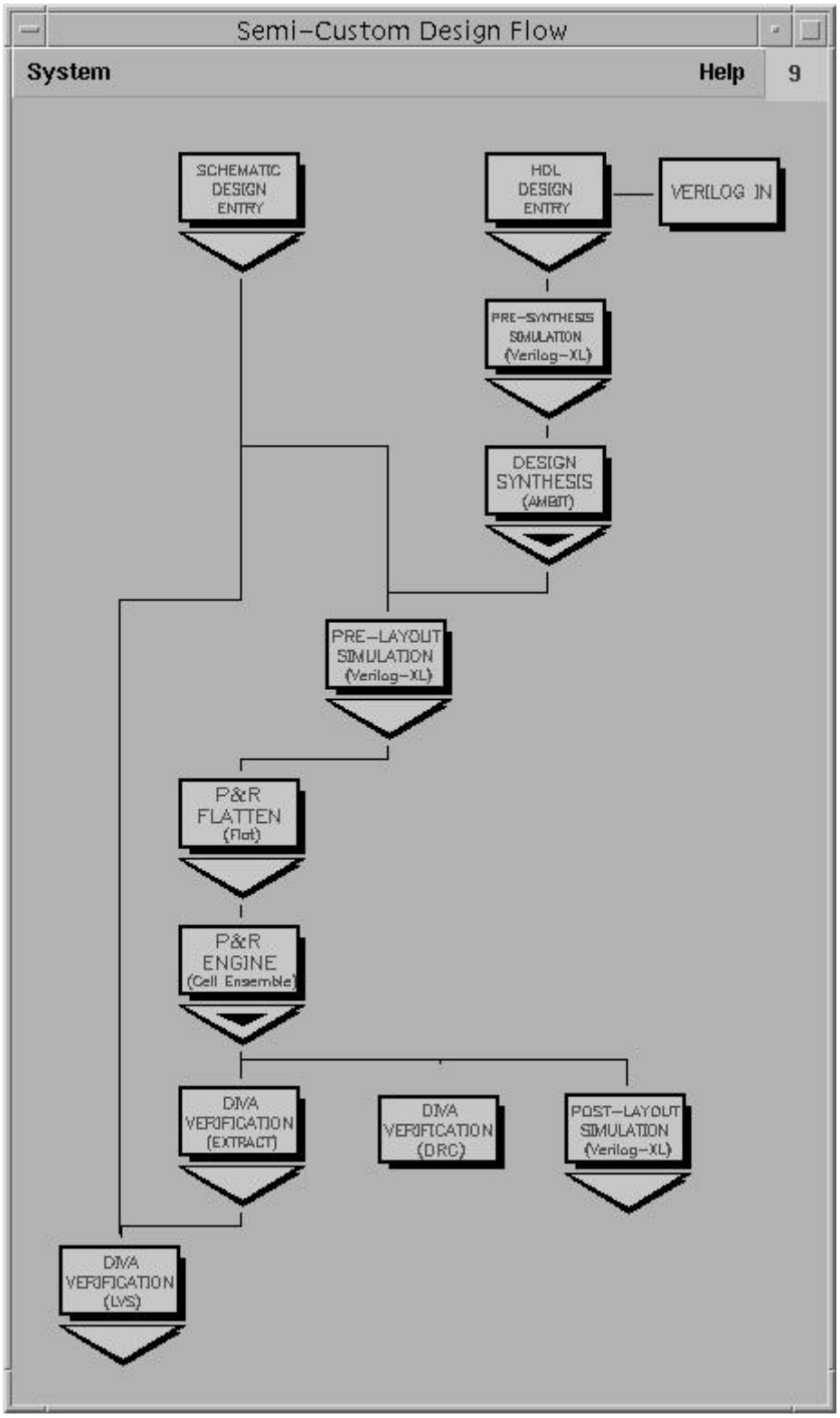
CADENCE има собствена структура на организация и управление на базата данни. Всички данни в DFII са организирани в *библиотеки*. В библиотеките се съдържа цялата информация за

Слоеве (Layers);

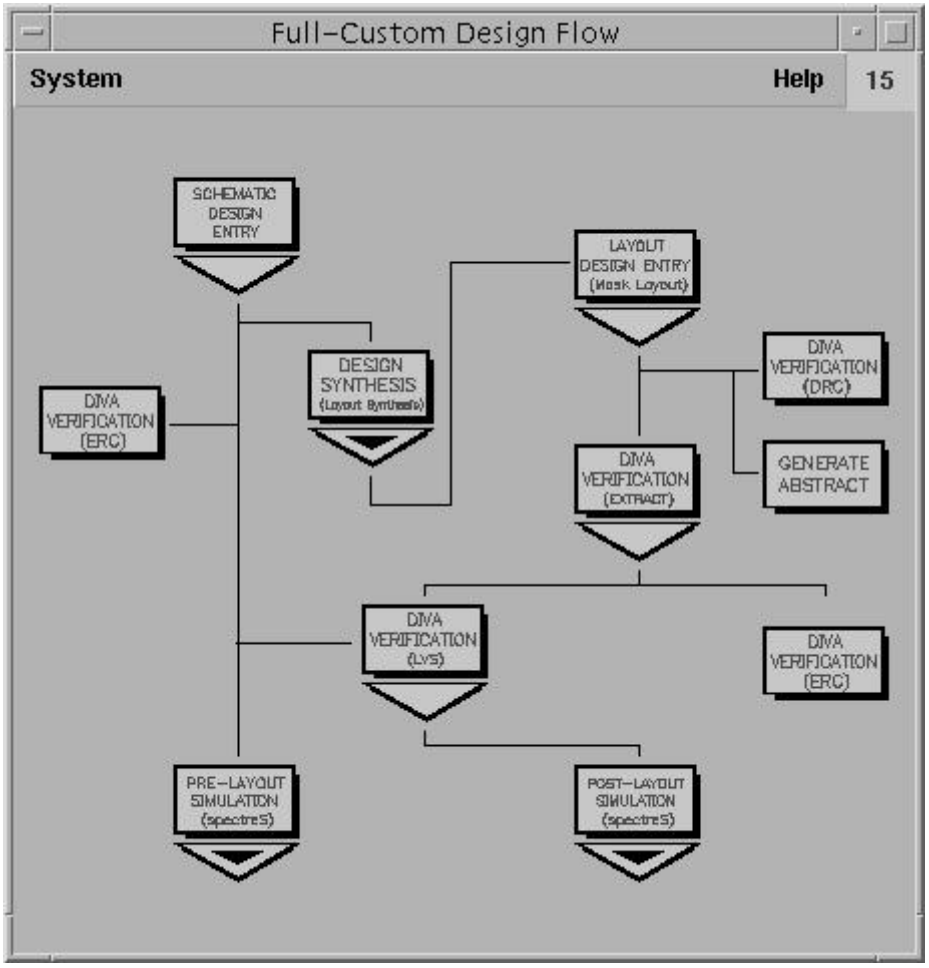
Правила за съответния процес (норми на проектирането; норми за разположение и трасиране; правила при символично представяне);

Клетки (Cells).

Библиотеките могат да се разглеждат като аналог на директориите във файловите системи и да се използват, за да се групират свързани файлове с данни.



Фиг. 2.1. Примерна последователност на работа за Semi-custom подход



Фиг. 2.2. Примерна последователност на работа за Full-custom подход

2.1.1. Технологична библиотека

Информацията, която дефинира технологичния процес се съхранява в технологичен файл, а от своя страна той е част от цяла технологична библиотека. Този текстов файл дефинира използваните технологични слоеве, заедно с техните цветове и начин на визуализация. В технологичната библиотека се съдържат също и файлове с нормите за проектиране и дефинициите на символните елементи за скелетните схеми при топологично проектиране и др..

Винаги, когато се създава нова библиотека, вече съществуваща *технологична библиотека* се закача към нея*,* а *технологичният файл* се компилира и необходимата информация се включва към библиотеката.

2.1.2. Библиотеки в DFII

Библиотеките се използват да съхраняват всички данни при проектирането. Те съдържат цялата необходима информация за използвания технологичен процес, облекчавайки по този начин преместването на данните. Организацията на библиотечно ниво позволява няколко потребителя да работят с едни и същи данни без да възникват конфликти.

2.1.3. Структура на библиотеките.

Библиотеките обхващат:
Cells - Клетки
CellViews - Способи за представяне на клетките

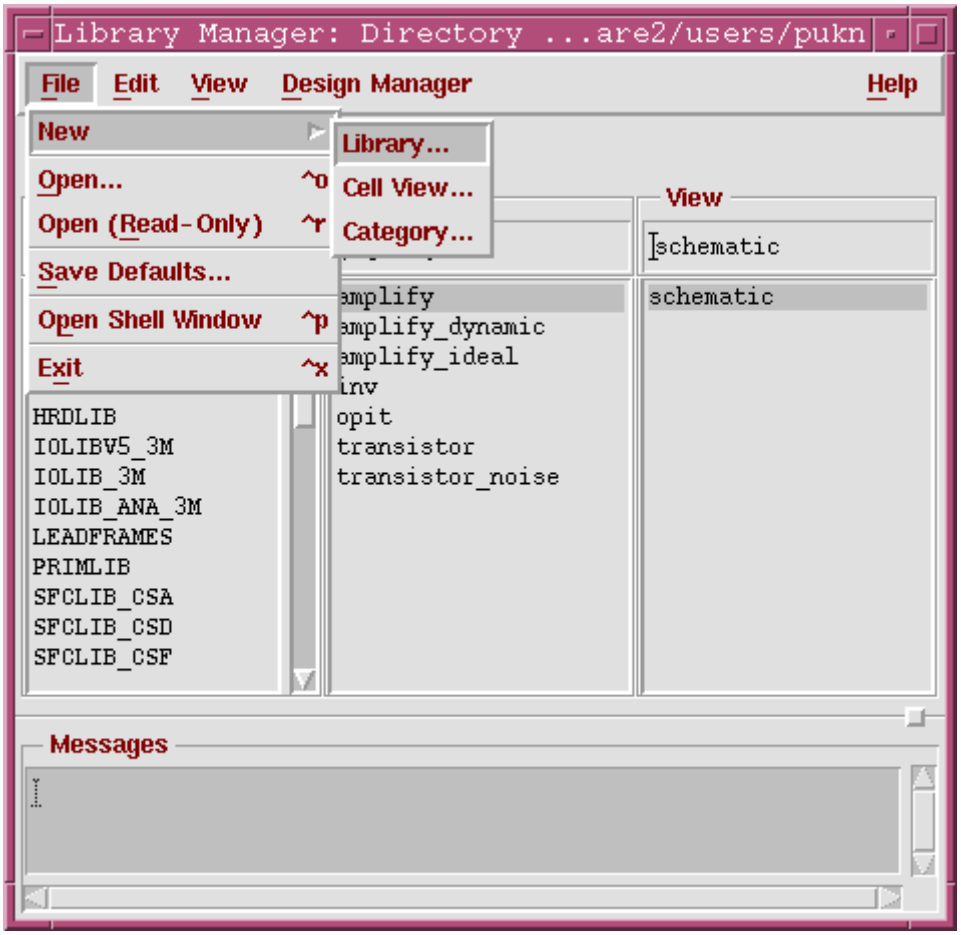
Клетка е най-общото име за проект. В дадена библиотека се съдържат различен брой отделни проекти (клетки). Всеки проект може да се представи по различен начин - т. е. като схема (*schematic*), като символ (*symbol*), като топология (*layout*) - или може да се каже, че проектът (cell) се характеризира с набор от различни *представяния* (cellviews).

Някои от видовете представяния на клетки са следните:

- symbol, schematic - използва се в схемния редактор;
- layout, compacted - за топологичен редактор;
- abstract - за генериране топологията на интегрална схема;
- spice, spectre - за аналогова симулация;
- verilog - за цифрова симулация;
- extracted - за извличане на паразитни елементи от топология;
- analog_extracted – за ресимулация след извличане на паразитните елементи, и др.

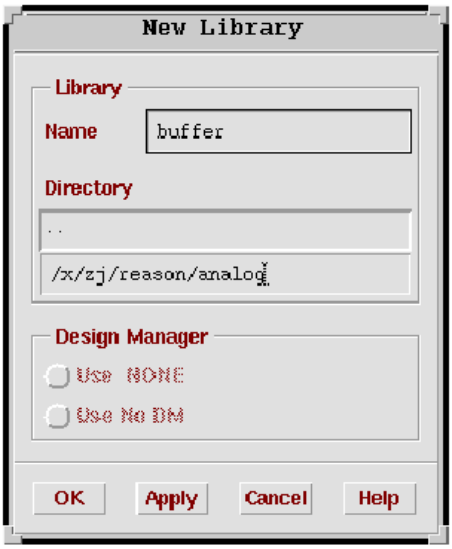
Библиотеките се създават първоначално, като към тях се прикрепва технологичен файл на технологията, която ще се използва. На UNIX ниво се създава директория, чието име е името на библиотеката. Тази директория може да бъде преместване свободно из файловата система,

без това да създаде проблем с нейната структура, но модифицирането на файлове вътре в нея би довело до повреда на нейната структура и тя ще стане неизползваема. Работата с библиотеките се извършва чрез Библиотечния мениджър.

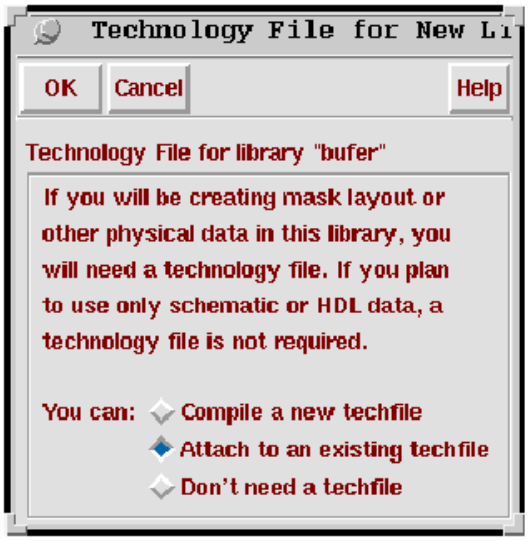


Фиг. 2.3. Библиотечен мениджър.

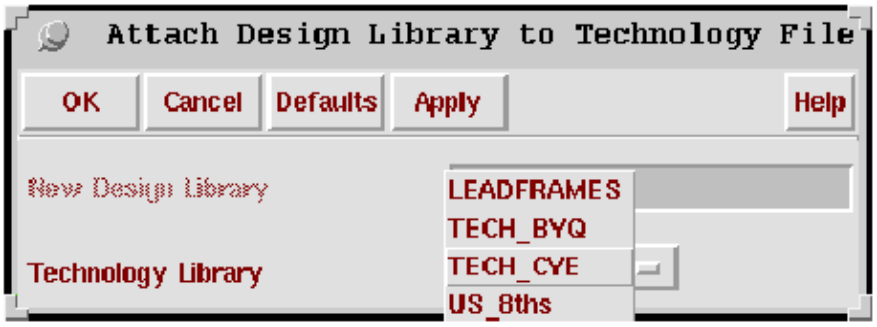
Всеки нов проект започва със създаването на библиотека, в която ще се съхранява, като се изпълняват последователно операциите а), б) и в).



а)



б)



в)

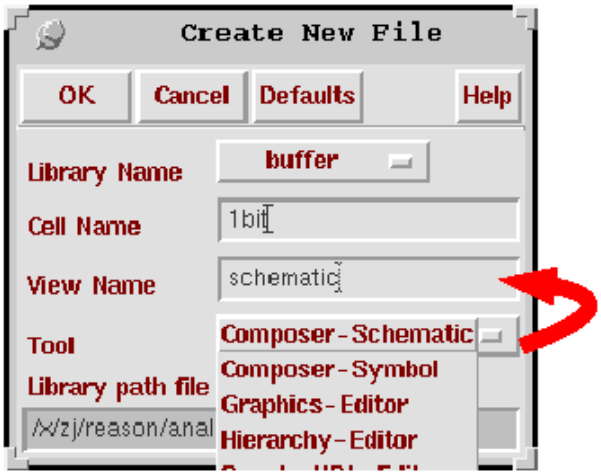
Фиг. 2.4. Стъпки при създаването на нова библиотека.

2.2. Въвеждане на електрическата схема

В CADENCE съществуват различни начини за въвеждане на електрическата схема.

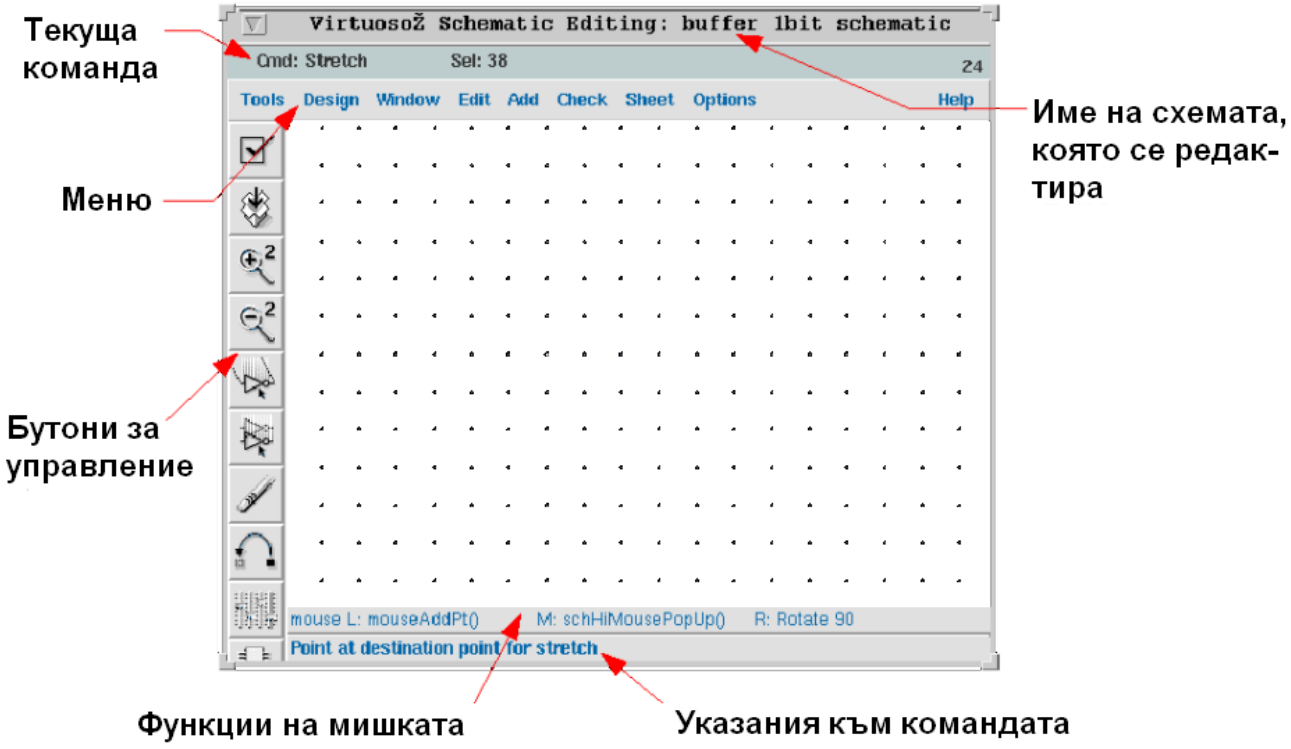
Единият от тях е чрез предварително генериран netlist, който може да бъде изход от друг програмен продукт за автоматизирано проектиране, например **SYNOPSYS**. Форматите, които се поддържат са **EDIF**, **VERILOG**, **CDL**, **TEGAS**, **SILOG** и **SDL**. Важно условие за този вид въвеждане е елементите, изграждащи схемата, да са съобразени с използваните библиотеки в CADENCE.

Другият начин е чрез използване на схемния редактор.



Фиг. 2.5. Създаване на нова клетка

Елементите се извикват чрез *библиотечния браузър* или директно чрез имената им, ако се знаят. Използват се представяне тип symbol. Връзките между елементите се осъществяват чрез прости шини (**narrow wire**) или магистрали (**wide wire**). На всяка шина има възможност да се задава име (**label**), като по този начин е възможно да се осъществи свързването на възли, които се намират далече един от друг в схемата. При свързване тип магистрала наименуването на шините е задължително. Входовете и изходите на схемата се обозначават с т. нар. пинове (**pins**), които се делят на входни, изходни и входно/изходни.



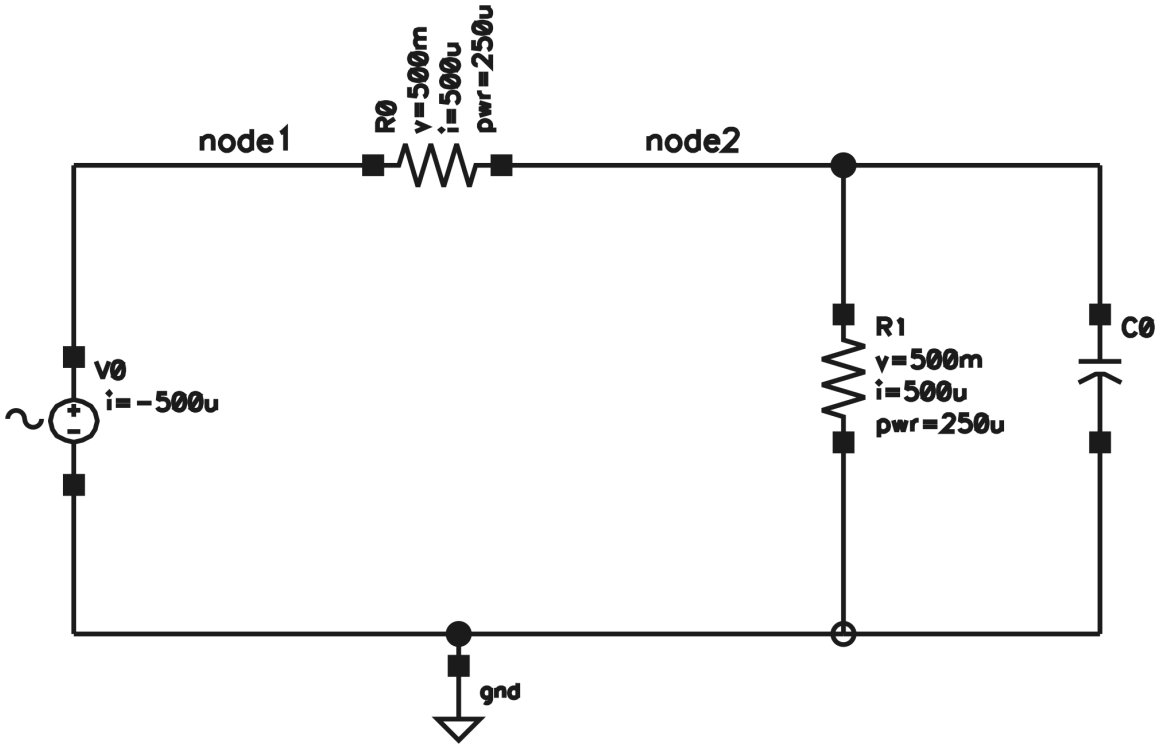
Фиг. 2.6. Прозорец на редактора

Когато се работи на най-ниско ниво от йерархията за всеки елемент, напр. транзистор, резистор, кондензатор и т. н. могат да се задават конкретните му параметри. Схемният редактор притежава модул за проверка на електрическите правила, който издава съобщения за свързани накъсо изходи или несвързани входове на елементите. Тук, както във всеки схемен редактор съществуват стандартните възможности за копиране, триене, местене на елементите или дадена връзка. Копирането е възможно както в рамките на един и същи прозорец (схема), така и между различни прозорци (схеми).

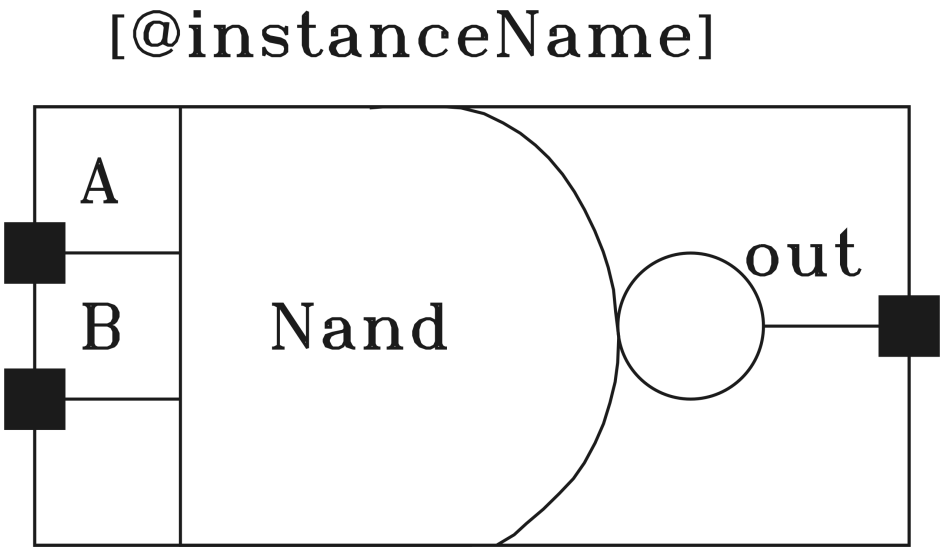
Трети вариант за въвеждане на електрическата схема е чрез модула **Synergy** - **Verilog HDL** на CADENCE.

Редакторът на електрически схеми има модул за проверка на електрическите правила за проектиране, който издава събщения за грешки или предупреждения, когато има проблеми с връзките между елементите (“висящи” изходи или окъсени входове). Както при всеки стандартен редактор за електрически схеми, и тук присъстват функции за преместване, копиране, въртене на елементите. Възможно е също така и копиране между отделни прозорци с различни схеми.

Представянето тип symbol може да се генерира автоматично от представянето тип schematic.



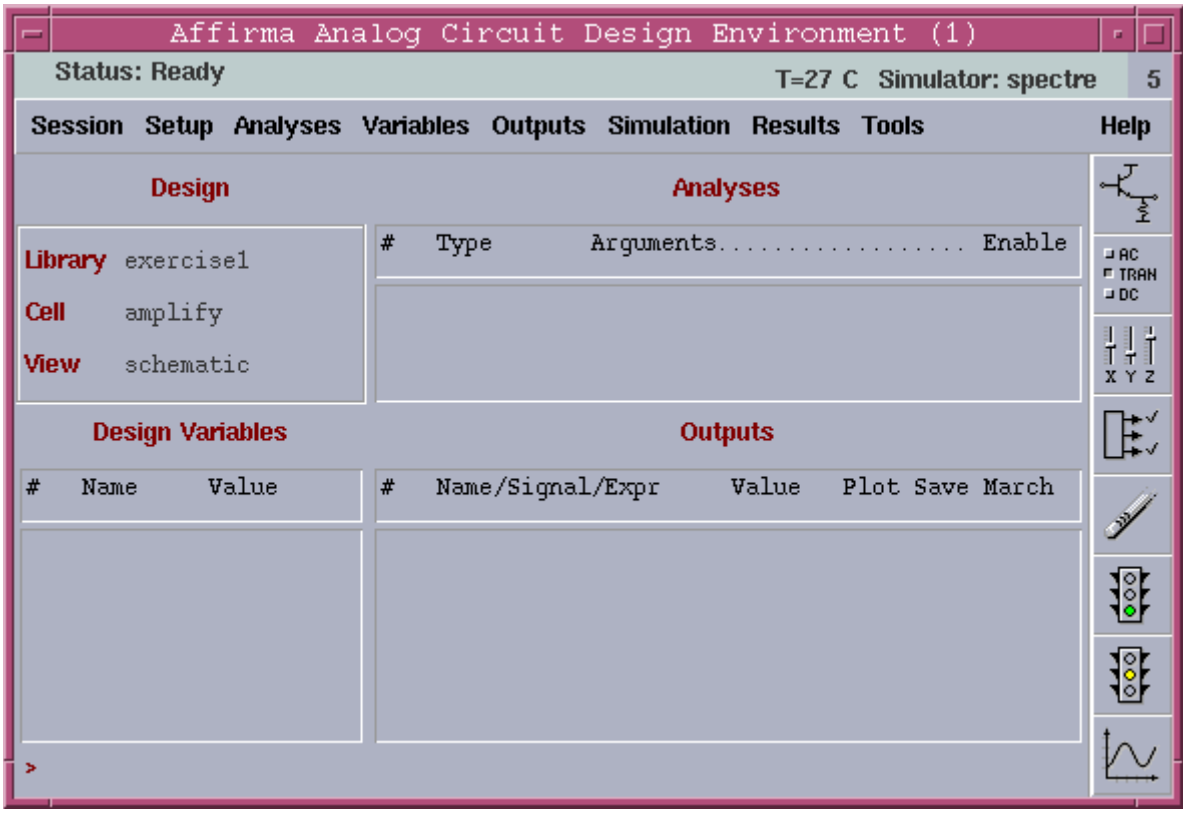
Фиг. 2.7. Представяне тип schematic



Фиг. 2.8. Представяне тип symbol

2.3. Симулация на схеми

Когато е готова електрическата схема може да се пристъпи към извършване на симулация. Симулаторите, които поддържа CADENCE са **SPICE**, **SPECTRE** и **SpectreS** - за аналогова симулация, **VERILOG** - **XL** - за цифрова симулация, както и симулатори за смесена аналогово-цифрова симулация – spiceVerilog, spectreVerilog и spectreSVerilog.



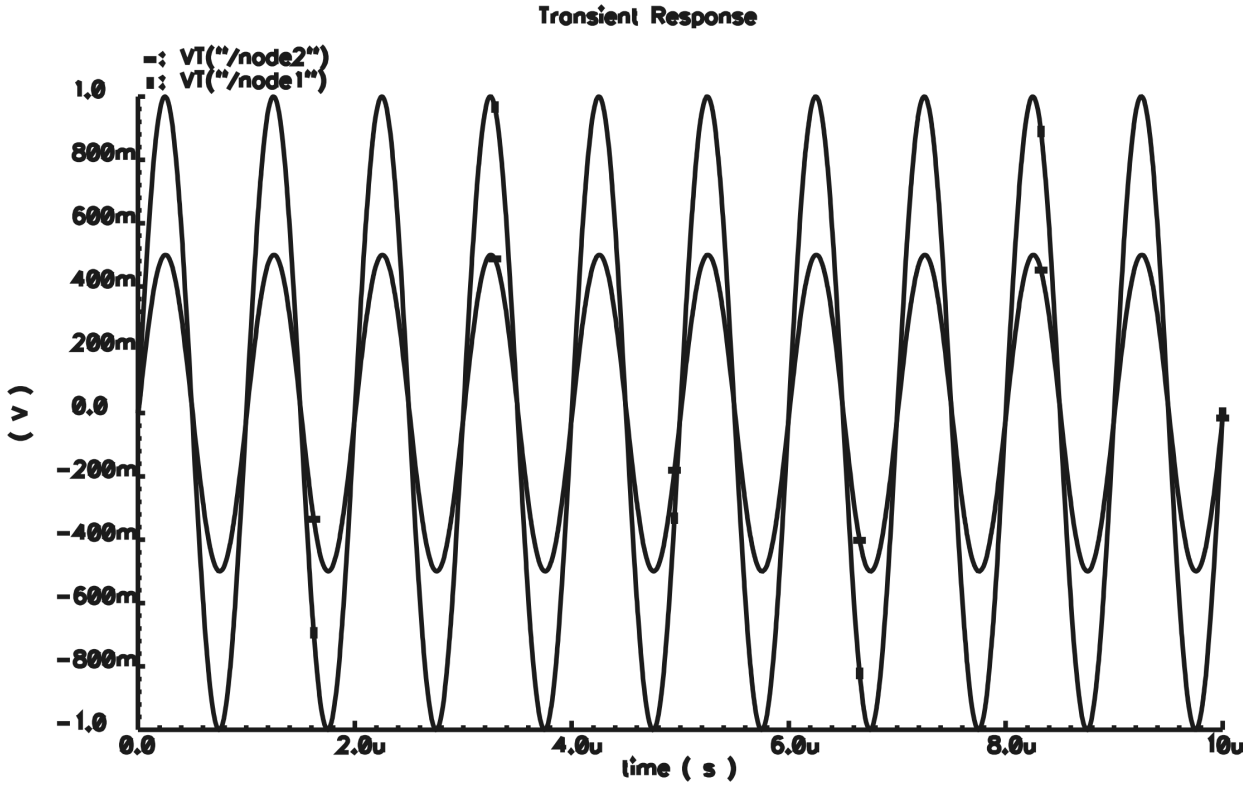
Фиг. 2.9. Среда за аналогова симулация – Analog Environment

При аналоговата симулация, в зависимост от типа на симулатора, могат да се правят преходен анализ, постояннотоков анализ, честотен анализ, Corners анализ, Monte Carlo анализ, S-parameter анализ и параметричен анализ. Изходният формат от симулацията е във вид на графика, като има възможност на нея да бъде представен потенциала на даден възел от схемата или тока през даден елемент. Също така координатните оси могат да бъдат представяни в логаритмичен или линеен мащаб. Сигналите могат да се представят в една или в отделни координатни системи. Съществува възможност да се изобрази графика, която е функция от два или повече сигнала - например за усилването на дадено стъпало от схемата.

На външен вид симулаторите SPICE, SPECTRE и SPECTRES работят по подобен начин, като има някои дребни разлики в съобщенията, които се генерират по време на симулацията. Например при SPECTRE се отчита процента от общия интервал от стойности, за които е извършена симулацията до момента, докато при SPICE това го няма. Разликата между тях е в техните алгоритми и параметрите, които използват от моделния файл. Моделите на елементите, които се използват при аналоговата симулация се вземат от моделните файлове, които се разпространяват със стандартните технологии, с които работи CADENCE - AMS 0.6μm, 0.8μm 0.35μm.

Елементите, които изграждат електрическите схеми задължително трябва да притежават на най-ниско ниво от йерархията си представяне от типа spice или spectre.

Съществува възможност за онагледяване на резултатите още в процеса на тяхното пресмятане. Това е особено полезно в случаите, когато симулацията на дадена схема отнема доста време, тъй като, ако резултатите не са задоволителни, симулацията може да се прекъсне, без да се изчаква края й, за да се видят резултатите.



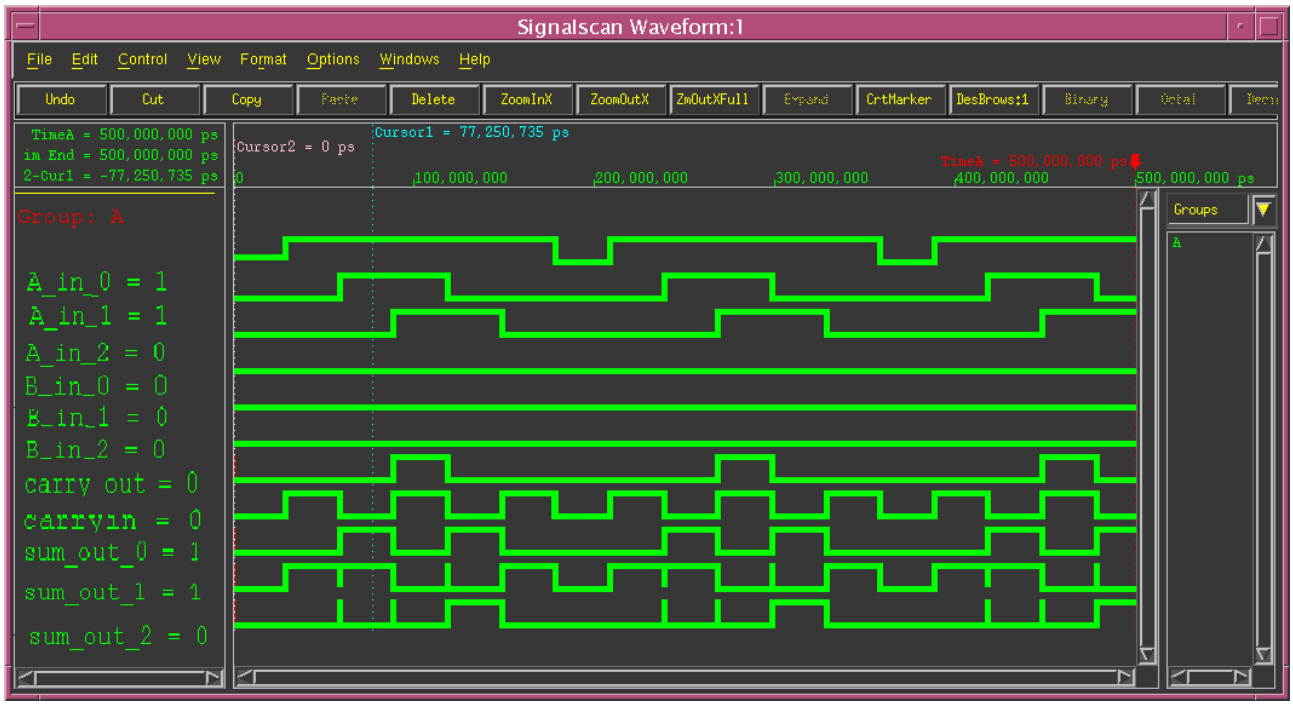
Фиг. 2.10. Визуализатор на резултати от аналогова симулация.

Другият тип симулатор е цифровия симулатор VERILOG - XL. Както при аналоговата симулация и тук на най-ниско ниво от йерархията на схемите, елементите, които ги изграждат трябва да имат представяне от типа verilog. Този тип представяне отчита минималните, максималните и типичните закъснения на логическите елементи.

За да бъде възможно да се осъществи цифровата симулация е необходимо да бъде зададен *файл с входни въздействия*. Този файл се създава в съответствие с изискванията на езика Verilog.

След като се създаде файла с входните въздействия и той няма грешки, системата автоматично генерира поредицата от тестове, като има възможност да се видят зададените входни въздействия под формата на времедиаграми.

След успешна симулация, резултатите от нея се онагледяват във вид на времедиаграми, като изходите, които са прости шини са дадени в двоична логика, а тези които са магистрали са дадени като числа в десетична, шестнадесетична или осмична бройна система.



Фиг. 2.11. Визуализатор на резултати от цифрова симулация – SignalScan

Смесената аналогово-цифрова симулация представлява комбинация от аналогова и цифрова симулации, като за аналоговата част на схемата се задават съответния тип анализ - постояннотоков, честотен или преходен, а за цифровата част - файл с входни въздействия, както при цифровата симулация. Възможно е в този файл да няма входни въздействия, като например в случаите. когато цифровата част на схемата е обградена от аналогова част, но е задължително да има поне един дефиниран цифров изход, макар и да е фиктивен, т. е. реално да не представлява изходен сигнал за схемата. Достатъчно е към някой възел на схемата да се свърже изходен пин.

Симулаторът автоматично прави разделяне на цифровата от аналоговата част, като за целта се използват специални модели на транзистори - т. нар a2d и d2a модели.

Изходните данни са подобни на тези при аналоговата симулация, като особеността тук е, че сигналите от цифровата част на схемата се обозначават само с логически нива.

2.4. Топологично проектиране

Когато резултатите от симулацията удовлетворяват проектанта и всички параметри на елементите са уточнени, може да се пристъпи към създаване на топологията на интегралната схема.

2.4.1. Топологично проектиране на клетка.

Когато елементи, изграждащи дадена схема нямат стандартно топологично представяне в CADENCE, е необходимо това представяне да бъде създадено, за да бъде възможно да се използва в топологията на интегралната схема.

Създаването на топологията на дадена клетка е възможно да стане ръчно или автоматично.

При ръчното създаване на топология се изчертава на ръка всеки детайл от всеки слой на топологията. Този тип изчертаване се използва при чертане на резистори и кондензатори, тъй като този модул не е включен в разглеждания софтуерен пакет.

Автоматичното генериране на топология става, когато най-ниското ниво от йерархията на даден блок от електрическата схема съдържа само транзистори. В този случай се използва модула на CADENCE - **Layout Synthesis**.

Трети вариант е топологията да се въвежда от файл, ако например тя е била генерирана на друго място. Форматите, които се поддържат са: Stream, Applicon, CIF, CALMP, DEF и LEF.

2.4.1.1. Автоматично генериране на топология на стандартна клетка от представяне schematic в представяне layout

В модула, който извършва автоматичното генериране на топологията на стандартни клетки има възможност да се задават параметри, касаещи разположението на входно/изходните пинове, разполагането на транзисторите, опроводяването, захранващите шини. Съществува възможност така зададените параметри да бъдат записани във файл, от който след това могат да бъдат зареждани многократно.

При параметрите на пиновете могат да се оказват начина на достъп до тях от опроводяващия модул; слоевете, на които да се разполагат; размерите им, съобразени с минималните допуски за всеки слой; разстоянието между отделните пинове; отместването спрямо началото на координатната система, в което се намира левия долен ъгъл на клетката, а също така и на коя от страните на клетката да бъдат изведени.

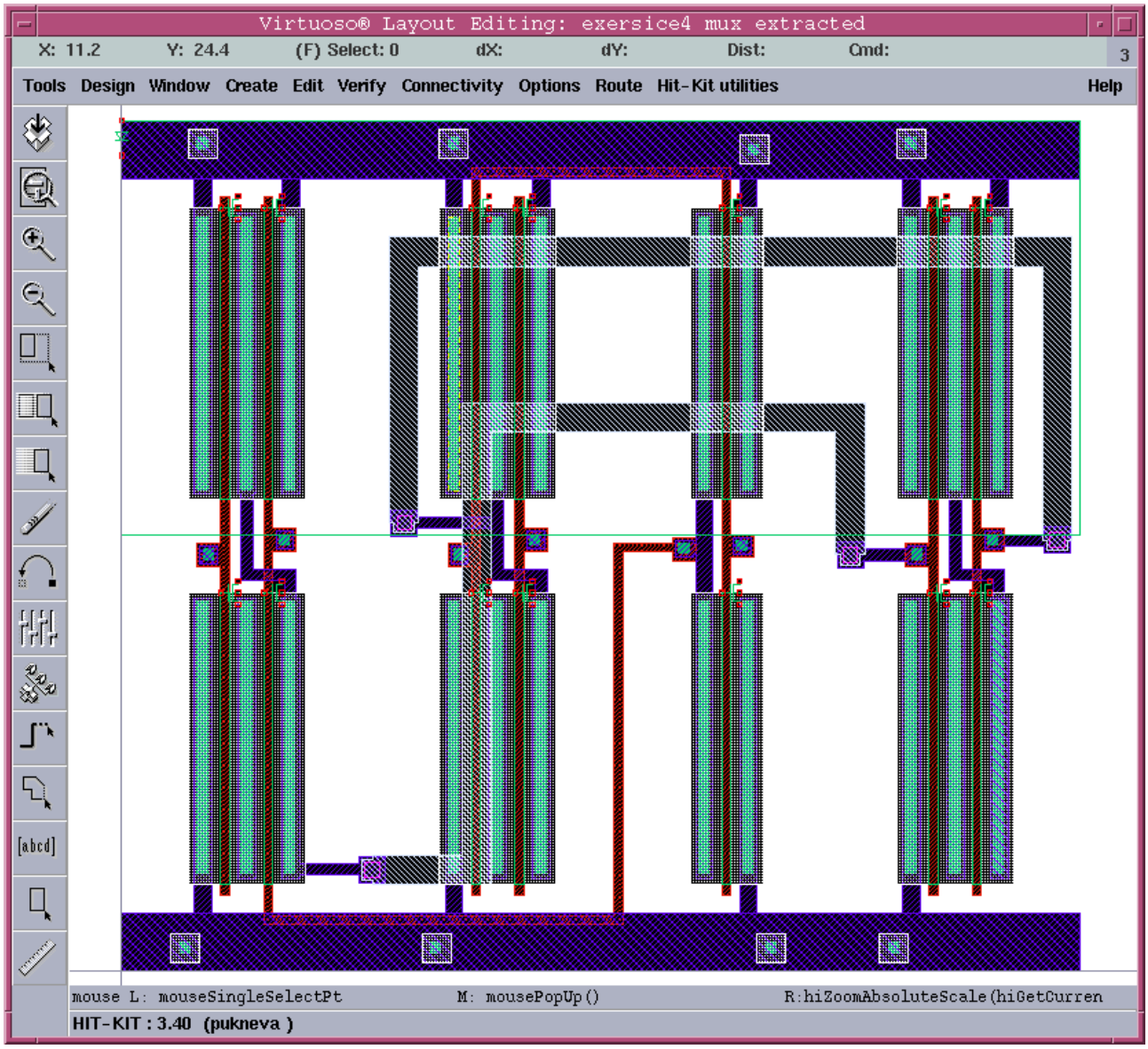
При параметрите на транзисторите може да се оказва метода за тяхното разполагане; отношението широчина към височина на клетката или броя на редове, в които да се разположат транзисторите; максималния размер на транзисторите, който ако бъде превишен ще бъде създадена гребеновидна структура на гейта на транзисторите и др.

При параметрите на опроводяването може да се задава приоритета на металните слоеве; дали гейтовете на отделните транзистори да се свързват с полисилиций или метал, ако има наличие на такива връзки в схемата и др.

За използване на автоматичния метод за създаване на **layout** на клетка е необходимо технологичната библиотека на дадената технология да съдържа съответните описания, които се използват от модула.

2.4.1.2. Ръчно изчертаване на топология на клетка

Тук съществуват три начина – първия позволява напълно ръчно изчертаване на елементите и връзките между тях, втория дава възможност за автоматично разполагане на активните и пасивни компоненти и ръчно изчертаване на връзките между тях. Третият се базира на втория, като след разполагането на елементите, топологията се експортира в IC Craftsman за автоматично свързване на елементите.



Фиг. 2.12. Представяне тип layout

2.4.1.2.1. Напълно ръчно изчертаване.

Съответната клетка се отваря в layout представяне и с помощта на средствата на топологичния редактор се изчертават ръчно елементите на топологията, като се спазват технологичните норми. Дори и да се допусне грешка, модулът за проверка на правилата на проектиране ще я маркира на екрана и тя може да се коригира.

Когато са готови топологиите на всички клетки от схемата се генерира тяхното абстрактно представяне (**abstract**), което се използва

за получаване на топологията на интегралната схема. За целта е необходимо само да се очертаят границите на съответната стандартна клетка, след което автоматично се генерира абстрактното й представяне. Съществува възможност в клетката да се създадат т. нар. **feed** пинове, които служат за улесняване на опроводяването на каналите на интегралната схема. Те служат като мостове между отделните канали.

2.4.1.2.2. Полу-ръчно изчертаване

Използва се модулът **Layout-XL**. Той е подобен на гореописания модул, с тази разлика, че компонентите се разполагат ръчно, а тяхното представяне във вид на топология става автоматично. Това се постига като се маркира дадения елемент в прозореца със електрическата схема (представяне тип *schematic*) и след това се посочи неговото място в прозореца, където е представянето *layout*.

Когато е готова топологията се прави верификация, за отстраняване на евентуални грешки, допуснати при създаване на топологията на интегралната схема. Това се извършва с помощта на модула за **Design Rules Check - Diva**.

2.4.1.2.3. Автоматично изчертаване

Информацията за този вид е дадена по-долу при описанието на модула IC Crafstman.

2.4.2. Проектиране на топология на интегрална схема

Процесът на разполагане и опроводяване на една интегрална схема може да бъде осъществен по няколко различни метода, като в зависимост от модулите, които се стартират се променят специфичните параметри и реда, по който се изпълняват отделните етапи от топологичното проектиране на интегралната схема.

В момента CADENCE поддържа два модула, с помощта на които може да се извърши разполагане и опроводяване на една интегрална схема. Това са IC Craftsman и Silicon Ensemble. Първият се използва при аналогови и смесени цифрово-аналогови интегрални схеми, а втория – при цифрови схеми. Когато се използва IC Craftsman за смесени схеми, цифровата им част се обработва предварително в Silicon Ensemble.

2.4.2.1. IC Craftsman

Този модул има възможности за разполагане, но те не са големи, за това в следващата част на материала се предполага, че клетките, които обикновено не са много при аналогови ИС, са вече разположени в **Layout-XL.**

След стартирането му, модулът проверява целостта и пълнотата на схемата. Маркират се пинове, които са извън координатната мрежа, готови връзки, които имат недопустими размери и др. Това са грешки, допуснати на предходния етап от проектирането и за това е предвидена възможност в менюто те да бъдат проследени.

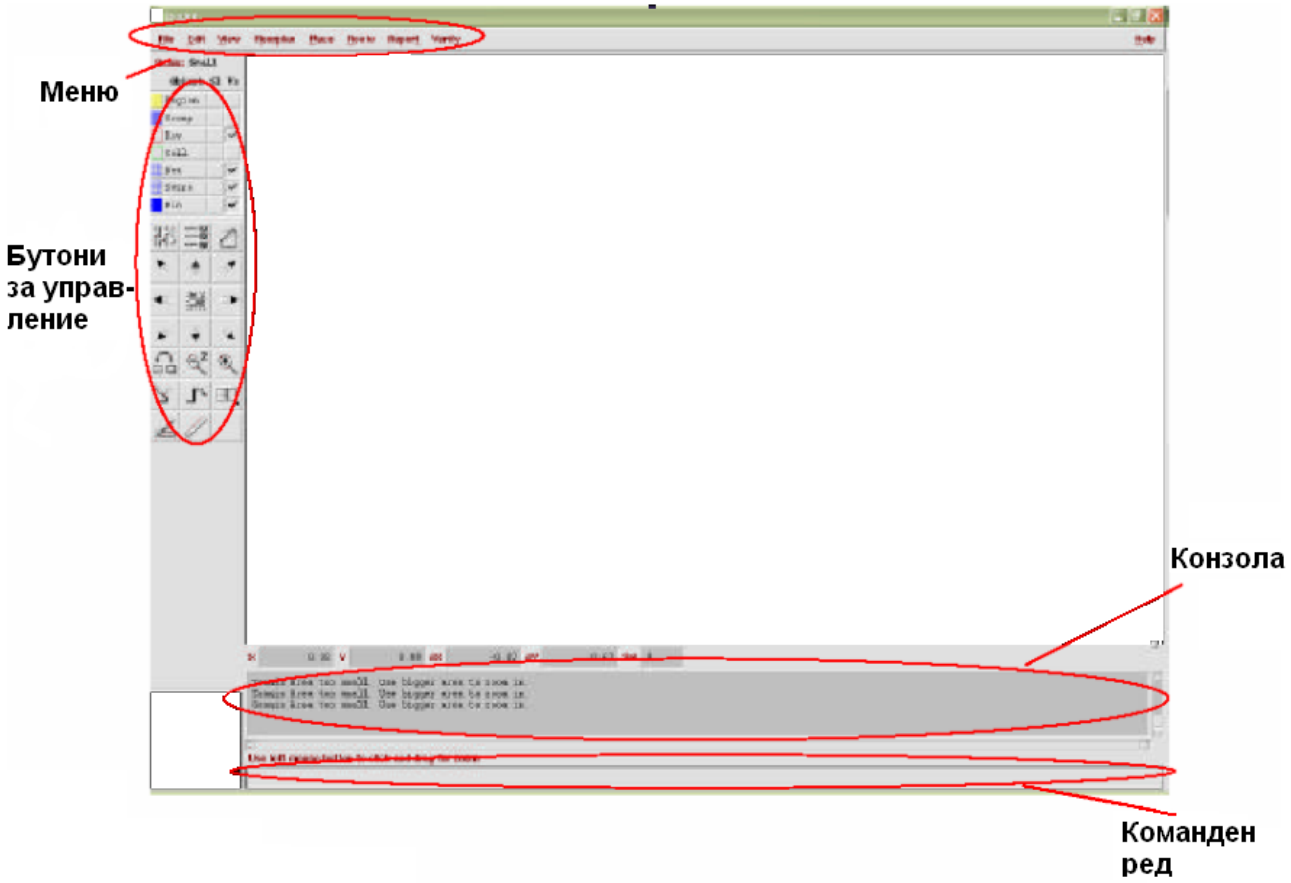
Преди да се започне с опроводяването се задават слоевете, които ще се използват и техните ширини. Ширините могат да се отнасят за всички връзки или да се дефинират индивидуално за всяка една, към която трябва да се подходи с по-голямо внимание.

Първоначално се изгражда захранването през периферните клетки. Посочват се използваните слоеве, нетове и клетки, през които минава то.

След това се прави глобално опроводяване. При него се проверява дали е възможно опроводяването, след което, ако то е възможно се пристъпва и към детайлното опроводяване. Самият процес може да бъде проследен на екрана, тъй като визуално се показва свързването. Ако е оставено твърде малко място е възможно да се получат къси съединения, които не се отчитат по време на опроводяването. За това накрая се проверяват всички връзки, като за целта има специализиран модул, който докладва всички налични нарушения.

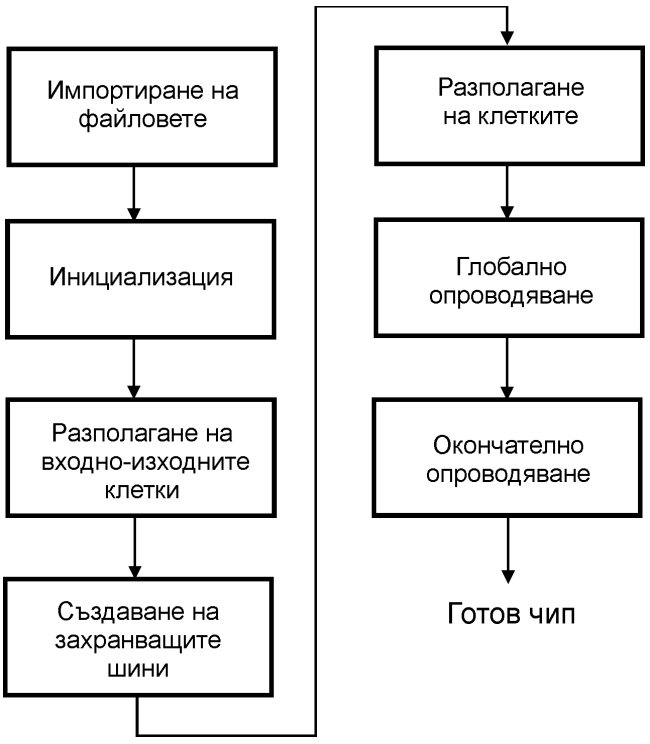
2.4.2.2. Silicon Ensemble (SE)

Когато изходната схема има само представяне тип schematic, процесът започва с използването на транслатора **Prflatten**. Това е транслатор, който създава списък от връзките в схемата във формат, подходящ за модулите за разполагане и опроводяване. Създава се ново представяне на схемата от тип **autoLayout**. От него чрез друг транслатор се получава представянето DEF.



Фиг. 2.13. Прозорец на SE

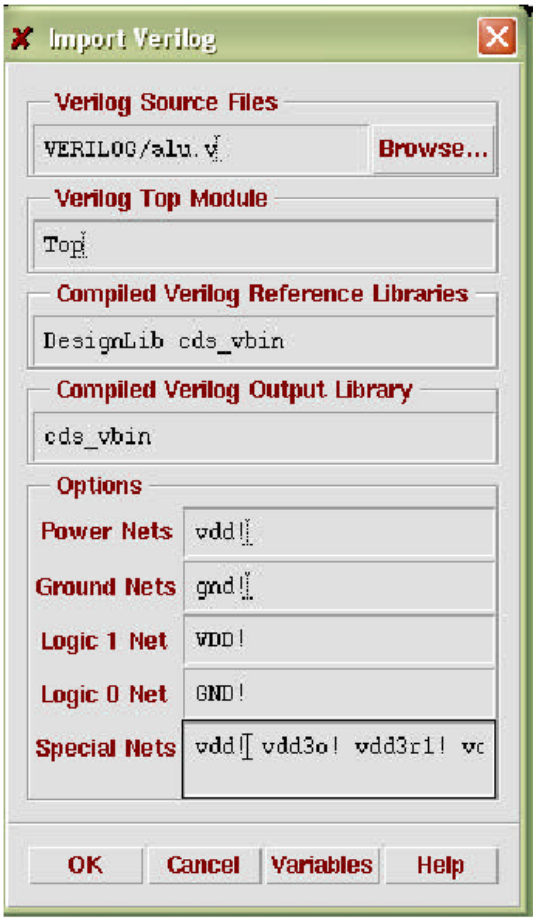
Другият вариант на работа е да се използва описание на схемата на езика Verilog.



Фиг. 2.14. Последователност на работа в SE

2.4.2.2.1. Импортиране на файловете

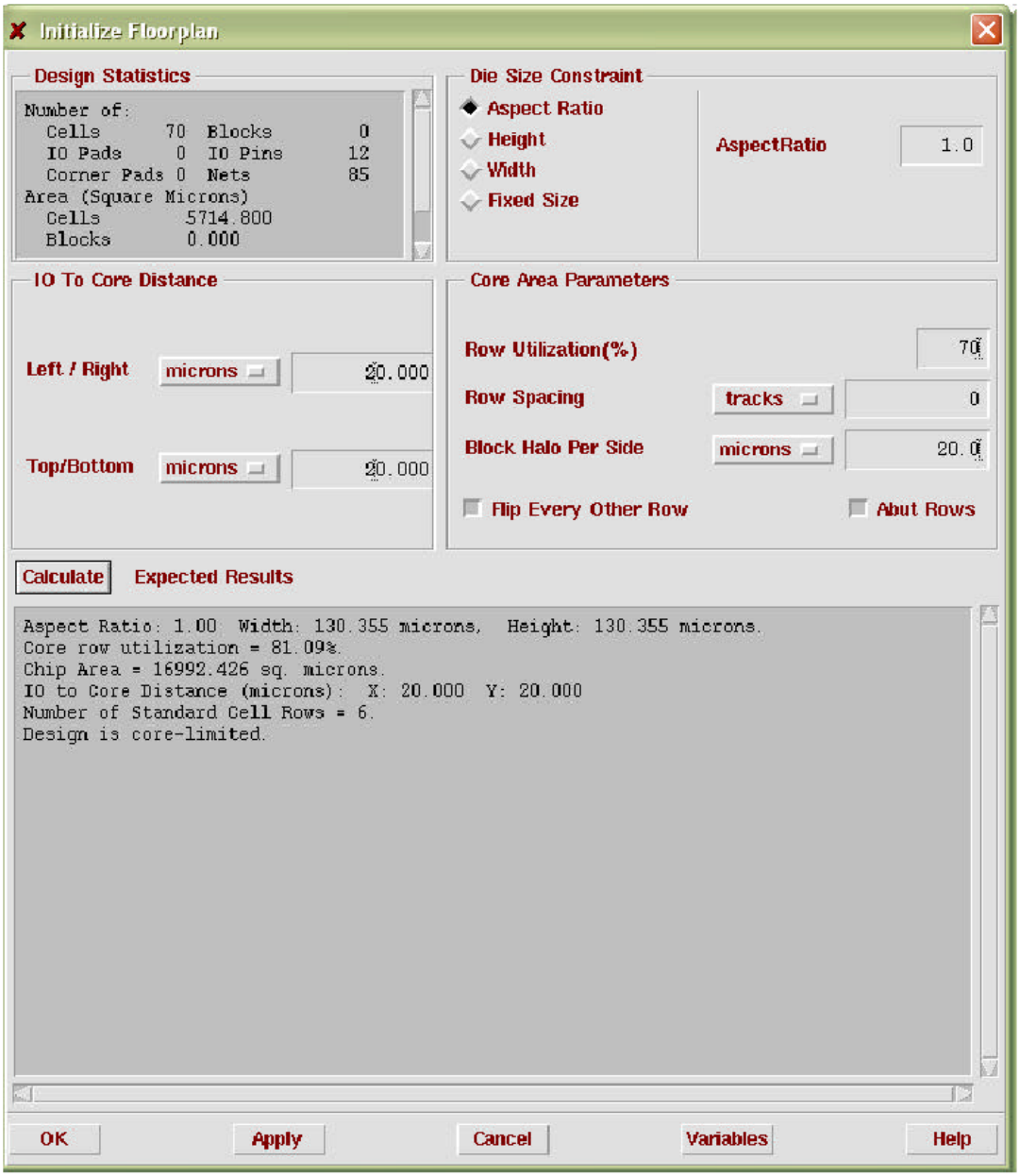
Файловете, които се използват в SE са във формат *LEF* (описание на библиотеката), *TLF* (времеви съотношения, валидни за дадения процес), *DEF* (описание на проекта, получено от представяне тип schematic) или *Verilog*, ако има такъв, като резултат от синтез на схемата (напр. от SYNOPSYS). Реда, в който се импортират файловете е: LEF, TLF и DEF (Verilog) (фиг. 2.15).



Фиг. 2.15.

2.4.2.2.2. Инициализиране

При инициализирането се извършва първоначална оценка за използваната площ и разполагането на стандартните клетки в редове.



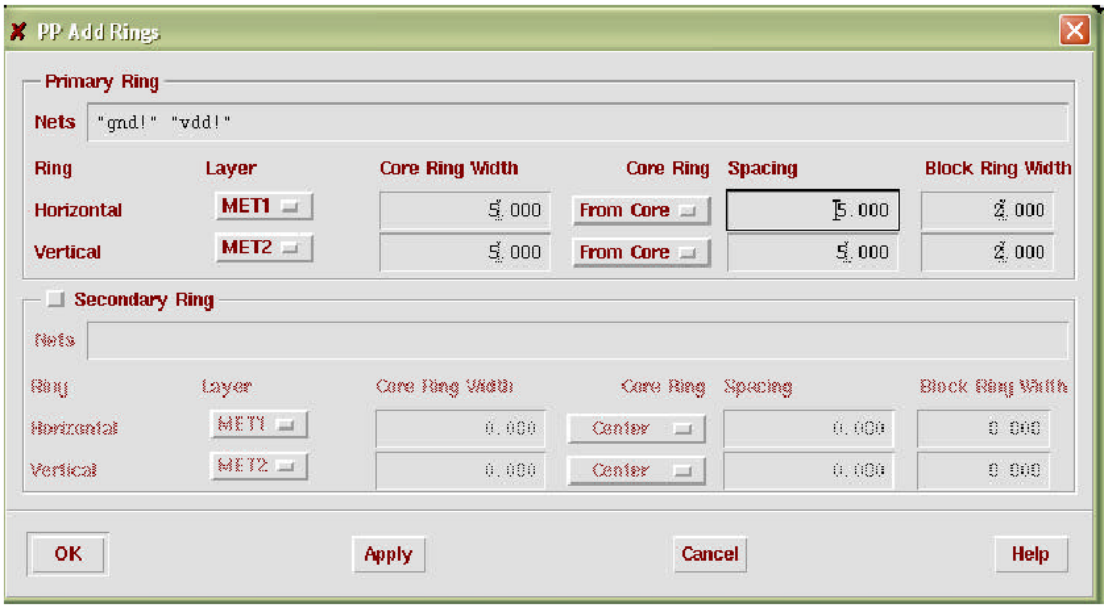
Фиг. 2.16.

2.4.2.2.3. Разполагане на входно-изходните стандартни клетки

Тук се оказва по какъв начин да бъдат подредени входно-изходните клетки по периферията на чипа. Те могат да бъдат разположени равномерно или неравномерно, долепени или разделени.

2.4.2.2.4. Захранващи шини

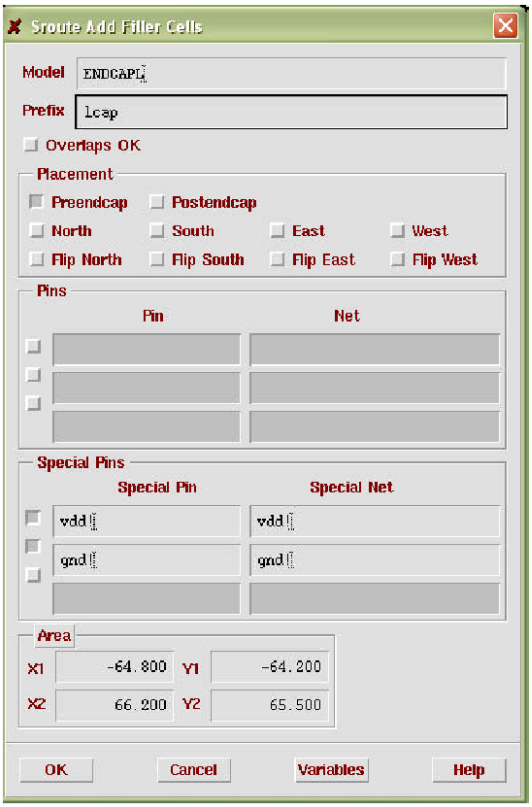
На този етап от проектирането се указва колко слоя ще се използват за захранващи шини, които участват в захранващите пръстени и кои метализации ще използват (броят им зависи от използваната технология). Тук също така се указва и тяхното разстояние до ядрото и периферията на чипа.



Фиг. 2.17.

2.4.2.2.5. Разполагане на ядрото

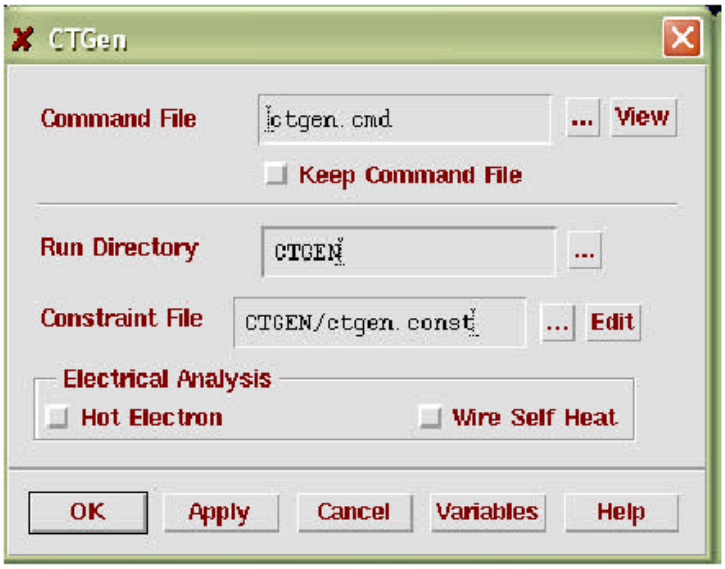
Този етап започва с разполагането на т.нар. *cap cells*. Те се поставят в двата края на всеки ред и спомагат за свързването на захранването на ядрото към захранващите пръстени. След това се извършва същинското разполагане на клетките от ядрото. Използва се модула QPlace. С цел пестене на място накрая се извършва компактиране, така че максимално да се уплътни използваната площ.



Фиг. 2.18.

2.4.2.2.6. Глобално опроводяване

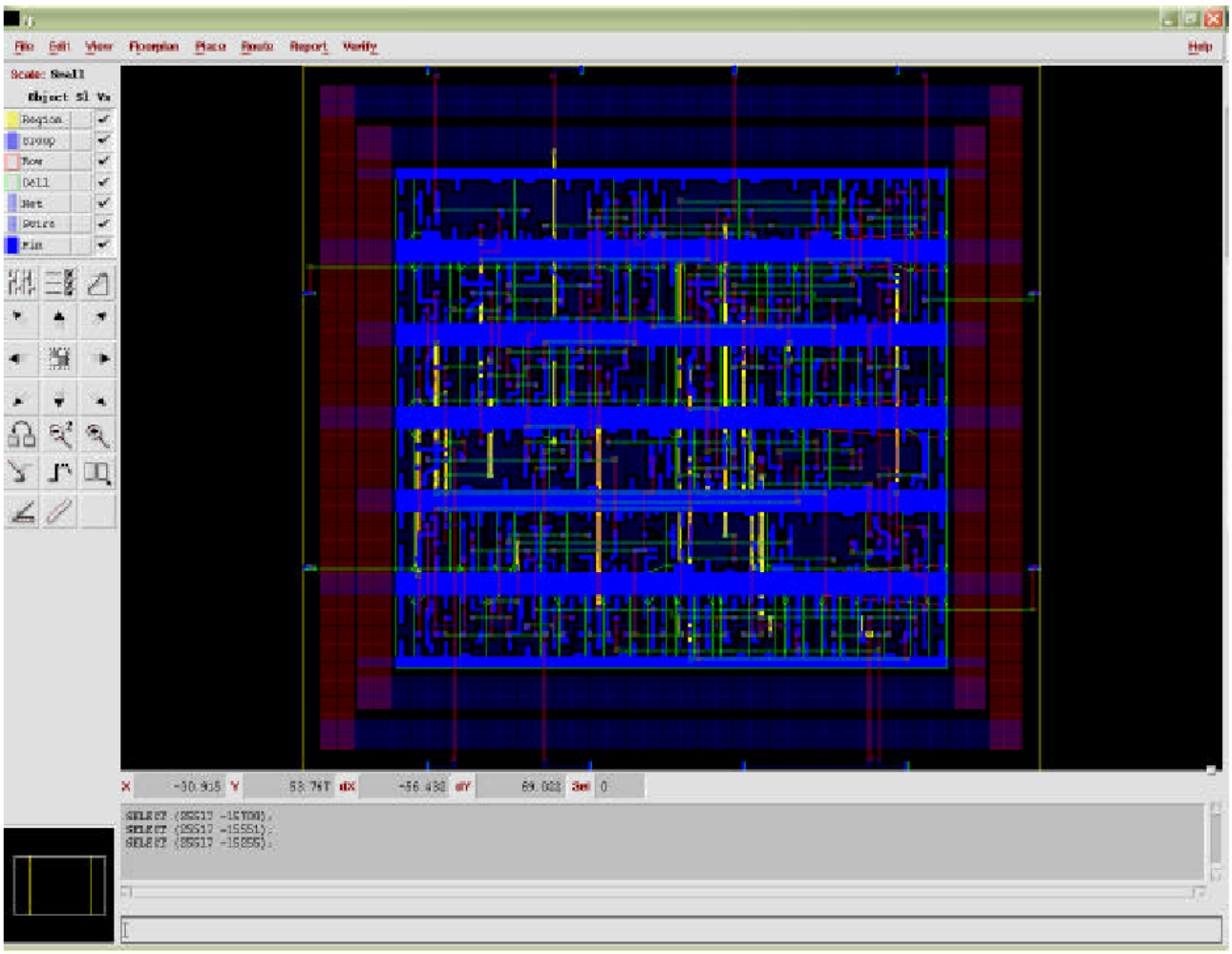
На този етап се генерира дървото на тактовия сигнал, който е най-критичен по отношение на закъсненията в схемата. Използва се специализирания модул **CTGen**. Съществено при него е, че към схемата се добавят автоматично допълнителни елементи – буфери, които после по определен начин се отразяват и в нетлиста на схемата.



Фиг. 2.19.

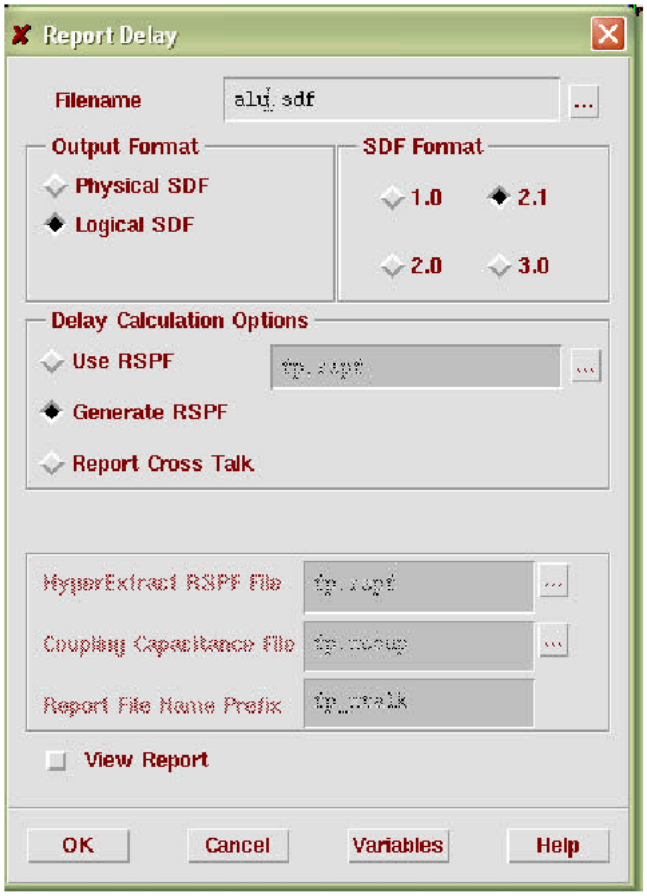
2.4.2.2.7. Окончателно опроводяване

Етапът започва с попълване на празното място в ядрото с т.нар. *feedthru* клетки, с помощта на които захранването на цял един ред се свързва в обща шина. На този етап се свързва захранването на схемата към ядрото и се прокарват останалите връзки между нейните елементи, с което чипа е готов.



Фиг. 2.20. Готов чип

Накрая се извършват различните видове проверки и екстракция на параметрите след разполагане и опроводяване.



Фиг. 2.21. Проверки

Последователността от операциите, които се извършват в SE може да бъде записана във файл, който да се използва в последствие за тяхното повторно, автоматично изпълнение. Такъв файл може да бъде генериран автоматично в началото на процеса, в зависимост от използвания design kit. Файлът съдържа само някои основни, начални параметри на процесите в SE.

ГЛАВА 3

Среда за проектиране Synopsys

3.1. Инструменти за симулация

3.1.1. Scirocco (VHDL симулация)

Scirocco-i дава възможност да се анализират, компилират и симулират описания на проекти във VHDL формат и осигурява набор от средства за симулации и отстраняване на грешки в VHDL описанията. Тези средства предоставят възможности за проверка на сорс кода и визуализиране на резултата от симулацията. Scirocco-i поддържа всички видове описания на проекта, но е оптимизиран за поведенчески и RTL описания. Scirocco-i ускорява цялостната проверка на система чрез бърза и високо ефективна VHDL симулация за функционална проверка на RTL ниво. Неговата изключителна поддръжка за смесени езикови симулации предоставя решение за проблемите с интеграция на интелектуалната собственост и симулация на ниво гейтове.

Основните Scirocco-i програми са:

3.1.1.1. VHDL Analyzer

VHDL Analyzer, *vhdlan*, се използва за анализиране на VHDL сорс файловете на проекта. VHDL Analyzer идентифицира синтактични и семантични грешки в VHDL кода. Ако няма намерени грешки, анализатора създава междинни файлове и ги поставя в библиотеката на проекта.

3.1.1.2. Scirocco-i Compiler

Scirocco-i compiler, *scsi*, се използва за детайлно доразвиване и компилиране на проекта на най-високо ниво. Scirocco-i Compiler извършва детайлно разработване, генериране на код и статично свързване на всички обекти, за да създаде изпълнима симулация(*scsim*). Генерираната изпълнима симулация провежда симулацията и осигурява мощен език за нейния контрол Simulation Control Language, базиран на TCL, който може да се използва за контрол и наблюдение на симулацията. За да симулира компилиран VHDL проект, потребителя просто стартира изпълнимия файл - резултат от компилационния процес (*scsim*). По време на симулацията Scirocco може да улови и съхрани данни за протичането на симулацията в двоичен формат, използвайки VCD+ файл формат, така че тя да може да бъде разгледана и анализирана след като симулацията е извършена.

3.1.1.3. VirSim

VirSim е графична среда за изследване и откриване на функционални грешки, която дава възможност да се контролира интерактивната симулация или да се анализират съхранените резултати от симулацията. VirSim може да се използва за да се проследят сигналите докато се разглеждат обявените стойности в първичния код или в схемните диаграми. С VirSim могат също да се разглеждат и сравняват времедиаграми, да се извлича информация за сигналите, да се създават HDL тестове на базата на изходните времедиаграми.

Synopsys предлага еквивалентни по мощност и способности инструменти за симулация за Verilog и SystemC.

3.1.2. VCS™

VCS е високо ефективен симулатор на Verilog® описания, който обединява различни технологии на абстрактни проверки на високо ниво в една платформа.

3.1.2.1. VCSi

VCSi е алтернативна версия на VCS. VCS и VCSi са идентични с изключение на това, че VCS е по-високо оптимизиран, което довежда до по-голяма скорост за RTL и смесени проекти.Независимо от това и двата симулатора VCS и VCSi предоставят един и същ резултат от симулацията. Проекти на ниво гейтове протичат с еднаква скорост.

3.1.2.2. VCS MX

VCS MX е комбинация от VCS и Sciroccо. Той поддържа смесени аналогови/цифрови заедно със смесени HDL проекти.

- Превръща Verilog проект във VHDL проект и после симулира смесен HDL проект, използвайки Sciroccо.
- Превръща Verilog проект във VHDL проект и после симулира смесен HDL проект, използвайки VCS.

И двата, VHDL и Verilog симулатора (Sciroccо-I и VCS) имат C интерфейс за използване на модели и приложения, базирани на езика C.

3.1.3. CoCentric System Studio

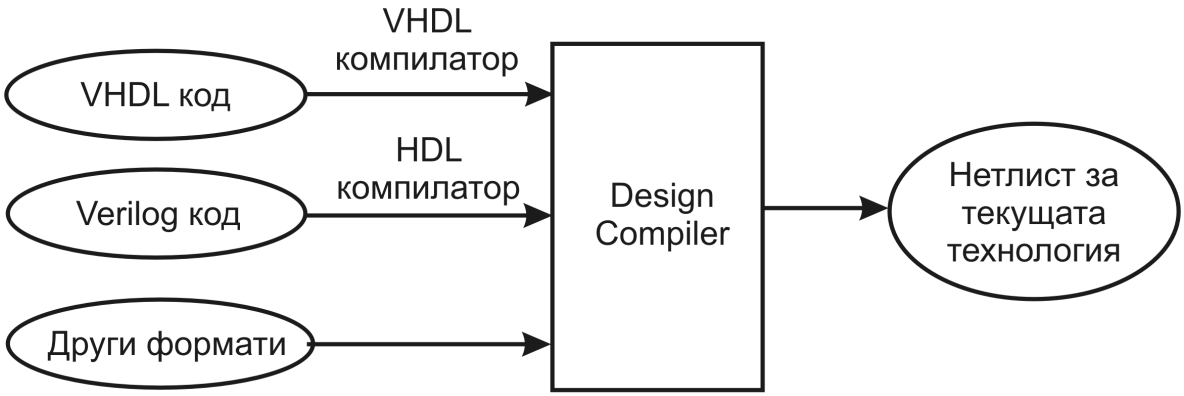
Продуктът CoCentric System Studio е SystemC симулатор и среда на комбинирани проверки и анализи на алгоритмични, архитектурни, хардуерни и софтуерни модели на множество нива на абстракцията.

Може да се симулират смесени SystemC и HDL блокове със System Studio и HDL симулаторите чрез механизми за импорт и експорт, в зависимост от това, дали главния фокус на проекта е SystemC или HDL.

3.2. Инструменти за синтез

3.2.1. Design Compiler

Продуктът Design Compiler е ядрото на софтуерните продукти за синтезиз на Synopsys. Той представлява инструменти, които синтезират вашите HDL проекти в оптимизирани технологичнозависими проекти на гейт ниво. Той поддържа широка гама от плоски и йерархични стилове за проект и може да оптимизира и комбинативна и последователна логика за скорост, площ и мощност.



Фиг. 3.1. Design Compiler

Design Compiler чете и пише файловете на проекта във всички стандартни EDA формати (electronic design automation), включително вътрешния за Synopsys (.db) В допълнение, Design Compiler предоставя връзки към инструментите на EDA, като механизмите за разполагане и опроводяване, и към техниките за ресинтезиране след Layout, като оптимизация на разполагането. Тези връзки позволяват поделяне на информация между Design Compiler и външните механизми.

Design Compiler има два интерфейса:

- Интерфейс за команден ред на Design Compiler или shell, познат като *dc_shell*. Този интерфейс поддържа двата езика Design Compiler shell language (dcsh) и инструментния команден език Tcl (tool command language).
- Графичен интерфейс на Design Compiler (graphical user interfaces GUI), Design Analyzer или продуктът Design Vision.

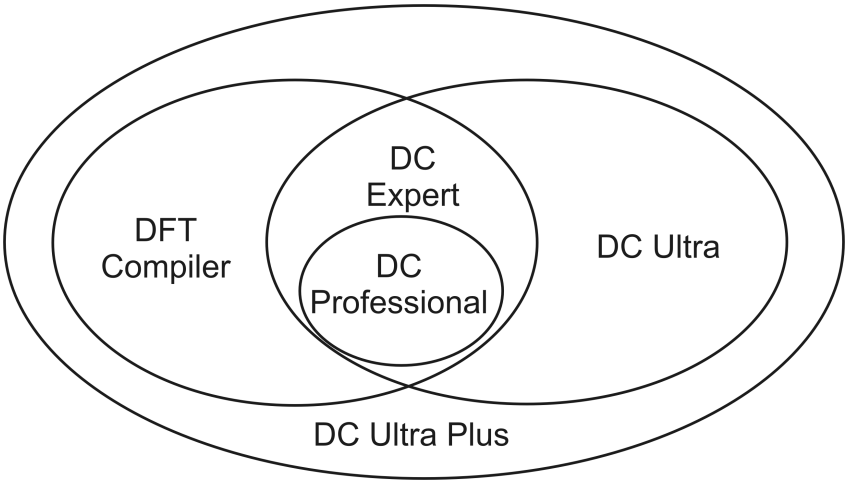
Използвйки продуктите на Design Compiler, проектантите могат:

- Да изработват бързо, ефективни по площ ASIC проекти.
- Да превеждат проекти от една технология в друга.
- Да проучват компромисите в проекта, които включват бързодействие, площ и мощност под различни условия на натоварване, температура и потенциал.
- Синтезиране и оптимизиране на крайни автомати.

Продуктите на Design Compiler включват:

- DC Professional;

- DC Expert;
- DFT Compiler;
- DC Ultra;
- DC Ultra Plus.



Фиг. 3.2. Продукти на Design Compiler

На фиг. 3.2, вътрешния кръг – DC Professional, представлява основната функционалност на Design Compiler. DC Professional е включен във всички други продукти. Елипсите представляват продуктите с допълнителна функционалност. Поделените области представляват обща функционалност на продукта. На пример DFT Compiler включва DC Expert функционалност, която на свой ред включва DC Professional функционалност.

3.2.1.1. DC Professional

Механизмите на DC Professional се прилагат при типични ASIC проекти, които използват CMOS технология. Тези проекти могат да използват множество тактуващи сигнали, но всички те трябва да имат една и съща честота. DC Professional не поддържа заемане на време (time borrowing) при проектите с тригери управлявани по ниво.

Наборът от характеристики предоставени в продукта изграждат основните признаци на синтезиса и са налице при всички продукти на Design Compiler.

Основните признаци на синтеза включват:

- Йерархично компилиране (top down или bottom up);
- Пълни и допълнителни техники за компилиране;
- Последователна оптимизация за сложни тригери;
- Оптимизация на входно/изходните операции;
- Оптимизация на крайната машина на състоянието (FSM);
- Балансиране размера на буферите (в рамките на йерархичните блокове).

3.2.1.2. DC Expert

Инструментите на DC Expert се прилагат към ASIC и IC проекти с висока производителност.

В допълнение на основните синтезисни възможности на DC Professional, DC Expert осигурява следните характеристики:

- Мултичестотни тактове;
- Планиране на време;
- Ре-синтез на критичния път;
- Преоразмеряване на минималния период на такта;
- Оптимизация на място.

3.2.1.3. DC Ultra

DC Ultra се прилага към дълбоко субмикронни ASIC и IC проекти с високопроизводително изпълнение, където се изисква максимален контрол над оптимизационния процес.

В допълнение на способностите на DC Expert, DC Ultra осигурява следните характеристики:

- Допълнителни усилени алгоритми за оптимизиране на закъснението;
- Поддръжка на правилата за разлагане на клетките;
- По-добър контрол на оптимизацията.

3.2.1.4. DC Ultra Plus

Инструментите на DC Ultra Plus се прилагат към дълбоко субмикронни ASIC и IC проекти с високопризводителни изпълнение, които използват сканиращи тестови техники.

В допълнение на способностите на DC Ultra, DC Ultra Plus осигурява интегрирани тестови (design-for-test) способности.

3.2.1.5. DFT Compiler

Механизмите на DFT Compiler се прилагат към ASIC и IC проекти с високопроизводително изпълнение, които използват сканиращи тестови техники.

В допълнение на способностите на DC Expert, DFT Compiler осигурява интегрирани тестови способности.

3.2.1.6. Design Compiler FPGA

Продуктът Design Compiler FPGA има вградена архитектурно определена FPGA синтезисна технология, която осигурява плавно преминаване между ASICs и FPGAs. Design Compiler FPGA включва същата функционалност като Design Compiler, но има допълнителна способност да извършва оптимизация на проекта, насочена към FPGA архитектури.

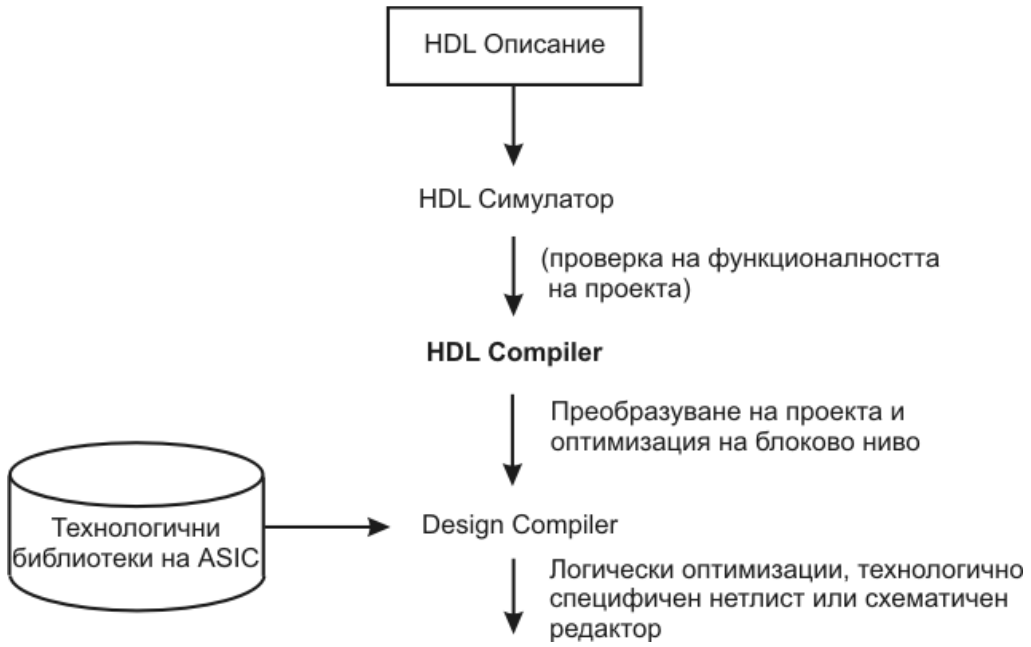
3.2.2. Design Analyzer

Design Analyzer е графичния интерфейс за редица продукти за синтез: Behavioral Compiler, BSD Compiler, DC Expert, DC Professional, DC Ultra, DesignPower, DesignTime, DesignWare Developer, DFT Compiler,

FloorPlan Manager, HDL Compiler, Library Compiler, Power Compiler, VHDL Compiler.

Design Analyzer предлага следните възможности (фиг. 3.3):

- Установяване на стойностите на системните променливи, като желаните технологични библиотечни имена.
- Разчитане и запис на проекта в многочислен формат като EDIF, PLA, Verilog, VHDL или във вид на уравнение.
- Графично представяне на атрибутите на схемата - клетки, изводи, възли, магистрали и др.
- Работа с йерархичен проект:
  - разглеждане и движение по нивата на йерархия;
  - представяне на схеми и подсхеми като блокове с входно/изходни портове;
  - графично групиране и разгрупиране на клетки и подсхеми.
- Синтезиране на цифрови схеми:
  - поставяне на базови модели и връзките между тях, дефиниране на очакваните операционни състояния на схемата;
  - синтезиране на йерархични схеми, поддържане на схемни и подсхемни ограничения по време на изпълнение на глобалната оптимизация;
  - извличане на машини на състоянията (FSM) от последователна логика, дефиниране на състоянията, изпълнение на специфична за FSM оптимизация и синтезиране на еквивалентна схема от последователен тип;
  - присъединяване на тестови последователности към вече съществуващ проект, създаване и форматиране на тест вектори и генериране на съобщения за грешки.



Фиг. 3.3. Последователност при проектиране с Design Analyzer

- Генериране, представяне и изчертаване на схемата.
- Предоставяне на изчерпателна информация за съставните части на схемата (портове, изводи, клетки, възли и др.).
- Визуализиране на главните пътища на схемата.
- Генериране на различни видове съобщения.

3.2.2.1. HDL Compiler

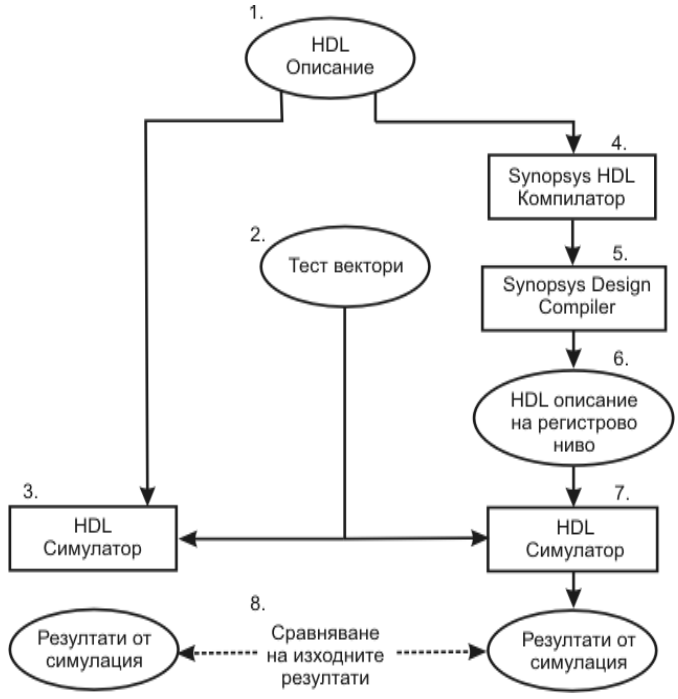
Продуктите на HDL Compiler включват:

- HDL Compiler (Presto Verilog);
- VHDL Compiler.

Verilog или VHDL compiler чете HDL файловете и извършва превод и архитектурна оптимизация на проектите към вътрешен еквивалент на гейт ниво. Тези вътрешни файлове после се компилират от инструмента Synopsys Design Compiler, за да направи оптимизиран проект на гейт ниво в дадена ASIC технология. Подходящият HDL компилатор се извиква автоматично от Design Compiler, когато той чете HDL файл на проекта.

Най-напред се симулира входното VHDL описание за проверка на функционалността на проекта. Използва се HDL симулатор като VCS или Scirocco. След това входното описание се транслира и оптимизира с помощта на HDL компилатор - HDL Compiler(Presto Verilog) или VHDL Compiler). Следваща стъпка е извикването на Design Compiler за синтезиране на схемата в съответствие с правилата за синтез. Правилата за синтез са съставени от три части - методология на схемата, стил на схемата и конструкция на езика.

Ф



Фиг. 3.4. Типичен пример за употребата на HDL компилатор

Когато HDL компилаторът чете HDL описание, проектът се превежда от Design Compiler във вътрешен формат. По време на логическата оптимизация на проекта, Design Compiler може да декомпозира част или цялата схема. Нивото на декомпозиция се контролира от потребителя:

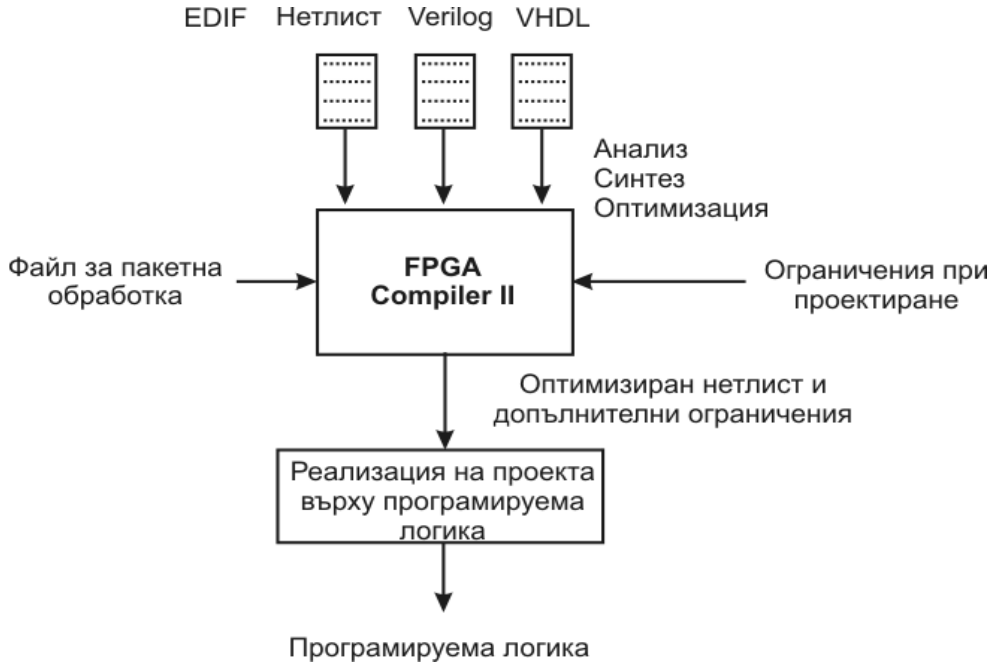
- Йерархията на проекта може да се запази.
- Могат да се предвижват отделни модули нагоре и надолу по нивата на йерархия.

- Могат да се комбинират модули.
- Цялата схема може да се събере в един модул.

В Design Compiler всеки проект може да се запише в различни формати, включително VHDL и Verilog. Това описание може да обслужва първоначалния проект или да се използва като отправна точка за изпълнение на нова технология.

1. Запис на проекта във VHDL формат. Това описание може да бъде комбинация от структурни и функционални елементи.
2. Тест вектори за симулацията и генериране на изходни данни.
3. Симулация на проекта с помощта на HDL Simulator за проверка на прецизността на входното описание.
4. Синтезиране на HDL описание с помощта на HDL компилатор (HDL Compiler(Verilog Presto) или VHDL Compiler). HDL компилатора извършва архитектурна оптимизация и създава вътрешно представяне на проекта.
5. С помощта на Design Compiler се изработва оптимизирано описание на ниво гейтове за изпълнение на дадена ASIC технология.
6. Използва се Design Compiler за създаване на изходно гейтово описание. То има същата конструкция на портовете и модулите, както оригиналното HDL описание.
7. Симулация на гейтовото описанието с оригиналните тест вектори.
8. Сравнение на изхода от симулацията на ниво гейт с изхода на симулацията на оригиналното HDL описание за проверка на правилността на изпълнение.

3.2.2.2. FPGA Compiler II



Фиг. 3.5. Проект с FPGA Compiler

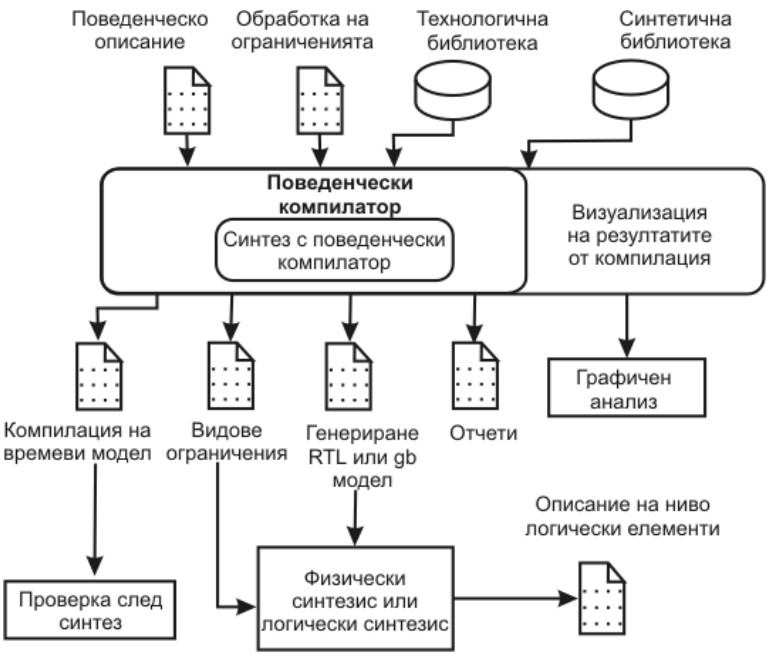
Продуктът FPGA Compiler съществува или като самостоятелен продукт или като опция на Design Compiler. Като опция на Design Compiler (Design Compiler FPGA), FPGA Compiler позволява използване на FPGA технологични библиотеки и формати на данни както и осигурява FPGA алгоритми за оптимизация и FPGA реализация. Като самостоятелен продукт FPGA Compiler II осигурява FPGA алгоритми за оптимизация и FPGA реализации.

3.2.2.3. Behavioral Compiler

Инструментът Behavioral Compiler синтезира HDL хардуерно описание, написано на поведенческо ниво на абстракция, в RTL или нетлист на гейт ниво. Този нетлист може да се използва като входяща информация за други продукти на Synopsys, като инструментите Design Compiler или Physical Compiler.

3.2.2.3.1. Входни данни на Behavioral Compiler

Behavioral Compiler изисква HDL описание на проекта и библиотеки, дефиниращи компонентите и технологията, които ще бъдат използвани за реализиране на хардуера.



Фиг. 3.6. Потокът данни към и от Behavioral Compiler

Поведенческо описание

Използвайки абстракция на поведенческо ниво, функционалността на проекта се описва на поведенческия компилатор (Behavioral Compiler). Описанието на функционалността на проекта се използва, когато Behavioral Compiler чете входните данни, когато извършва операции с входните данни и когато вписва обработените резултати в изходната информация. Не е необходимо да се определя крайна машина на състоянието (finite state machine FSM) за управление на проекта или тактуващ сигнал, когато се изпълнява всяка входно изходна операция; Behavioral Compiler решава това по време на поведенческия синтез. Поведенческото описание е независимо от технологията и архитектурата на изпълнение. Използвайки Behavioral Compiler, може да се промени желаната технологична библиотека или да се вмъкнат

ограничения без да се променя поведенческото описание. Това позволява бързо проучване на различни реализации на проекта.

Технологична библиотека

Определя се коя ASIC технология да се използва за реализиране на проекта. Технологичната библиотека съдържа основни компоненти като например логически гейтове, тригери управлявани по ниво, модели на wire load и работни условия, които Behavioral Compiler използва по време на синтеза.

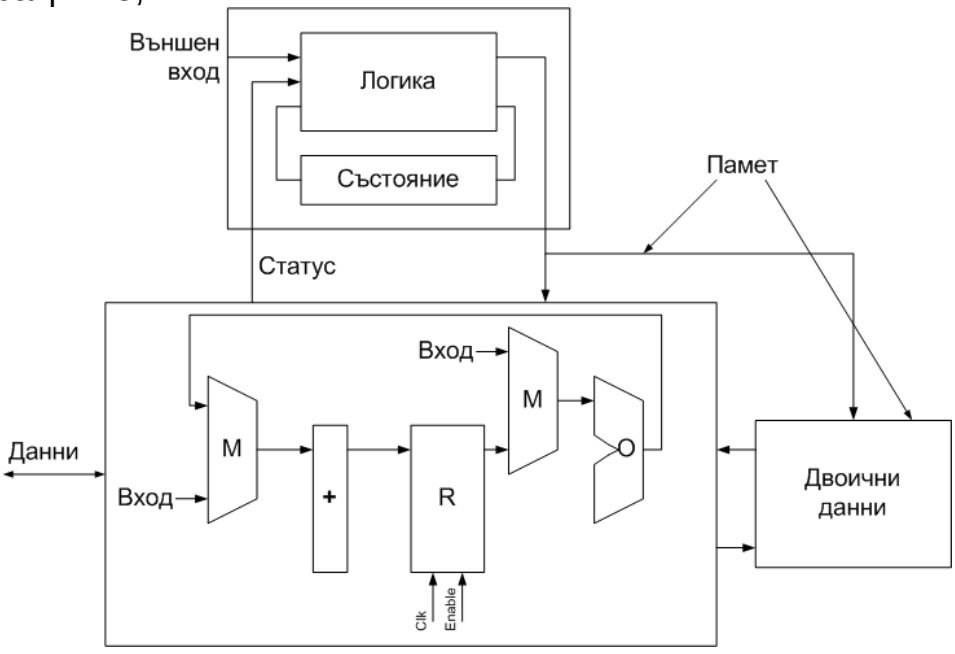
Синтетична библиотека

Синтетичната DesignWare библиотека е технологично независима библиотека от логически компоненти като суматори и умножители. Behavioral Compiler решава кои компоненти се използват от проекта и автоматично поставя компонентите от синтетичната библиотека за реализации използвани от избраната технологична библиотека.

3.2.2.3.2. Синтез с Behavioral Compiler

Behavioral Compiler синтезира хардуер от поведенческо описание като:

- Определя времето на всички операции съобразно технологичната библиотека;
- Планира за всяка операция четенето на входа и извеждането на изхода да се осъществят при зададен такт;
- Разпределя синтетичните компоненти, които да извършат операциите в проекта;
- Разпределя регистрите за съхраняване стойностите на променливите, на сигналите и на междинните резултати от операциите;



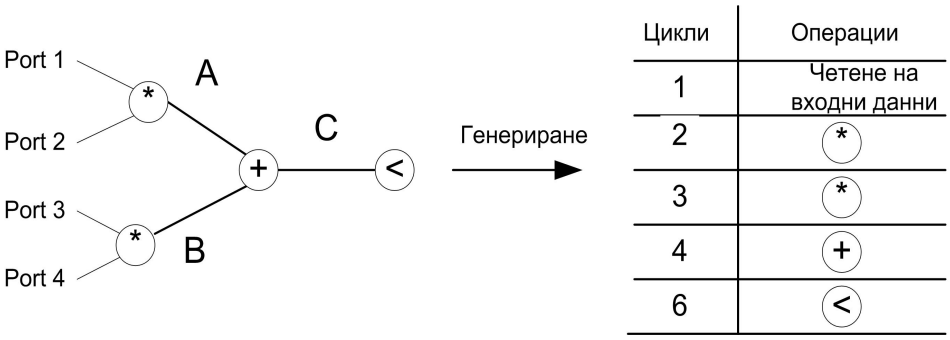
Фиг. 3.7. Структура на схемата създадена от Behavioral Compiler по време на синтеза

- *Изгражда път на данните с разпределените синтетични компоненти и регистри като вкарва мултиплексори и междинни връзки при необходимост;*
- Създава краен автомат на състоянието и контролна логика за интерфейса на паметта.

Behavioral Compiler генерира проект, който се състои от краен автомат, път на данните и памет (фиг. 3.7).

3.2.2.3.3. Времеразпределяне

При времеразпределянето Behavioral Compiler решава кой компонент от синтетичната библиотека може да използва, за да осъществи операциите в поведенческото описание и изчислява закъснението при всеки компонент. Behavioral Compiler ползва основните компоненти, така наречените “wire load” модели и работните условия от ASIC



технологичната библиотека, за да изчисли времеразпределянето.

По време на плани-рането и разпределянето Behavioral Compiler използва времеви оценки, за да синтезира най-ефективното изпълнение за проекта.

Фиг. 3.8. Планиране на времето

Целите на Behavioral Compiler по време на планирането са:

- Да удовлетвори зависимостите на потока от данни и управлението описани в поведенческото описание.
- Да удовлетвори всякакви ограничения на латентност, продуктивност и период на такта зададени с командите на dc_shell.
- Минимизиране броя на тактовите цикли, необходими на завършването да осъществи зададената функционалност.
- Да спомогне споделянето на ресурсите чрез разпределяне на операциите между наличните тактови цикли, за да минимизира използваната площ.
- Да спомогне споделянето на регистрите като интелигентно създава и използва данни, за да минимизира използваната площ.

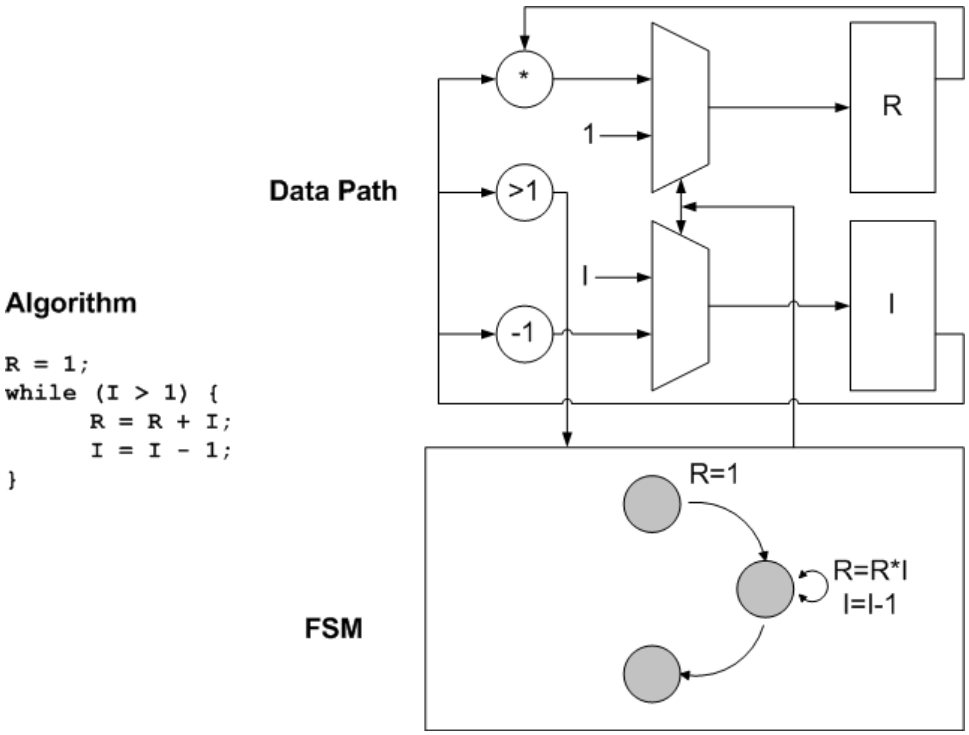
Входно-изходните операции между синтеза от Behavioral Compiler и външната среда обикновено имат предварително определен протокол. Входно-изходния протокол обикновено дефинира или пълен синхронен

интерфейс с фиксиран такт за всеки вход и изход или handshake-управляван интерфейс, където относителния ред на входящите и изходящите данни е определен, а тактът може да варира.

3.2.2.3.4. Разпределяне на хардуера

По време на разпределянето към проекта се добавят синтезираните компоненти, за да изпълнят операциите в поведенческото описание и регистрите, за да съхраняват стойностите на променливите, сигналите и междинните данни, произведени от операциите. Когато е целеоправдано, Behavioral Compiler минимизира площта чрез споделяне на компонентите между операциите и на регистрите между стойностите.

След като проекта е разпределен по циклите на такта, Behavioral Compiler разпределя хардуера. Той добавя синтетични компоненти към реализацията, за да изпълнят операциите в поведенческото описание и добавя регистри, за да съхраняват стойностите на променливите, сигналите и междинните данни получени при извършване на операциите.



Фиг. 3.9. Хардуерно разпределяне с един множител, един суматор и един компаратор

Един и същ множител се използва за извършване на две операции по умножение, което е пример за споделяне на ресурси.

3.2.2.3.5. Определяне пътя на данните и синтез на краен автомат

Планирането и разпределението напълно дефинират архитектурата на проекта. Behavioral Compiler създава път на данните на архитектурата чрез свързване на разпределените компоненти и регистри с проводници и мултиплексори и създава крайни автомати за управление пътя на данните.

FSM конфигурира пътя на данните по време на схематичното изпълнение, с което обезпечава, че операциите са изпълнени и данните

са правилно съхранени и направлявани по пътя си. Крайният автомат създава и контролни конструкции като **loop** (цикъл), **if** и **while** становища от поведенческото описание.

3.2.2.3.6. Изходни данни на Behavioral Compiler

Behavioral Compiler създава проект от поведенческото описание, избрана ASIC технология и набор от синтетични библиотеки. Проектът може да се синтезира в три формата:

- Файл във формат Synopsys .db, съдържащ синтезирания проект и необходимите ограничения, който се препоръчва да се компилира до нетлист на гейт ниво с други инструменти за синтез на Synopsys.
- Синтезируем RTL модел във VHDL или Verilog за компилиране до ниво гейтове, проверка или някакъв друг аспект на процеса на проект, който изисква HDL входни данни.
- RTL VHDL или Verilog файл, оптимизиран по скорост за симулация, който се препоръчва за етапа на проверка.

Behavioral Compiler има среда за графичен анализ, наречена BCView, която се използва за анализ на създадената реализация.

3.2.2.3.7. Архитектурен анализ с BCView™

BCView е интерактивна среда за графичен анализ, която помага на проектантите интуитивно да анализират и разбират резултатите от синтезът на Behavioral Compiler чрез показване на информация за техния проект и свързване на тази информация към сорс кода, използван за нейното генериране. Това позволява на проектантите да откриват проблемни части в техните първоначални описания и да ги коригират, за да подобрят производителността и/или площта.

- **Source Code Browser**

  Показва поведенческия сорс код и е свързан с другите прозорци.

- **Reservation Table**

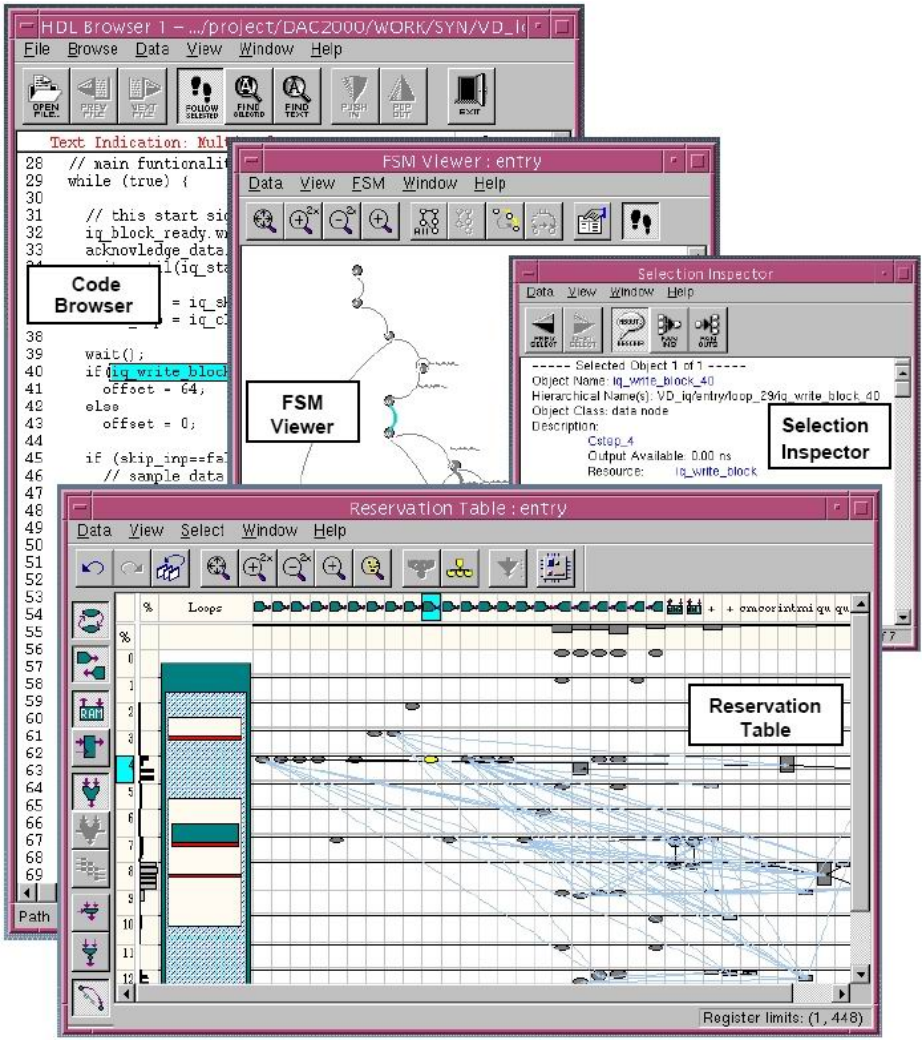
Показва разпределянето на ресурсите и зависимостите на данните между операции, регистри и вход/изход.

- **FSM Viewer**

Илюстрира създадения краен автомат в блокова диаграма и позволява на потребителя да изследва стъпка по стъпка проекта, за да провери в определено състояние какви ресурси се използват.

- **Selection Inspector**

Показва детайли за избраната информация или в Reservation Table, Source Code Browser или FSM Viewer.Например площ и скорост, име на компонента, технология и т.н.



Фиг.3.10. BCView се използват 4 четири прозореца
Source Code Browser, Reservation Table, FSM Viewer и Selection Inspector

Cocentric SystemC Compiler

SystemC Compiler синтезира хардуерни описания, написани на SystemC в нетлист на гейт ниво за реализация на интегрални схеми или в Verilog или VHDL RTL описание за FPGA синтез.

Floorplan Manager

Продуктът Floorplan Manager осигурява след-layout и след-floorplan физическа информация, която може да се използва за реоптимизиране на проектите в Design Compiler.

3.3. Инструменти за анализ

3.3.1. TimeMill

TimeMill (Timing Evaluator for Mixed-Level Logic systems) е многоетапен времеви симулатор и инструмент за проверка за CMOS и BiCMOS схемни проекти, които включват поведенчески, гейтови, ключови и транзисторни нива. TimeMill улавя грешките на проекта и времевите проблеми по време на фазата на проекта. Това дава

гъвкавостта от повече повторения за оптимизиране на проекта. Моделите на ключово и транзисторно ниво щателно симулират физическото поведение на MOS устройствата.

Преди разположение(Layout), TimeMill се използва за анализиране и проект на времевите параметри увеличавайки изпълнението на блоковете на паметта, пътищата на данните, логиката и проектите със смесени сигнали. При етапа на проверка след разположението (Layout), TimeMill проверява въздействието на паразитите върху времекритичните блокове.

3.3.2. PrimeTime

PrimeTime е статично времеви инструмент за анализ на ниво гейтове, който е изключително важна част от проекта и аналитичния поток за съвременните огромни проекти на чипове. PrimeTime задълбочено потвърждава времевото изпълнение на проекта като проверява всички възможни пътища за времеви нарушения, без да използва логическа симулация или тестови вектори. Той е подходящ за обемни, многочестотни проекти, които обединяват синтезирана логика, вградени памети и микропроцесорни ядра. Изпълнението на PrimeTime позволява осъществяването на многократни времеви анализи на проекта, докато задълбочено анализира всички критични пътища.

3.3.3. PrimeTime SI

PrimeTime SI е сигнално-интегриращо аналитично решение, което разширява възможностите на PrimeTime да идентифицира времеви и шумови проблеми, причинени от смущение и напрежетилен пад.

3.3.4. PathMill

Инструментът за времеви анализ на PathMill изследва критичния път на транзисторно ниво. PathMill използва техниките за статичен времеви анализ, за да позволи на проектанта да открие и коригира времевите нарушения. PathMill търси всички възможни пътища в проекта, изчислява закъснението и проверява времевите изисквания на всички възли.

PathMill генерира времеви модел за употреба в PrimeTime, което позволява да се ускори времевото затваряне за SoC (Система-върху-чип) проекти. Налице е също автоматична транзисторна оптимизация.

3.3.5. PathMill Plus

PathMill Plus, инструмент събрал най-доброто от PathMill, снабдява проектантите, IP доставчиците и крайните потребители с прецизно средство за проверка и характеризиране на времевите параметри на прости или сложни схеми. Процесът на характеризация на PathMill Plus се изпълнява в по-голямата си част, използвайки техниките на статичния

анализ с динамичен анализ, използван когато статичния анализ не е подходящ.

3.3.6. AMPS

AMPS® е инструмент с възможност да оптимизира едновременно по мощност, скорост и площ цифрови CMOS схеми. AMPS автоматично променя размера на транзисторите, така че да намери комбинация, която най-добре отговаря на търсените от потребителя мощност, скорост и площ, без да променя функционалността на проекта.

Приемайки нетлист като входни данни, AMPS пуска в действие статичен анализ на закъснение и динамична симулация на мощността. От симулацията и анализа AMPS изчислява размерите на транзисторите в опит да изпълни зададените цели. При това обаче AMPS не се опитва да достигне целта с една единствена стъпка, а се опитва да пристъпи към нея постепенно, като интерактивно променя размерите на устройството, след което рестартира статичния и динамичния анализ.

AMPS не се опитва да идентифицира кое е най-доброто решение, а позволява на проектанта да избере от серия изготвени итерации.

Динамичните и статичните анализи, които AMPS изпълнява, са базирани на инструментите на PowerMill и PathMill. AMPS не управлява тези инструменти, но съдържа основната технология на всеки един от тях, за да постигне по-голяма точност.

В AMPS са на разположение четири метода на изпълнение, които позволяват да се избере анализ, фокусиран към определена нужда на проекта.

По подразбиране AMPS търси в три схеми: първоначалната, пропорционално мащабно намалена схема и схема с всички транзистори, променени до минимално възможни размери.

Ако най-малката мащабна и най-малката схема са много близки до оригиналната схема, това означава, че оригиналната схема е вече много близо до минималния си размер.

AMPS приема изходните данни от SPICE файл за изходни данни като описанието на схемата. Две други помощни средства *vlog2e* и *edif2e* могат да се използват когато трябва да конвертират Verilog или EDIF нетлисти.

AMPS е проектиран да оптимизира блоковете за мощност, закъснение и площ. Някои от специфичните приложения са:

- Минимизиране на мощността/площта на проектиран блок, докато се постигне закъснението.
- Увеличаване на скоростта на проектирания блок.
- Оразмеряване на транзисторите в нетлиста, за да се постигне закъснение с минимални мощност и площ.

Ефективността на AMPS често се определя от момента, в който се ползва в цикъла на проекта, преди разположението или след

разположението. Оразмеряването на схемата, описано по-горе, обикновено се прави по време на разработването преди разположението. Другите две задачи могат да бъдат изпълнени по време, преди или след разпределението.

3.3.7. Arcadia

Arcadia е инструмент за екстрактване на съпротивления и капацитети, който е най-подходящ за дълбоко субмикронни проекти.

Arcadia най-често се използва след физическия проект за точна екстракция на паразити на устройството и вътрешните връзки и на параметри на устройството.

3.3.8. RailMill

RailMill е инструмент за симулация и анализ, който помага да се посочат и елиминират два критични проблема, свързани с проектите на ниво нанометри:

- Пад на захранващото напрежение – намаляване на схемното напрежително захранване, което може значително да намали ефективността и функционалността.
- Електромиграция (ЕМ) – миграция на метални атоми в захранващите вериги, възникващо в даден период от време, което застрашава дългосрочната надеждност на схемата.

RailMill се използва за проверка за проблеми при пад на напрежението в захранващите мрежи и ЕМ смущения на захранващите и сигналните мрежи. RailMill също може да извърши "What-if" анализ, който спестява силициева площ, чрез препоръчване на размерите на шините и предотвратяване на излизане извън размерите на проекта. Когато се използва във фазата на проекта, RailMill спомага за намаляване на времето за откриване и коригиране на грешки на ниво силиций и предотвратява скъпи премествания на силициево ниво.

3.3.9. PowerMill

PowerMill® е точен бърз схемен симулатор, който се използва при проекти за ниска мощност, чрез предоставяне на комбинация от техники за управление на мощността и диагностика. PowerMill предсказва точно консумацията на мощност, проучва алтернативи за минимизиране на мощността, планира бюджета на мощността, идентифицира нарушения на бюджета и посочва къде и как е консумирано излишно количество мощност, така че проектите може да се оптимизират.

3.3.10. PrimePower

PrimePower е динамичен инструмент за симулация и анализ на ниво гейтове, който точно анализира разсейването на мощност на проектите изградени от клетки.

PrimePower работи в две фази:

- Първо се пуска HDL симулация, за да направи файлове, съдържащи превключващата активност като функция на времето.
- После се задейства PrimePower, за да построи детайлен профил на мощността на проект, основан на свързването на схемата, превключването, капацитета на мрежата и поведенчески данни за мощността на ниво клетки в Synopsys *.db* библиотечен файл. PrimePower изчислява динамичното и статичното мощностно поведение на схемата на ниво клетки и отчита разхода на мощност на чипово, блоково и клетъчно ниво.

3.3.11. Power Compiler

Power Compiler предлага анализ на захранването и оптимизация по време на целия цикъл на проектиране, от RTL до гейт ниво.

Работейки съвместно с Design Compiler или Physical Compiler, Power Compiler осигурява едновременно времева, мощностна и площна оптимизация.

Може да се извърши анализ за мощността на проекта на:

- RTL ниво с Power Estimator, използвайки RTL симулация;
- Гейтово ниво, използвайки симулация на RTL или гейтово ниво.

Технология на Power Compiler оптимизира проекта за разход на мощност. Изчислява се средната мощност на консумация, на базата на активността на мрежите в проекта.

Оптимизация на мощността на проекта може да се извърши на RTL ниво и ниво гейтове.

Power Compiler анализира проектите за мрежова мощност на превключване, вътрешна мощност на клетките и мощност на утечка.

3.4. Инструменти за Layout

3.4.1. Apollo-II

Apollo-II е инструмент за оптимизация по време, площ, шум и мощност. Той също оптимизира разсейването на топлината и намалява консумацията на енергия и напрежителния пад. Apollo работи съвместно с Mars-Rail, който също осигурява анализ на електромиграцията, за да адресира надеждността на проекта.

3.4.2. Columbia

Columbia™ предлага високо продуктивно решение за асемблиране на чипове, което включва междублоково автоматично опроводяване, редактиране на свързването, решение за проблеми с бързодействие и сигналния интегритет. Columbia съкращава задачите за интерактивното асемблиране на чипа чрез своите автоматични, полуавтоматични и

интерактивни способности. Технологията на Columbia е основана на формите и осигурява бързо безгридово опроводяване.

Columbia има същия алгоритъм за извличане на паразити и закъснения както Astro, за да гарантира съвместимост по време из цялата йерархия на проекта. Това довежда до изключително гъвкава среда на върхово ниво на проекта, в която може да се:

- Извърши автоматично опроводяване;
- Модифицират резултатите от опроводяването, независимо дали е запазена свързаността или не;
- Защитят някои шини или критични мрежи;
- Преместват проводници, да се увеличи пространството или да се добавят буфери;
- Извърши времеви и шумов анализ на критични мрежи и шини.

3.4.3. JupiterXT

JupiterXT е важен инструмент за проектантите на чипове за запазване еднаквостта на проекта от ниво гейтове до разполагането и опроводяването в йерархична физическа среда. JupiterXT позволява да се изпълнят ограниченията за време, площ и мощност. JupiterXT се използва предимно при VDSM (Very Deep SubMicron) проектите.

3.4.4. Astro

Продуктът Astro е инструмент за физически проект, изграден на основата на Milkyway базата данни. Оптимизацията на Astro по време на реализацията едновременно се обръща към всички проблеми на реализацията – време, площ, мощност, сигнален интегритет и опроводяване. Astro предсказва времето, целостта на сигналите, целостта на захранването, площта, претовареността и възможността за опроводяване във всяка фаза от проекта, така че да подсигури изпълнението на проектите докато физическото решение се развива. Astro технологията обединява физическа оптимизация, извличане и анализ през етапите на разполагане и опроводяване с висока ефективност и точни зависимости.

3.5. Продуктите Astro

Astro се предлага в две версии **Astro Basic** и **Astro Express**, като и двете са построени на основата на Milkyway. Astro Basic съдържа технология за извличане, времеви анализ, синтез на тактуващия сигнал. Astro Express добавя физическа оптимизация, създаване/оптимизация на тактуващата верига и задълбочено опроводяване.

Astro-Xtalk™ - опцията за сигнален интегритет на Astro, дава задълбочено решение на проблеми на сигналния интегритет, което съществува за UDSM. Използвайки същия алгоритъм за бързодействие, извличане и оптимизация, който е използван за разположението

(Layout), Astro-Xtalk осигурява ранна видимост в сигналната цялост. Проектантите могат от рано да вземат мерки за предотвратяване или коригиране на потенциални проблеми със сигналния интегритет по време на физическия проект, а не след това.

Astro-Rail™ - опцията за сигнален интегритет на Astro, осигурява цялостно решение за проблеми с консумацията на мощност, пад на напрежението и електромиграция при UDSM (Ultra Deep SubMicron) проекти. Astro-Rail може да се използва за да се изчисли общата консумация на мощност и да се провери пада на напрежението и нарушения в електромиграцията на дадена схема веднага щом цялото опроводяване е приключило по време на проекта. Това позволява да се открият потенциални проблеми, преди да се извърши детайлно разполагане и опроводяване. След като разполагането и опроводяването е завършено и по време на етапа на проверка на физическия проект, се използва Astro-Rail, за да се провери функционалността и бързодействието на проекта.

3.5.1. Floorplan Compiler

Floorplan Compiler е инструмент за планиране на проекта. Чрез доставяне на информация за проекта по време на синтеза, Floorplan Compiler намалява времето на цикъла на проекта и подобрява качеството на резултатите.

Power Network Analysis (Анализ на захранващата мрежа) е характеристика на инструмента Floorplan Compiler, която позволява да се анализира пада на напрежение и електромиграционните ефекти на захранващата мрежа. За избрана йерархия, Power Network Analysis осигурява статичен анализ за утечка на захранващата верига и опроводяваните проекти, и what-if анализ на виртуални топологични и промени в захранващите източници.

3.5.2. Mars-Xtalk

Mars-Xtalk осигурява задълбочено и цялостно решение на проблемите на сигналния интегритет на VDSM(Very Deep SubMicron) SoC (Системи-върху-чип) проекти.

Способности на Mars-Xtalk :

- Твърди стратегии за съкращаване на преплитанията в сигналите (crosstalk).
- Спазване на различни crosstalk ограничения.
- Точен и бърз crosstalk анализ, базиран на Apollo LPE.
- Задълбочен crosstalk отчет.
- Опроводяване за избягване на смущенията на сигналите с възможности за предотвратяване и поправяне .

3.5.3. Mars-Rail

Mars-Rail е инструмент за анализ на мощността и утечката, за оптимизация на VDSM (Very Deep SubMicron) проекти.

Ключови характеристики на Mars-Rail:

- Анализ за утечка на ток;
- Електромиграционен утечен анализ;
- Разполагане на елементите, оптимизирайки мощността и температурата;
- Разполгане на елементите за избягване на утечки;
- Минимизиране на закъсненията в клетките.

3.5.4. NanoSim

NanoSim е инструмент за схемотехнична симулация на транзисторно ниво и анализ на аналогови, цифрови и смесено сигнални проекти. NanoSim комбинира най-добрите технологии за симулация от TimeMill и PowerMill, и комбинира времеви анализ, анализ на мощността и диагностика в един инструмент. NanoSim е обратно-съвместим с TimeMill и PowerMill, приема същите нетлист файлове и дава същите висококачествени резултати по точност, изпълнение и капацитет.

3.5.5. Saturn

Saturn™ снабдява проектантите с ефективни средства за елиминиране на логически и физически повторения на проекта чрез трансформиране на логиката на проекта и разположението. Чрез извършване на повтарящо се оразмеряване и логическо реструктуриране по време на поставяне на елементите, след поставянето и дори след опроводяването, бързодействието, площта и мощността на проекта могат да бъдат оптимизирани с Saturn.

3.5.6. Proteus

Proteus Progen и **Prospector** оформят мощна среда за изпълнение на приблизителна корекция, изграждаща модели за коригиране и анализ за разположението на елементите в интегралните схеми.

3.5.7. HERCULES

HERCULES е мощен комплект инструменти, които проверяват разполагането на интегралните схеми. HERCULES може да провери правилата на проекта за разположение, да изпълни проверка на електрическите правила, да извлече структурите на разположението и да сравни извлечените структури на разположението спрямо оригиналния нетлист на проекта.

3.5.8. Star-RCXT

Star-RCXT е инструмент, който извлича паразити от свързани бази данни, които представляват Layout проекти на интегрални схеми. Star-RCXT генерира нетлисти, които могат да бъдат използвани за провеждане на времеви, тактов или шумов анализ или за извличане на паразити като съпротивления, капацитети или индуктивности от базата данни на Layout проекта. Star-RCXT може да извлече информация за съпротивлението и капацитета от напълно опроводен блок на проекта.

Star-RCXT може да бъде използван по всяко време през цикъла на физическия проект, за да извлече точно паразитите.

3.5.9. Star-MTB

Star-MTB автоматично извлича всички необходими данни от описанието и създава всички симулационни, времеви, синтезисни и мощностни модели, изисквани от проектанта на логиката.

3.5.10. Star-SimXT™

Star-SimXT™е симулатор, който се справя с милиони елементи на ултра дълбоки субмикрони (0.10 микрона) и по-долу. Star-SimXT осъществява анализи след разположението, включително времеви, мощностни, на утечка, по такт и за смущения.

3.5.11. Cosmos

CosmosSE е схемно-базирана среда за симулация и анализ. В комбинация с инструментите за симулация HSPICE, Star-SimXT и Saber, CosmosSE осигурява цялостно решение за бързо проектиране на аналогови, смесеносигнални и произволни високоскоростни цифрови схеми.

CosmosLE използва схемно управлявано разположение и опроводяване на схемите и е перфектно допълнение на CosmosSE схемно-базирани симулации и анализи.

CosmosScope е графичен анализатор на времедиаграми, който помага на проектантите да анализират изпълнението на проекта и да гарантират качеството му.

Enterprise е редактор на физическото разположение, който увеличава продуктивността на проектантите по време на създаването и редактирането на сложни IC проекти.

3.6. Инструменти за проверка

3.6.1. Formality

Formality е приложение, което използва формални техники за доказване или опровергаване на функционалната еквивалентност на два проекта. Например Formality може да се използва за да се сравнят нетлист на гейт ниво и неговия източник на RTL ниво или към

модифицирана версия на този нетлист на гейт ниво. След сравнението, Formality отчита дали двата проекта са функционално еквивалентни. Така се проверява дали след реализацията проектът ще функционира според очакванията.

Техниките, които Formality използва са статични и не изискват симулационни вектори. Следователно проектантите трябва само да предоставят функционално верни или “златни”проекти (наречени справочни проекти), и модифицирана версия на проекта (наречена реализация, или имплементация). Чрез сравняване на реализирания проект към справочния проект, може да се определи, дали реализирания проект е функционално еквивалентен със справочния проект.

3.6.2. Продукти на Test Compiler

Продуктите на Synopsys за тестване комбинират тестов синтез, автоматично създаване на тестови модели (automatic test pattern generation ATPG), симулация на грешки и тест мениджмънт, за да автоматизират проекта за тест (design-for-test DFT). Тези способности помагат на проектантските екипи бързо да изкарат на пазара проекти с висока тестопригодност.

3.6.2.1. TetraMAX

TetraMAX е инструмент за създаване на висикоскоростен, високо капацитетен автоматичен тестов модел (ATPG). Той може да генерира тестови модели, които максимизират тестовото покритие, използвайки минимален брой от тестови вектори за широк спектър от типове проекти.

3.6.2.2. BSD Compiler

BSD Compiler е автоматизиран инструмент за синтез и проверка на логиката за гранично изследване (boundary scan) в ASIC и IC в рамките на средата за синтез на Design Compiler™. BSD Compiler синтезира гранично изследване от RTL описанието на потребителя, използвайки компонентите на DesignWare® JTAG. След синтеза мощна и задълбочена проверка за съвместимост в BSD Compiler тества граничната логика за съвместимост със стандарта IEEE 1149.1. BSD Compiler автоматично създава файл на език за описание на гранично сканиране (boundary scan description language BSDL) за тест на ниво платка и изготвя функционални и постояннотокови параметрични вектори за производствени тестове.

3.6.2.3. Vera

Vera® е изключително важна част от платформата на Synopsys за проверка. Vera предлага автоматизирани тестбенчове за модулна, блокова и цялостна системна проверка. Системата за автоматизирана тестова площадка на Vera е базирана на OpenVera™, интуитивен

обектно-ориентиран език за програмиране на високо ниво, разработен специално да задоволи изискванията за функционална проверка.

3.6.2.4. Magellan

Magellan е хибриден RTL продукт за формална проверка, който позволява на инженерите да намерят бързо трудноотриваеми грешки,което води до съкращаване на цикъла на проверка. Magellan е с вграден VCS™ двигател за симулиране, за да провери качествата на големи и сложни проекти.

3.7. Инструменти за многократно използване и създаване на IP

3.7.1. DesignWare

Продуктите от семейството на DesignWare обхващат инструменти, които позволяват многократно използване на проекта и балансиране на възможностите на инструментите на Synopsys за синтез, симулация и тестване.

Семейството от решения за интелектуалната собственост (IP) на DesignWare® предоставя на проектантите многостранен набор от интелектуална собственост за реализация и проверка, включително на компоненти за високоскоростен пренос на данни, памети, микроконтролери, микропроцесори и др.

Тъй като библиотеката на DesignWare е тясно интегрирана с инструментите за синтез на Synopsys, като Design Compiler и Physical Compiler, инструментите за синтез автоматично избират правилната архитектура с най-добра оптимизация на скорост и площ.

Интелектуалната собственост за проверка на DesignWare Verification Library се възползва от техниките за генериране на произволен ограничен тест, значително подобрявайки продуктивността на тест-инженерите и драстично намалява риска от неоткрити функционални грешки. Всяка интелектуална собственост на DesignWare за проверка включва възможности за автоматично проверяване за грешки, което помага на тест-инженерите да създадат бързо по-точни и по-ефективни среди за проверка.

DesignWare Cores предоставя на системните проектанти силициево утвърдени, цифрови и аналогови интелектуални собственост.

3.7.2. DesignWare Developer

DesignWare Developer е софтуерен инструмент, който позволява на проектантите да създадат DesignWare компоненти от техните собствени данни на проекта. Компонентите на DesignWare включени в проект са обект на оптимизация на високо ниво, извършвана от софтуера за синтез.

DesignWare Developer е реализиран като сбор от команди и променливи за *dc_shell*. Основните стъпки за създаване на DesignWare компоненти са:

1. Създаване и проверка на проекта. Стилът на описание може да варира от специфичен за дадена технология нетлист, до пълно йерархично HDL описание на параметризиран и оптимизиран проект.
2. Прилагане на директиви за моделиране, компилация и лицензиране. Моделни директиви се прилагат за контрол на моделирането на проекта от Design Compiler. Компилационните директиви контролират как Design Compiler оптимизира създадения компонент. Лицензионните директиви защитават проекта от нерегламентирана употреба.
3. Поставяне на проекта в библиотеката на проекта. Библиотеката на проекта е UNIX или WinNT директория, която съдържа елементите на проекта, преведени в различни междинни формати, директно употребяеми от инструментите на Synopsys.
4. Написване на синтетичен библиотечен код. Синтетичното библиотечно описание установява връзките на трансформация при разработване на DesignWare компонент. Проектантите могат просто да свържат проекта към съществуващ синтетичен модул или да създадат нов синтетичен модул.
5. Компилиране на синтетичния библиотечен код.
6. Проверка на новия DesignWare компонент. Изпълняват се прости тестове за цялост на компонента. Проектанта трябва да изработи тестове, за да провери, дали реализациите му са избрани автоматично от инструментите, когато това се очаква. Дали проектите са защитени от нерегламентиран достъп и т.н.
7. Пакетиране на DesignWare компоненти за дистрибуция - в случай че проектантът възнамерява да разпространи компоненти си на трети лица.

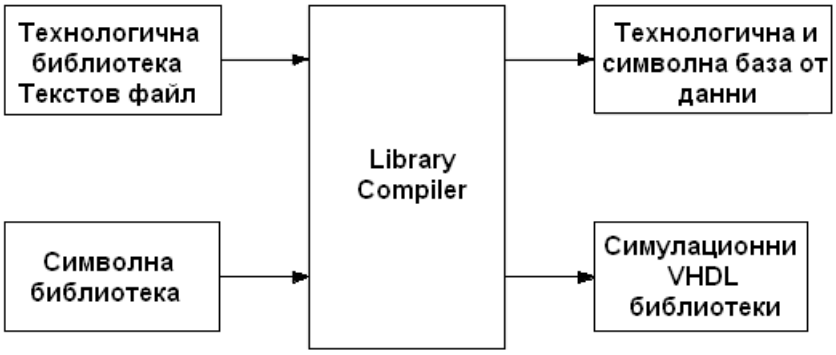
3.7.2.1. Milkyway

Milkyway е широко използвана база данни за инструментите за проект на интегрални схеми позволяваща безпроблемна взаимо-оперативност сред инструментите за реализация и анализ. Тъй като данните се съхраняват в единична база данни с бърз достъп за четене и писане, продължителни периоди от време за превод между инструменти и несъвместимо превеждане на променливи данни са предотвратени.

3.7.2.2. Library Compiler

Library Compiler помага на проектантите лесно да създават и разработват SoC (System-on-Chip) технологии за проверка по време на синтез, симулация, тестване, мощност и др.

Library Compiler автоматично проверява технологичния файл за завършеност и съвместимост, после го обединява във формати, използвани за логически синтез, тестов синтез, симулация, мощност, статично времеви анализ, физически инструменти за синтез и функционална проверка.



Фиг. 3.11. Структура на Library Compiler

Library Compiler чете входното описание на технологичната библиотека от текстов файл и компилира описанието във вътрешен ***.db** (database) формат или във VHDL библиотека.

Компилираната база данни се ползва от инструментите за синтез. VHDL библиотеките се ползват от VHDL симулационните инструменти.

Тези синтезисни библиотеки се състоят от технологични библиотеки, физически библиотеки и символни библиотеки, които съдържат типовете информация, изисквана, за да се опишат ASIC компонентите: технически и физически характеристики и схематични символи.

фиг.3.11 Структура на Library Compiler

Фиг. 3.11. Структора на Library Compiler

- Технологичната информация съдържа характеристиките и функциите на всички компоненти от ASIC библиотеката. Тя е съставена от площ, време, функции и др. Инструментите на Synopsys за проектиране използват тази информация по време на синтеза.
- Символната информация се състои от графични символи, представящи всеки ASIC компонент, в това число и специални символи като граници на чертежа и преходни съединения.

Технологични библиотеки

Технологичната библиотека е текстов файл, съдържащ четири вида информация за дадена ASIC технология.

Структурна информация

Структурната информация описва съединенията на всяка клетка с външното й обкръжение, включително с други клетки, с магистрали и изводи.

Функционална информация

Технологичната библиотека трябва точно да определи логическата функция на всеки извод на всяка клетка, за да може Design Compiler логически да свърже проекта с използваната ASIC технология. Клетките,

които нямат функционално описание в технологичната библиотека остават несвързани по време на оптимизацията.

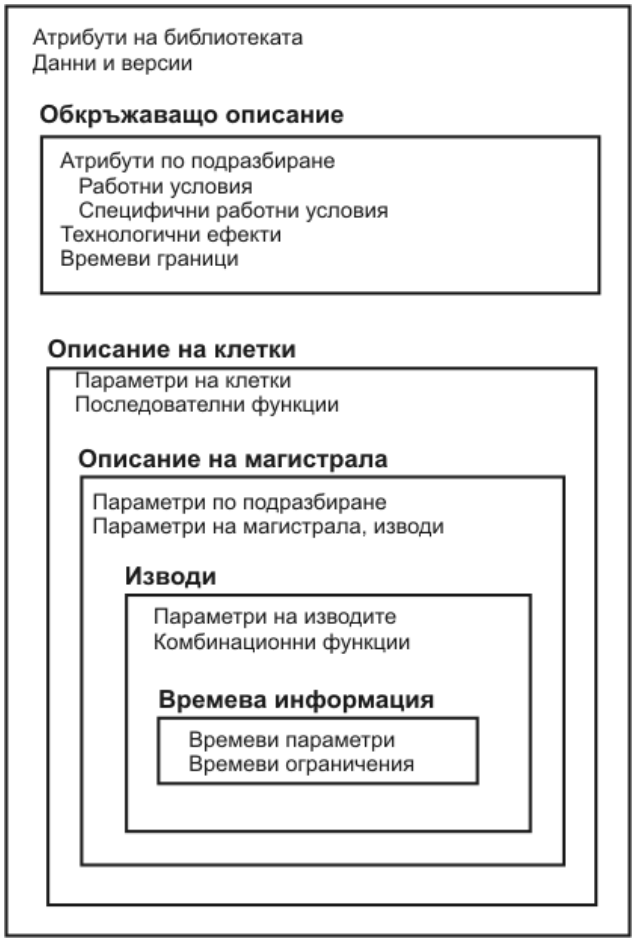
Времева информация

За точен времеви анализ и времева оптимизация на проекта е необходимо всяка клетка, включена в технологичната библиотека, да съдържа параметри за съответните времеви отношения и изчислените закъснения.

Информация за обкръжаващата среда

Обкръжаващата информация описва процеса на производство, операционната температура, напрежителните изменения и топологията.

На фиг. 3.12 е показана структурата на технологичната библиотека. Тя е съставена от описание на клетки и обкръжаващо описание:



Фиг. 3.12. Структура на технологична библиотека

- Описанието на клетки дефинира всеки компонент от дадената ASIC технология, включително клетка, магистрала, извод, област, функция и времева информация.
- Обкръжаващото описание съдържа информация за ASIC технология, която не е уникална за даден компонент. Също така съдържа информация за резултата от работните състояния на технологията, стойностите на компонентите от закъснителните уравнения, статистическа информация за междинните пресмятания и атрибутите на клетката по подразбиране.

Символни библиотеки

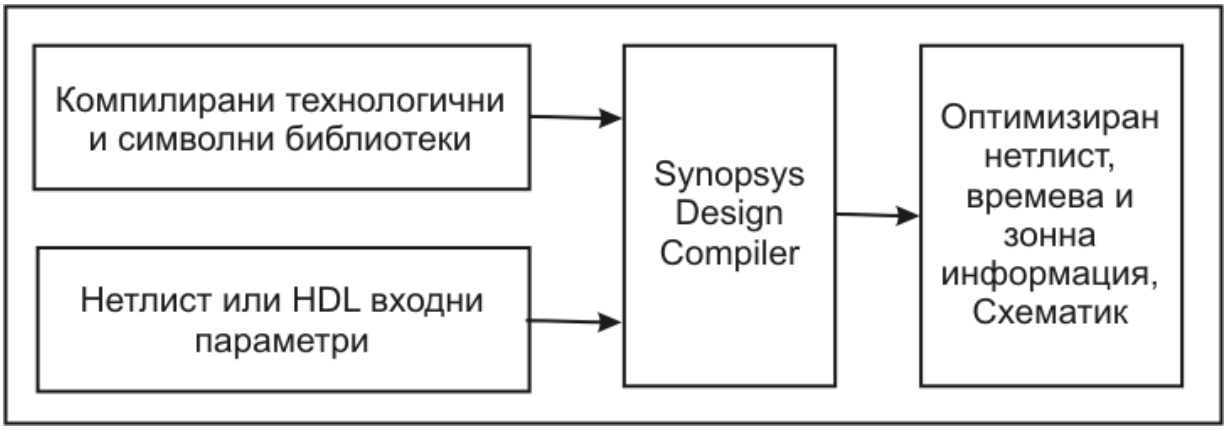
За да може Design Analyzer да изобрази на екрана даден проект, е необходимо всеки елемент от технологичната ASIC библиотека да има графично представяне – схемен символ в символната библиотека. Освен това няколко специални символа в символната библиотека

описват връзки между чертежите на проекта, шаблони на границата на листа и логически символи по подразбиране.

Компилирани библиотеки и Design Compiler

Компилираните технологична и символна библиотеки могат да се използват с Design Compiler за генериране на схема или проект (фиг. 3.13).

Компилираните библиотеки са независими от работната станция. Библиотеките могат да се прехвърлят между различни хардуерни платформи, ако файловете са третирани като двоични.

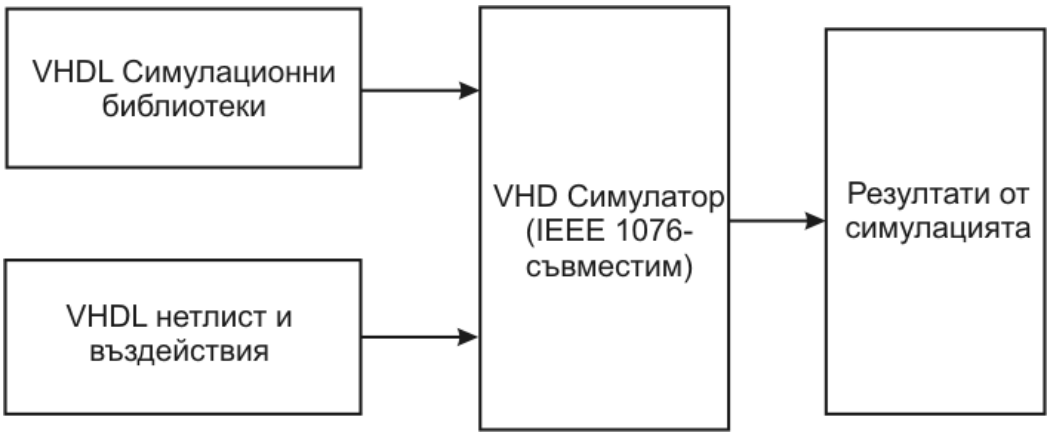


Фиг. 3.13. Използване на компилираните библиотеки с Design Compiler

VHDL симулационни библиотеки

Library Compiler може също така да генерира VHDL библиотеки, съдържащи функционална информация, необходима за симулацията.

След създаване на VHDL симулационната библиотека, VHDL симулаторът съвместно с информацията от нетлиста, извършва симулацията на проекта (фиг. 3.14).



Фиг. 3.14 Използване на VHDL библиотеки с VHDL симулатор

3.8. Други инструменти

3.8.1. SaberDesigner

SaberDesigner® софтуерът симулира физичните ефекти в различните инженерни области (хидравлични, електрически, електронни, механически и др.). Saber софтуерът се използва в автоматиката, аеронавтиката, енергийната и IC индустрии за симулиране и анализиране на системи, подсистеми и компоненти за намаляване необходимостта от прототипи.

Saber е смесено-сигнален, смесено-технологичен поведенчески инструмент за симулация. Saber се използва за извършване на времеви, честотни, статистически и надеждностни анализи на проектите от архитектурно до транзисторно ниво. Saber е математически двигател, който решава мрежата от уравнения, представени от моделите и техните взаимовръзки в схема или система.

Saber може да симулира проекти, които включват модели написани на Verilog или VHDL. Saber може да бъде свързан с популярни цифрови симулатори като Cadence Verilog-XL, ModelSim и ModelSim Plus от Model Technology и симулатора Innoveda Fusion. Тъй като Saber е по своята същност смесено-сигнален симулатор, има поддръжка на ко-симулации с други симулатори.

SaberSketch е интерактивен графичен интерфейс, който се използва за импортване на схеми, анализиране на проекти и контролиране на симулатора. След създаването на аналогови, цифрови или смесено-технологични проекти SaberSketch доставя голяма библиотека от символи и модели за симулация. SaberSketch също включва инструмент за разглеждане на сигналите на всеки схемен възел, на всяко йерархично ниво, направо върху схемата.

SaberScope е мощен графичен инструмент за разглеждане и измерване на резултатите от симулацията на смесени сигнали. SaberScope може да бъде използван за разглеждане на сигнали и параметри, дълбоко в системата или модела. При нужда от разглеждане на допълнителни сигнали, те могат да се извлекат без да се пуска отново симулацията. Тази функция спестява симулационно време и прави диагностицирането на схемата по-лесно.

Saber HDL симулаторът може да симулира всяка комбинация на смесеносигнални HDL и SPICE описания. Saber HDL може да се използва през целия цикъл на проектиране – от архитектурно изследване на електрическата система до проверка на транзисторните блокове в контекста на цялата система.

SaberHarness се използва за проект и симулация на електрически проводящи системи или снопове от проводници, конектори и не електрически компоненти. SaberHarness осигурява инструменти за изработване на отчети, списъци на частите и варианти на проекта за

използване и в механично разполагане на елементите и в производството. SaberHarness осигурява разширена информация за проводниците (кабелите), включително диаметър, цвят и номер на частите за улесняване избора на необходимия проводник за производство.

Testify позволява на проектантите да използват Saber симулатора за развитие и оценяване на тестове, използвани за откриване на дефекти условия на схемната платка.

iQBus е интегриран пакет инструменти за проект, симулация и анализ за инженери в транспортната индустрия. iQBus е инструмент за проектиране на електрически архитектури, електромеханични системи, хибридни системи, електрически возила, магистрални системи (CAN, TTP, LIN, MOST, D2B), захранващи системи и други приложения на транспортния проект.

SaberRT е интерфейс към Saber симулатор, позволяващ проектите да бъдат симулирани интерактивно в реално време. SaberRT може да бъде извикан от SaberSketch, SaberHarness или iQBus или като самостоятелно приложение, използвано като инструмент за проверка на проекта или като интерактивен тестбенч на проекта.

3.8.2. Вътрешен език за хардуерно описание

MAST е език за пълно описание на хардуера за смесени сигнали за Saber. Богатия набор от конструкции за моделиране налице в MAST позволява моделирането на аналогово поведение и подтиквано от събитието поведение.

MAST има способността да моделира не-електрически технологии, като термични, механични, хидравлични и оптични системи. Тези технологии могат да бъдат описани на своите родни единици и резултатите от симулатора ще запазят единиците на технологията, в които се моделира.

MAST има добре дефиниран интерфейс, който позволява достъп до функциите на C++ и FORTRAN от рамките на модела. MAST също притежава много развити характеристики, които позволяват да се задава информация, като параметри за натоварване и статистически вариации на параметрите.

ГЛАВА 4

Език за описание на хардуер VHDL

Въведение

VHDL е език за описание на хардуер, предназначен за документиране и моделиране на широк спектър цифрови схеми – от малки чипове до големи системи. Той може да се използва за моделиране на цели цифрови системи при произволно ниво на абстракция – от архитектурно до ниво елемент.

4.1. Историческо развитие на VHDL

В първоначалния си вариант езикът е създаден през 1980г. по инициатива на министерството на отбраната на САЩ, свързана с проект за разработка на интегрални схеми с много високо бързодействие (*VHSIC - Very High Speed Integrated Circuits*). Този проект дава и името на VHDL (***V****HSIC* ***H****ardware* ***D****escription* ***L****anguage*). Поради изключителната нужда в индустрията от стандартизиран език за описание на хардуер, VHDL е избран, а по-късно (1987г.) утвърден като стандарт от IEEE под наименованието IEEE Std 1076-1987.

През 1993г. езикът е допълнен с редица пояснения и нови компоненти, като групи и поделени променливи (*shared variables*). Тази, последна за сега, редакция на стандарта е регистрирана като IEEE Std 1076-1993. През 1996г. на пазара излизат програмни средства за симулация и синтез, съобразени с горепосочения стандарт, което дава възможност на проектантите да го приложат съвместно с *top-down* методологията. Под наименованието IEEE 1076.3, част от стандарта става цял VHDL пакет, предназначен за използване от автоматизирани системи за синтез. Това чувствително повишава преносимостта на проектите и реализацията им в различни развойни среди. Друга част от IEEE Std 1076-1993 – IEEE 1076.4 (VITAL) – третира моделирането на библиотеки на VHDL, като значително улеснява производителите и проектантите на ASIC и FPGA.

VHDL се използва предимно за описание на цифрови схеми, като напоследък намира все по-широко приложение и в аналоговите схеми (VHDL-AMS).

4.2. Интерфейсно и архитектурно тяло – ENTITY и ARCHITECTURE

Интерфейсното и архитектурното тела са първични блокове във VHDL. Ако се използва аналогията с ИС, интерфейсът отговаря на описанието на изводите (pins) на интегралната схема, докато архитектурата описва функцията й.

Интерфейсът съдържа списък на портовете, като на всеки порт се задават име, посока и тип на данните.

Пример: **ENTITY entity_name IS**
[PORT(port list)]
END entity_name;

Интерфейс без входно-изходни изводи (ports) представлява затворена система, която няма връзка с външния свят. Типът на изводите може да бъде: *IN, OUT, INOUT, BUFFER, LINKAGE.*

Архитектурно тяло се изисква за всеки интерфейс във VHDL дизайна. То описва взаимовръзката между портовете и интерфейса. VHDL позволява за един интерфейс да се дефинират много архитектури.

Основната форма на архитектурата е :

```
ARCHITECTURE body_name OF entity_name IS
-- declarative statements
BEGIN
--statements to describe function of architecture
END body_name;
```

4.3. Типове данни

Всички обекти във VHDL (порт, сигнал, променлива и т. н.) имат тип. Той определя набор от стойности, които обектът може да заеме, а също така и операциите, които могат да бъдат извършени върху обекти от този тип. Обектите във VHDL са сигнали, променливи величини или константи. Те ще бъдат обсъдени по-късно.

След като веднъж обектът е бил деклариран като някакъв тип, върху него могат да бъдат извършвани операции според ограниченията, наложени от декларацията на типа. Например обектите от тип integer (цяло число) могат да приемат само стойностите (..., -2, -1, 0, 1, 2, ...). Операциите върху тези обекти са ограничени до такива, дефинирани за съответния тип. Например, за целочислените данни валидна операция е събиране.

VHDL има четири главни класа от типове данни: скаларни, съставни, тип ACCESS и тип FILE.

Скаларният клас включва всички прости типове данни като INTEGER и REAL (реални). Обектите от скаларните типове имат стойности, които са прости величини. Съставните типове позволяват на обектите да имат повече от една стойност и включват редове и архиви. ACCESS типовете са еквивалентни на указателите от обикновените езици за програмиране, докато типовете FILE осигуряват достъп до файловете.

4.3.1. Скаларни типове данни

Обектите, декларирани като скаларен тип могат да имат най-много една стойност по едно и също време. Скаларните типове включват целочислен, реален, физически и изброим тип.

Нови VHDL типове се дефинират при самата декларацията на типа. Обявяването на типа определя неговото име и обхвата му. Синтаксисът е следния:

TYPE *име на типа* IS *бележка за типа*;

Тук *име на типа* е името на новия тип, а *бележка за типа* е дефиницията му. Типът може да дефинира подтип, като се използва построението RANGE.

TYPE *име на типа* IS RANGE *израз* TO *израз*
TYPE *име на типа* IS RANGE *израз* DOWNTO *израз*

Под името **израз** следва да се разбират числовите стойности, чрез които се определят границите, в които съответния тип, може да варира. Чрез конструкцията "*израз* TO *израз*", се дефинира диапазона на изменение, във възходящ ред, а чрез конструкцията "*израз* DOWNTO *израз*", в низходящ.

4.3.2. Целочислени типове

Целочислените типове определят ред от цели стойности, които декларираният обект от този тип може да приеме. VHDL определя минималния обхват на целочисления тип да бъде -2,147,483,647 до +2,147,483,647. Всички VHDL изпълнения са свободни да използват целочислен ред, който е по-дълъг от този. Примери за други дефинирани от потребителя целочислени типове могат да бъдат:

TYPE IntegerByte IS RANGE 0 TO 255;
TYPE SignedByte IS RANGE -127 TO 128;

4.3.3. Реални типове данни

Обектите от реален тип се използват за числа с плаваща запетая. Реалното число се различава от цялото с присъствието на десетична точка. Минималния обхват на реалните числа е определен в стандартния пакет и е от -1.0E+38 до +1.0E+38.

4.3.4. Изброими типове

Изброимият тип дефинира стойности чрез изреждането им в подредени списъци. Елементите в списъка могат да бъдат или идентификатори или буквени символи. Вътре в простия изброим тип идентификаторите или буквените символи трябва да бъдат уникални, следователно те могат да бъдат използвани отново с различни изброими типове.

Изброимите типове са приложими, тъй като допускат даден тип да бъде дефиниран, за да представи точно стойностите, изисквани за специфичния оператор. Например:

TYPE FourBit IS ('X', '0', '1', 'Z');

TYPE SnookerBall IS (White, Red, Yellow, Brown, Pink, Black);

В първия случай обектът FourBit е деклариран и представя израза за логическа симулация на четири стойности. Този тип на деклариране използва буквени символи (единични символи, отделени с единични кавички). Чрез този начин на означение, ясно биха могли да се различават буквените символи ‘0’ и ‘1’ от целите числа 0 и 1. Не е необходимо, символите X и Z да бъдат буквени символи, тъй като те не са представени от нито един от другите типове, но кавичките се използват за уеднаквяване.

Във втория случай, абстрактният нов обект SnookerBall е деклариран в диапазон на цветовете, в които могат да бъдат, например топките върху една билярдна маса. Така всеки обект деклариран като тип SnookerBall би бил в състояние да заеме една от стойностите: White, Red, Yellow, Brown, Pink или Black.

Тези стойности вътре в изброимия тип имат присъединен позиционен номер. Позиционният номер на мястото е определен от ляво на дясно във възходящ ред, като се започне от 0. Следователно в типа SnookerBall, White има номер 0, Red има номер 1, Yellow позиционен номер 2 и т. н.

Позиционните номера дефинират числеността подреждане на стойностите. Това дава възможност да се използват изрази от вида Red < Yellow. По-късно, когато бъдат обсъдени атрибутите, ще бъде показано как най-пълно може да се използва подредбата на позициите.

VHDL има някои предефинирани изброими типове:

```
TYPE bit IS (‘0’, ‘1’)
TYPE boolean IS (false, true);;
TYPE severity_level IS (note, warning, error, failure);
TYPE character IS (‘a’, ‘b’, .....);
```

4.3.5. Физически типове

Физическите типове са числени типове данни, които се използват, за да описват физически величини като време, дължина, напрежение и др. Физическият тип данни се базира на основния елемент. Последователните елементи след това са дефинирани чрез умножаване на този елемент. Най-малката единица, която може да бъде представена е един основен елемент. Например:

```
TYPE ImperialMeasure IS RANGE 0 TO 633600
    UNITS
        inch;
        foot = 12 inch;
        yard = 3 foot;
        chain = 22 yard;
        furlong = 10 chain;
        mile = 8 furlong;
    END UNITS;
```

Дефиницията на типа задава името на типа, в този случай ImperialMeasure и обхвата му, от 0 до 10 мили. RANGE ограничава минималните и максималните стойности, които обектите от този тип могат да заемат и е изразен в основни единици. Чрез основна единица може да бъде изразен повече от един елемент, например един ярд може да бъде деклариран като 36 инча.

VHDL съдържа само един предефиниран физически тип – TIME:

```
TYPE Time IS RANGE <зависим от инструментариума>
    UNITS
        fs;                 --femosecond
        fs = 1000 fs;       --picosecond
        ns = 1000 ps;       --nanosecond
        us = 1000 ps;       --microsecond
        ms = 1000 us;       --millisecond
        sec = 1000 ms;      --second
        min = 60 sec;       --minute
        hr = 60 min;        --hour
    END UNITS;
```

Обхватът на времето е зависим от инструментариума, но е най-малко обхвата на целочисления тип в базови времеви единици, т.е. от -2,147,483,647 до +2,147,483,647.

4.3.6. Съставни типове данни

Съставните типове данни позволяват на обекта да има повече от една стойност. VHDL поддържа два съставни типа – масиви и записи.

4.3.6.1. Масиви

Масивът групира заедно като един обект определен брой елементи от един и същи тип. Елементите на масива могат да бъдат от всеки VHDL тип, дори масиви. Отделните елементи на масива са достъпни чрез стойността на техния индекс, който може да бъде единична стойност в случай на едномерен масив или съставен индекс в случай на многомерни масиви.

Пример за дефиниране на масив е:

```
TYPE DataBus IS ARRAY (0 TO 31) OF BIT;
TYPE Scores IS ARRY (11 DOWNTO 0) OF INTEGER;
```

Първият от тези примери декларира тип DataBase, който е масив от 32 елемента, всеки от които е тип BIT. Вторият декларира масив от 12 елемента от целочислен тип.

На всеки един от елементите могат да бъдат приписвани стойности. Например, ако сигналът Total е бил деклариран от тип score, ще бъдат валидни следващите операции:

```
SIGNAL Total : Scores;
SIGNAL Tmp : INTEGER;
```

```
[...]
Total (11) <= 356;
Total(0) <= 1;
Tmp <= Total(11);
[...]
```

Типът на индекса на масива може също така да бъде и изброим тип. Например:

```
TYPE SnookerBall IS (White, Red, Yellow, Brown, Pink, Black);
TYPE BallsPotted IS ARRAY (SnookerBall) OF NATURAL;
```

Дефинирането на типа на масива може да бъде ограничено или неограничено. При ограничено дефиниране, границите на индексите се установяват, когато е дефиниран типът, докато в случай на неограничено, границите на индексите на матрицата се установяват по-късно.

VHDL има два предефинирани типа масиви и двата неограничени:

```
TYPE String IS ARRAY (POSITIVE RANGE <>) OF CHARACTER;
TYPE Bit_Vector IS ARRAY (NATURAL RANGE <>) OF Bit;
```

Типът String е неограничен масив от символи, който се използва за създаване на текст за отпечатване. Типът BIT_VECTOR е неограничен масив от елементи от тип BIT, използван за моделиране на пренасяне на отделни битове. Неограничените масиви са полезни като типове на аргументи от подпрограми и интерфейсни части. Това дава възможност на една единствена дефиниция на подпрограма/интерфейсна част да оперира върху масиви с различна големина. В случая на BIT_VECTOR големината на масива е ограничена от дефиницията на натуралния тип. Например, моделът на generic nand gate може да бъде написан, като се използват неограничени матрици:

```
ENTITY Nandgen IS
      PORT ( A,B : IN BIT_VECTOR;
                Y   : OUT BIT_VECTOR);
END NandGen;

ARCHITECTURE Simple OF NandGen IS
BEGIN
      Y <= NOT (A AND B);
END Simple;
```

4.3.6.2. Масиви и константи

Константите могат да бъдат използвани за инициализиране на всички елементи в масива. Например:

```
TYPE Word IS ARRAY (0 TO 7) OF BIT;
CONSTANT EmptyWord : Word := ('0', '0', '0', '0', '0', '0', '0', '0');

SIGNAL DataBus : Word;
DataBus <= EmptyWord;
```

4.3.6.3. Записи

Записите са набор от наименовани елементи, всеки от които може да бъде от различен тип. Те се използват, за да моделират структури от данни, които се състоят от тясно свързани части от различен тип. Например, следващата декларация на запис декларира нов тип Date (дата). Като условие датата съдържа три полета: ден, месец и година, като всяко от тях е от различен тип.

```
TYPE MonthName IS (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);
TYPE Date IS RECORD
    Day : INTEGER RANGE 1 TO 31;
    Month : MonthName;
    Year : INTEGER RANGE 1950 TO 2001;
END RECORD;
```

Всяко поле от записа представлява уникална запаметяваща област, от която той може да бъде прочетен и да му се присвояват данни от подходящ тип. Всяко поле може да бъде споменато, като се използва име на обект от тип запис, следвано от точка и след това името на полето. Например:

```
SIGNAL NextBirthday : Date;
SIGNAL ChristmasDay : Date;
NextBirthday.Day <= 25;
NextBirthday.Month <=Dec;
NextBirthday.Year <= 1994;
ChristmasDay <= NextBirthday;;
```

4.3.7. Типове ACCESS

Обектите от тип ACCESS могат да бъдат наречени указатели на програмен език. Те осигуряват динамично определяне на паметта, полезно при употреба на поведенческо моделиране от високо ниво. Тези типове няма да бъдат разглеждани.

4.3.8. Типове FILE

Типовете и обектите FILE осигуряват начин на VHDL проекта да общува с външния свят. VHDL конструкцията може да чете или пише в повече от един файл. VHDL файлът е винаги от специален тип и може да съдържа само информация за този тип. Например:

```
TYPE IntegerFile IS FILE OF Integer;
```

По късно в този курс ще бъдат разгледани файловете INPUT и OUTPUT.

4.3.9. Подтипове

Често обектите заемат ограничен набор от стойности за типа към които са дефинирани. В някои случай е подходящо да се работи с

подтипове, покриващи диапазона от работни стойности. Обектите декларирани като подтип могат да бъдат само вътре в областта, покрита от дефинирането.

В допълнение, за да се осигури по-стриктна проверка на стойностите на обектите спрямо обхвата, се изисква подтиповете да дефинират само разрешени типове сигнали (разрешените типове сигнали ще бъдат разгледани по-късно). Във VHDL има два предефинирани подтипа: положителни и натурални.

```
SUBTYPE Positive IS RANGE 1 TO integer'HIGH;
SUBTYPE Natural IS RANGE 0 TO Integer'HIGH;
```

4.3.10. Конверсионен тип

VHDL прави разлика между целите и реалните числа като търси десетична точка. Например, 3 ще бъде разпознато като цяло число, докато 3.0 ще бъде интерпретирано като реално. Например, ако променливата Fred е декларирана като реален тип, тогава

```
Fred <= 3;
```

ще предизвика грешка поради разлики в типа. Не може да се присвоява целочислена стойност към реален обект. За да се реши проблема, в случая, може да се използва функцията за конвертиране на типа.

```
Fred <= REAL (3);
```

Добре е да се избягва използването на конвертиране на типа. Често нуждата от конвертиране може да се избегне чрез точна структура на модела и избор на типове.

4.4. Оператори

Операторите във VHDL са: логически, оператори за сравнение, оператори за присвояване, знакови и аритметични.

Те се характеризират с име, изчисление (функция), брои операнди, тип на стойността на резултата. Във VHDL могат да се дефинират и нови оператори, като функции, за всеки тип на операндите и стойностите на резултата. Дефинираните във VHDL оператори са дадени в таблица 4.1:

Таблица 4.1

ТИП	ОПЕРАТОР	ПРИОРИТЕТ
Logical	and or nand nor xor	Най-нисък
Relational	= /= < <= > >=	
Adding	+ - &	
Unary(sign)	+ -	
Multiplying	* / mod rem	
Miscellaneous	** abs not	Най-висок

Операторите в даден ред са с един и същи приоритет, а тези от даден ред са с по-висок приоритет от операторите от по-горен ред.

Приоритетът показва мястото на оператора (дали ще го използваме преди или след даден друг оператор). Например:

A + B*C = A + (B*C)
not BOOL and (NUM=4) = (not BOOL) and (NUM=4)

VHDL позволява съществуващи оператори да бъдат прилагани към нови типове операнди. Например, оператора AND може да бъде приложен да работи с нови логически типове. За да се използват два оператора с различен приоритет в един и същи израз, трябва да се използват скоби, за да се групират операндите.

4.4.1. Логически оператори

Операндите за логическия тип трябва да са от същия тип. Логическите оператори “and”, “or”, “nand”, “nor”, “xor”, и “not” се използват с операнди от тип BIT, BOOLEAN и едномерен масив от тип BIT и BOOLEAN. Операндите, които са масиви, трябва да имат един и същ размер. Логически оператор, приложен към два масива, е приложен към двойки елементи от двата масива.

4.4.2. Оператори за сравнение

Операторите за сравнение като = или >, сравняват два операнда от един и същи тип и връщат стойност от тип BOOLEAN.

IEEE VHDL дефинира оператори за равенство (=) и неравенство (/=) за всички типове. Два оператора са равни, ако имат една и съща стойност. За масиви и тип запис, IEEE VHDL сравнява съответстващите на операндите елементи.

4.4.3. Оператор IF

Операторът IF дава възможност за условно изпълнение на последователни изрази. Състоянието на израза на оператора IF трябва да дава резултат от тип BOOLEAN. Ако условието е оценено като “истина”, тогава изразът се обръща към полето THEN на израза IF, в противен случай условието се оценява като “неистина” и се изпълнява условието ELSE от израза за IF, ако съществува такова.

Изразът за IF може да бъде продължен, за да даде съставни варианти, като се използва конструкцията ELSIF. (Обърнете внимание на правописа!). Изразът IF може да има няколко ELSIF части, но само един ELSE. Изразът IF винаги завършва с единичен END IF, независимо от броя на представените ELSIF конструкции.

4.4.4. Оператор CASE

Този оператор дава възможност да бъде изпълнена последовательност от изрази, базирани върху подбрани изрази. При избиране на израз ще се върне стойност, която маркира един от възможните избори от списъка на оператора case, или маркира

“захващащ” избор в израза. След като бъде определен изборът на израз, изпълнението продължава с израза, който следва края на оператора case.

Изразяването на оператор CASE трябва да резултира в дискретен тип или едномерна матрица от символи. Изборът на оператор CASE трябва да е уникален. Не трябва да се дублира нито една стойност. Също така, ако всички възможни стойности, които изразът може да върне не са изрично направени като списък, тогава трябва да се включи и изборът на OTHERS.

4.4.5. Оператор NULL

Оператор дефиниращ нулева функция. Намира приложение при инструкциите за разклонение, давайки възможност за допълнително дефиниране на случаите, в които от проектираната система не се очаква активност. Допринася за синтезиране на по-стабилни работни структури (схеми).

4.4.6. Оператор LOOP

VHDL осигурява удобна кръгова конструкция, която дава възможност да бъдат конструирани няколко различни типа цикли. Това е илюстрирано по-долу.

4.4.7. Цикъл WHILE

В цикъла WHILE, условията на цикъла (BOOLEAN) се проверяват всеки път преди да се изпълни цикъла. Ако условието е “истина”, тогава цикъла се изпълнява. При “неистина” изпълнението продължава с изразът, който следва параграфа END LOOP. Напълно е възможно да се създаде цикъл while, където съдържащите се изрази не се изпълняват никога, т.е. условието винаги се определя като “неистина” или където цикълът никога не свършва, т.е. винаги се оценява като “истина”.

4.4.8. Цикъл FOR

Този цикъл определя фиксиран брой повторения. Декларирането на цикъл FOR дефинира показателя на цикъла, който заема последователни стойности от дадения ред за всяко повторение на цикъла. Индексът на цикъла може да бъде смятан за константа, тъй като той не може да бъде променян от операторите вътре в цикъла. Също така, индексът на цикъла не съществува извън него, т.е. ще изчезне, когато цикъла бъде прекъснат. Забележете: особеност на VHDL е, че индексът на цикъла не може да бъде изрично деклариран в полето за деклариране на даден процес, функция, процедура и т. н. Индексът на цикъла е безусловно деклариран от оператора FOR.

4.4.9. LOOP

За приложения, където циклите LOOP и WHILE са неприложими, VHDL осигурява обща форма на LOOP. В следващия пример на цикъла ще бъде даден етикет. Етикети на цикли могат да се задават за всички типове цикли и са особено подходящи ако се използват вложени цикли.

```
Simple: LOOP
        последователност от_изрази;
END LOOP Simple;
```

В тази проста форма на цикъл се изпълнява последователността от оператори. За да се предотврати безкрайното изпълнение на цикъла, в него трябва да се включи оператор WAIT или EXIT.

4.4.10. Оператор EXIT и оператор NEXT

Операторът EXIT и операторът NEXT могат да бъдат използвани, за да контролират изпълнението на всички видове цикли.

Операторът NEXT прекъсва изпълнението на текущото повторение на цикъла и стартира следващото повторение. Операторът EXIT също прекъсва текущото повторение на цикъла, но за разлика от NEXT излиза от цикъла.

Ако етикетът на цикъла е пропуснат от операторите NEXT или EXIT, тогава операторът се прилага към най-близкия вътрешен цикъл. Ако има етикет на цикъла, тогава операторът се прилага към цикъл с такъв етикет. Това е полезно например в случай на вложени цикли, където етикетът на цикъла позволява прекъсване на няколко или всички йерархии на цикли. Ако има оператор WHEN, тогава NEXT или EXIT ще се изпълнят, когато булевото условие получи стойност “истина”. Ако резултатът от оператора WHEN е “неистина”, изпълнението на цикъла продължава нормално.

4.5. Изрази и оператори

Изразите извършват определени операции с операндите, за да генерират резултат или набор от резултати. Най-простият израз е A + B, където оператора + е приложен към операндите A и B. Изразите могат да бъдат сложни и да съдържат съставни оператори. При това обстоятелство, споменатите по-горе правила се използват, за да се определи реда на изчисляване.

4.6. Сигнали и променливи

Сигналите се използват за динамична обмяна на данни между последователно изпълними VHDL модули. Ако се използва аналогията с печатните платки, сигналите са линии или проследяване на сигнал. Сигналите могат да бъдат декларирани в полето за деклариране на архитектурната част или като портове в интерфейсната част.

Променливите са подобни на сигналите с изключение на това, че те са валидни само вътре в даден последователен код. Следователно те са локални за определен процес или функция. Променливите не могат да бъдат използвани за обмяна на данни между модули, изпълнявани последователно, т.е. различни процеси, интерфейсни части и др. За да се обменят данни между последователни модули, се използва сигнал/порт.

На променливите може да се присвои стойност. Стойността присвоена на променлива се приемат веднага. Това е в противоположност на сигналите, където на сигнала може да се присвои нова стойност само след приключване на симулацията.

4.7. Атрибути

Типовете и обектите във VHDL съдържат и допълнителна информация, която е достъпна чрез техните атрибути. Атрибутите са определени от символ ‘ (апостроф), следван от името на атрибута. Обектът, предшестващ апостофа е този, към който са прикрепени атрибутите.

4.8. Процедури и функции

Процедурите и функциите във VHDL са същите, като процедурите и функциите в много други езици за програмиране. Процедурата може да бъде използвана като израз във VHDL, докато функцията може да бъде използвана като част от израз.

Процедурите могат да бъдат разделени на последователни или съвместно действащи. Ако се извикват вътре в процес процедурите/функциите се нарича последователни, а ако се извикват от архитектура, се наричат съответно съвместно действащи. Независимо как е била извикана процедурата/функцията, изразите, които тя съдържа винаги се изпълняват последователно.

4.8.1. Процедура

Процедурата може да връща повече от един аргумент. Може да има само входни, само изходни или входно/изходни параметри.

Правила, които са валидни за параметрите на процедурата:

1. Параметрите могат да бъдат от вид: IN, OUT и INOUT.
2. Параметрите могат да бъдат константи, променливи или сигнали.
3. Параметър от тип входен (IN), би могъл да се дефинира като резултат от логическа функция на два или повече входни параметъра.
4. Параметър от тип входно-изходен (INOUT) или изходен (OUT), не се допуска да съдържа в себе си, логически функции.

Параметрите, декларирани във функцията, са формални променливи, докато обектите, чрез които се извършва действителната

обработка, се наричат действителни. Трябва да се отбележи, че ако формален параметър е деклариран като константа, това означава, че процедурата може да не промени стойността на действителния и формалният параметър приема стойност на действителен.

Всяка процедура се състой основно от три елемента: източник – вектор, който да бъде преобразуван; файл – булева стойност за индикация на преобразуването и резултат – целочислена стойност. Резултатният параметър е от тип “INOUT” .

4.8.2. Функция

Функцията може да върне само един аргумент и всички параметри са само входни, т.е. това е превръщане от вида двоично в десетично. Единственото нещо, което може да бъде модифицирано е стойността, която се връща. Ако функцията трябва да модифицира съставни обекти, тогава или се използва процедура, или се връща съставен тип с по едно поле за всяка върната стойност.

4.9. Краен автомат

Крайните автомати (или машини на състоянието) са базови конструкции в една електронна система. Във VHDL те могат да се моделират по 2 начина:

- На ниво компонент чрез структурно VHDL описание на машина на състоянието с използване на прости компоненти като тригери и основни логически елементи (гейтове).
- Поведенчески. Поведенчески моделирана машина на състоянията е най-лесния начин за обяснение чрез примери. Следващият пример описва управление на светофар на ж. п. прелез.

Контролер за управление на светофар

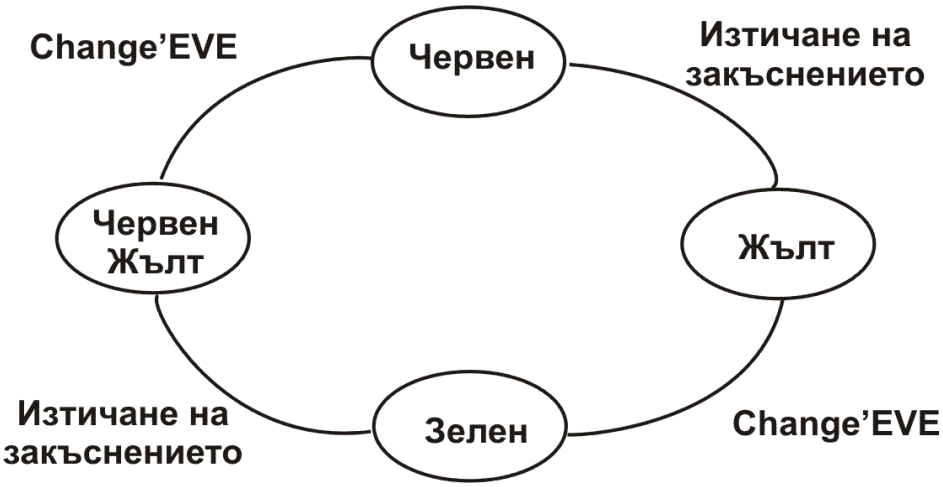
Машината на крайните състояния за управление на светофар преминава през следните състояния: червено-жълто, зелено и жълто. Задават се времената на червено-жълтия и на червения сигнал с отделни параметри. Системата остава или в червено или в зелено състояние докато не се детектира смяна във входния контролен сигнал.

Задание

Да се конструира контролен механизъм за ж.п. прелез, който работи по следния начин:

Нормално бариерите са вдигнати и колите свободно пресичат ж.п. прелеза. Сензор детектира пристигането на влак и подава заявка за спускане на бариерите. Спускането отнема даден интервал от време. Когато бариерите са спуснати, влакът може да премине и това отново отнема някакво време. След това бариерите могат да бъдат вдигнати.

Времето, необходимо за спускане и вдигане на бариерите и това за преминаване на влака трябва да се контролират от предварително зададени параметри.



Фиг. 4.1. Машината на крайните състояния за управление на светофар

4.10. Структурен VHDL

По своята същност структурното VHDL описание, дава възможност за обединяване на отделните съставни модули, в един проект, позволявайки дефинирането на йерархична структура.

Структурният VHDL се състои от три основни части – интерфейсна част, декларативна част на съставните компоненти и конфигурационна част.

4.10.1. Интерфейсна част

На това място се специфицира набора от входно-изходни сигнали, чрез които проектираната единица, би могла да се свърже с останалите елементи от по-горно ниво (йерархично или ниво – печатна платка).

4.10.2. Деклариране на компонент

Декларирането на компонент дефинира интерфейса, чрез който този компонент би могъл да комуникира с останалите компоненти в проекта. В дефиницията се включват брой на интерфейсните сигнали (портове), тяхната посока и тип. Тази декларация би могла да се оприличи на описание, което се изисква при разработката на печатни платки, на ниво компоненти.

Преди компонентът да бъде въведен в проекта, трябва да бъде декларирана конструктивната единица. Декларацията на компонента се поставя в раздела за декларации на архитектурната част (декларирането на компоненти също така може да бъде поставено в пакетите, както е показано по-късно).

Декларирането на компоненти е подобно на декларирането на величини.

COMPONENT Nand2

```
PORT ( A, B : IN BIT);
END COMPONENT;
```

Конкретните връзки на избрания компонент с останалите компоненти в проекта, се задават чрез така наречените етикети и изрази за присвояване. Те дават уникално име на компонента при всеки отделен случай на използване и определят как портовете му трябва да бъдат свързани към сигналите вътре в модула за непосредственост.

Изразът за компонента може да зададе единствено връзка между декларираните в проекта компоненти. При условие, че отново се направи аналогия с печатна платка, изразът за връзка на компонента е писта, свързваща изводите на две или повече интегрални схеми.

4.10.3. Конфигурационна част

С помощта на конфигурацията, всяко едно йерархично ниво във VHDL описанието на текущия проект, би могло да се свърже с нивото, стоящо под него. С цел улесняване, при задаването на връзките между отделните йерархични нива, в езика VHDL е дефинирана възможност за използване на служебни етикети, указващи връзката между съответните компоненти.

Основно конфигурацията се състои от списък на всички “компоненти” съставящи проекта. Конфигурацията определя коя интерфейсна или архитектурна част трябва да бъде използвана за всеки компонент. Може да бъде използвана проста конфигурация, за да се определи йерархията на дизайна, следователно е възможно също да се изгради йерархията на конфигурацията.

Когато са разработени съставни архитектури за един интерфейс, конфигурационните изрази определят коя архитектура трябва да бъде използвана. За този прост случай и при липса на присвоени конфигурационни изрази, VHDL симулатора ще използва последната симулирана (компилирана) архитектура за всеки интерфейс в конструкцията. За да се избегне възможността от объркване, се препоръчва да се използва винаги пълната конфигурация на конструкцията.

Основната форма на конфигурацията е:

```
FOR етикет на конфигурацията OF име на интерфейсната част IS
    конфигурационни изрази
END име на конфигурацията;
```

Полето на конфигурационните изрази може да бъде използвано, за да се конфигурират компонентите на конструкцията. Конфигурацията на компоненти е постигната чрез израза FOR:

```
FOR етикет на компонента:име на компонента
    USE ENTITY име на библиотеката.име на интерфейсната част
(име на архитектурната част);
END FOR;
```

Всеки конфигуриран компонент има израз FOR, за начало на неговата конфигурация и END FOR за край на конфигурацията.

Структурният VHDL може да се илюстрира най-добре чрез пример.

Пример. Полу-суматор.

Структурният VHDL се състои от подкомпоненти и сигнали, свързващи подкомпонентите. Следващият пример дава VHDL описание на полусуматор.

Полусуматорът, използван в този пример съдържа три подкомпонента – едно изключващо ИЛИ и два двувходови И-НЕ. Всеки подкомпонент има уникално текущо име, в този случай Gate1, Gate2 и Gate3.

Вътре в конфигурацията на пълния суматор са използвани два типа компоненти.

Текущата връзка на компонента съдържа списък, който определя кой сигнал на полусуматора е свързан с портовете на свързания компонент. Важно е да се отбележи, че текущите сигнали, присъединени към портовете на свързания компонент, могат да бъдат или портове или вътрешни сигнали на съставната архитектурна част. VHDL кодът, който съответства на интерфейсната и архитектурната част на полусуматора, показан по-долу, ще бъде:

```
ENTITY HalfAdderIS
PORT ( A1, B1 : IN BIT;
        Sum, Carry : OUT BIT);
ARCHITECTURE Structural OF HalfAdder IS
COMPONENT Xor2
      PORT (A, B : IN BIT;
            Y : OUT BIT);
END COMPONENT;
COMPONENT Nand2
      PORT (A, B : IN BIT;
            Y : OUT BIT);
END COMPONENT;
SIGNAL InternalCarry : BIT;
BEGIN
Gate1:Xor2
PORT MAP ( A => A1,
           B => B1,
           Y => Sum);
Gate2:Nand2
PORT MAP (A => A1,
           B => B1,
           Y => InternalCarry);
Gate3:Nand2
PORT MAP (A => InternalCarry,
           B => InternalCarry,
           Y => Carry);
END Structural;
```

Конфигурационните изрази за полусуматора ще определят коя конструктивна единица от VHDL библиотеката ще бъде използвана за трите текущи компонента, Gate1, Gate2 и Gate3.

Приемат се подходящи интерфейсни и архитектурни части за тези компоненти, които вече са били успешно анализирани и това е направено в текущата VHDL работна библиотека. Получава се следното:

Таблица 4.2

Име на компонента	Име на интерфейса	Име на архитектурата
Xor2	XorGate2	Simple
Nand2	NandGate2	Simple

Тогава конфигурацията за полусуматора ще бъде:

```
CONFIGURATION HalfAdderConfig OF HalfAdder IS
    FOR Structural
        FOR Gate1:Xor2
            USE ENTITY WORK.XorGate2(Simple);
        END FOR;
        FOR Gate3:Nand2
            USE ENTITY WORK.NandGate2(Simple);
        END FOR;
    END FOR;
END NahlfAdderConfig;
```

Името на тази конфигурация е HalfAdderConfig. Тъй като конфигурацията е конструктивна единица във VHDL, на VHDL симулатора може да бъде зададено конфигурационно име. Следователно, за да се симулира схемата на полусуматор, симулаторът трябва да симулира конструктивната единица HalfAdderConfig.

VHDL конфигурацията съдържа израза FOR за определяне на състоянието на всеки компонент. Вложеният израз FOR дава възможност йерархията на пълния дизайн да бъде определена. Първият израз FOR в този пример определя, че тази конфигурация е свързана със структурната архитектура на интерфейса на полусуматора. Вложените вътре в този FOR изрази са три израза FOR, по един за всеки от непосредствените компоненти.

Тези изрази FOR определят текущото име на компонента (например Gate1) и техния тип (например Xor2). Вътре в израза FOR има параграф USE, който определя коя VHDL конструктивна единица ще бъде използвана, за да осигури модел на този компонент.

Параграфът USE определя конструктивната единица, която ще бъде използвана, интерфейса (може да бъде друга конфигурация ако се използва йерархия на конфигурация), библиотеката, в която да бъде намерена и накрая специфичните имена на интерфейсната и архитектурната част.

4.11. Тестови VHDL описания

Пълната VHDL конфигурация се състои от две главни части: VHDL код, който описва конструкцията и тестова установка. Следователно повечето инструменти на VHDL разрешават употребата на драйв, за да принудят файловете да стимулират конструкцията. Препоръчваната методология е да се опишат стимулите във VHDL.

4.12. Тестова установка

Най-просто тестовата установка е допълнителна VHDL конструктивна единица, чиито изходи са свързани към входовете на тестваната конфигурация. Това се постига чрез създаване на допълнително ниво в йерархията, което свързва (на компонентно ниво) тестовата установка и тестваната конфигурация.

Това, което трябва да се отбележи за най-високото ниво на конфигурацията е, че то няма нито входни, нито изходни портове. VHDL описанието, което може да извърши това за схемата на полусуматор е:

```
ENTYTI HalfAdder OF HalfAddTop IS
END HalfAddTop;
ARCHITECTURE Structute Of HalfAddTop IS
COMPONENT HalfAdder
      PORT (A1, B1 : IN BIT;
            Sum, Carry : OUT BIT);
END COMPONENT;
COMPONENT TestBench
      PORT (A1, B1 : OUT BIT);
END COMPONENT;
SIGNAL DriveA1, DriveB1 : BIT;
SIGNAL MonitorSum, MonitorCarry : BIT;
FOR HA:HalfAdder USE ENTITY Work.HalfAdder(Structural);
FOR TB:TestBench USE ENTITY Work.TestBench(Simple);
BEGIN
HA:HalfAdder
      PORT MAP (A1 => DriveA1,
            B1 => DriveB1,
            Sum => MonitorSum,
            Carry => MonitorCarry);
      TB:TestBench
            PORT MAP (A => DriveA1,
                  B => DriveB1);
END Structure;
```

Компонентите на тестовата установка съдържат VHDL код, който осигурява стимулите за тестваната конструкция. Тестовата установка е свободна да съдържа всички валидни VHDL изрази и следователно може да бъде сложна. Най-простата тестова установка би могла да се състои от множество серийно подавани във времето стойности към съответните сигнали (портове). Например:

```
DriveA1 <= ‘0’ AFTER 20 NS, ‘1’ AFTER 40 NS, ‘0’ AFTER 60 NS;
DriveB1 <= ‘0’ AFTER 30 NS, ‘0’ AFTER 50 NS, ‘1’ AFTER 70 NS;
```

4.12.1. Стилове на VHDL тестовата установка

Като концепция VHDL, дава възможност за изграждане на различни конструкции на тестови постановки. Основно в практиката е възможно да се разграничат три основни типа тестови установки: управлявани от файл, управлявани от команда и самостоятелни.

4.12.2. Тестова установка управлявана от файл

Тестовата установка управлявана от файл използва VHDL, за да чете данни от файл, да симулира веригата и да запише отговора на схемата във втори файл. С други думи, достъпът до файл във VHDL не може да се сравнява с останалите операции. Следователно, за конфигурация с голям брой портове и/или дълги тестови вектори, изпълнението на този подход е ограничено. От друга страна, този метод е полезен там, където съществуват тестови вектори (от не VHDL симулация), за да се тестват нови VHDL конфигурации.

4.12.3. Командно ориентирана тестова установка

Подходът за управление на тестовата установка с команда е подобен на подхода за управление с файл, с изключение на това, че командите се четат и превръщат в тестови вектори от записания във VHDL команден интерпретатор. Предимството на този метод е, че намалява размера на изискваният входно/изходен файл и следователно е по-бърз.

4.12.4. Самостоятелна тестова установка

Самостоятелната тестова установка не изисква връзка с външния свят. Тя управлява и изобразява операциите при конструиране. Предимството на този метод е, че тестването е автоматично и външните комуникации са ограничени.

4.13. Конфигурации

4.13.1. Прости конфигурации

В следващия пример, текущият компонент, наречен ResetDff от тип DFLOP е насочен към интерфейса DFlipFlop, архитектурата BasicBehavioural, която е компилирана в библиотеката WORK:

```
[...]
FOR ResetDff : DFLOP
      USE ENTITY WORK.DFlipFlop (BasicBehavioural);
END ROF;
[...]
```

Възможно е да се използва йерархията на конфигурацията. В този случай параграфът USE от конфигурационния израз ще се обръща към други конфигурационни изрази, които един след друг определят интерфейса/архитектурната част, които трябва да бъдат използвани. Например:

```
[...]
FOR ResetDff : DFLOP
      USE CONFIGURATION WORK.Deffered;
END FOR;
[...]
```

Конфигурацията deffered е:

```
CONFIGURATION Deffered PF DFlipFlop IS
FOR BasicBehavioural
END FOR;
END Deffered;
```

В конфигурационния израз всеки компонент може да бъде изрично определен чрез неговия свързващ етикет. Например: ResetDff в долния пример може да бъде определен с параграфът ALL или OTHERS. Така множество компоненти, биха могли да бъдат обвързани с едно общо поведенческо описание, на по-ниско йерархично ниво. Например:

```
FOR ALL : And2
      USE ENTITY WORK.And2 (Complex);
END FOR;
```

В тази конструкция, всички компоненти с име “**And2”** ще използват поведенческо описание от по-ниското йерархично ниво с интерфейсна част And2 и архитектурата Comlex, компилирана в работната библиотека.

4.13.2. Изрази за конфигуриране на архитектури

Възможно е дефинирането на йерархията да става и чрез конфигурационни изрази, интегриращи VHDL описанието на по-горния йерархичен слой. От синтактична гледна точка, в този случай конфигурацията би трябвало да бъде описана в декларативната част на модела. В такива случаи не са необходими отделни конфигурационни изрази за тези компоненти.

```
[...]
COMPONENT And2
      PORT ( A, B : IN BIT;
             Y : OUT BIT);
END COMPONENT;
SIGNAL M13tcff, ResetCount, InvClk : BIT;
FOR Gate3 : And2 USE ENTITY WORK.And2 (Complex);
BEGIN
Gate3 : And2
```

```
        PORT MAP ( A => M13tcff,
                   B => InvClk,
                   Y => ResetCout);
    [...]
```

4.13.3. Конфигуриране на портовете

В повечето случаи имената на портове на интерфейса, който се конфигурира, отбелязват декларацията и появата на текуща карта на портовете. Ако имената на портовете се различават, конфигурацията трябва да съдържа израз за карта на портовете, за да позволи да се анализира или симулира конструкцията. Например, интерфейсът Inv е деклариран с портове A и Y, докато при декларацията на компонента в конфигурационния израз, портовете са наименовани като “Input” и респективно “Output”. Конфигурационните изрази съдържат карта на портовете, която казва, че порт A от реалния интерфейс е същия като порт “Input” от дефинирания на по-ниско йерархично ниво модел на компонента. Това е показано в следващите примери на реален интерфейс и конфигурация:

```
ENTITY Inv IS
    PORT ( A : IN BIT;
           Y : OUT BIT);
END Inv;
```

Използване на елемента “Inv”, като компонент в по-високото йерархично ниво

```
[...]
COMPONENT Inv
    PORT (Input : IN BIT;
          Output : OUT BIT);
END COMPONENT;
[...]
G2 : Inv
    PORT MAP (Input => Clk);
[...]
```

Конфигурация:

```
FOR OTHERS : Inv
    USE ENTITY WORK.Inv (Delay)
        PORT MAP ( A => Input, Y => Output);
    END FOR;
```

4.14. Рангове, стойности по подразбиране и отворени портове

4.14.1. Рангове

Ранговете позволяват да се променя непосредствеността на компонентите. Типичен пример за общоприета употреба на ранг е да се позволи модификация на закъснението на компонент. Например:

```
ENTITY Inv IS
    GENERIC (Tprop : Time :=2 ns);
```

```
        PORT ( A : IN BIT;
                Y : OUT BIT);
    END Inv;

    ARCHITECTURE Delay OF Inv IS
    BEGIN
        Y <= NOT A AFTER Tprop;
    END Delay;
```

В този пример закъснението в разпространението на модела на инвертор зависи от стойността на ранга Tprop, деклариран в интерфейсната част. Действителната стойност на ранга е определена по време на анализите и не може да бъде променена динамично при изпълнението на симулацията, т.е. има поведение подобно на константа.

Ранговете са декларирани в интерфейсната част, но техните стойности могат да бъдат определени по следните няколко начина:

- Подразбираща се стойност при декларирането на ранг в интерфейсната част;
- Стойност, разпределена към ранга в текущия компонент;
- Подразбираща се стойност, определена при деклариране на компонента;
- Стойност, разпределена при конфигуриране на компонента.

Подразбиращите се стойности в декларирането на ранг или компонент могат да бъдат преодолени като се разпределят стойности при конфигурирането на текущия компонент. Това е илюстрирано чрез следните примери:

1. Текуща стойност при деклариране на интерфейсната част. В следващия пример рангът Tprop ще приеме текуща стойност 2 ns, докато не е разпределена другаде:

```
ENTITY Inv IS
    GENERIC (Tprop : TIME := 2 ns);
    PORT ( A : IN BIT;
            Y : OUT BIT);
END Inv;
```

2. Стойност, разпределена в текущия компонент. Рангът Tprop ще заеме стойност 10 ns само за текущия Gate4 на инвертиращия компонент:

```
Gate4:Inv
GENERIC MAP(Tprop => 10 ns)
PORT MAP (A => ResetCout,
          Y => ResetCout);
```

3. Стойност по подразбиране при деклариране на компонент. В този пример рангът Tprop ще приеме стойност 4 ns за всеки текущ инвертиращ компонент, който няма изрично разпределение присъединено към него:

```
COMPONENT Inv
    GENERIC (Tprop : TIME := 4 ns);
    PORT ( A : IN BIT;
           Y : OUT BIT);
```

4. Стойност разпределена при конфигурацията. В този пример рангът Tprop ще приеме стойност 3 ns за текущия Gate1:

```
FOR Gate1:Inv
    USE ENTITY WORK.Inv(Delay)
        GENERIC MAP (Delay => 3 ns);
END FOR;
```

Възможността да се разпределят рангове при конфигуриране е полезна, тъй като стойността на ранга изисква само прекомпилиране на конфигурацията, а не нова текуща архитектурна част на компонента. Конфигурациите и реда на компилиране на йерархията на композицията ще бъдат разгледани в подразделите.

Пример за генериране на часовник

Възможен метод за генериране на часовник във VHDL е:

```
ENTITY ClockGen IS
    GENERIC (Period : TIME := 100 ns);
    PORT (Clk : INOUT BIT);
END ClockGen;

ARCHITECTURE Behavioral OF ClockGen IS
BEGIN
    Clk <= NOT Clk AFTER Period;
END Behavioural;
```

4.14.2. Стойности по подразбиране

Всички сигнали и портове от вида OUT, INOUT и BUFFER имат драйвер, присъединен към всеки от тях, който определя стойността на сигнала или порта. Всички драйвери имат стойност по подразбиране, която е получена от типа на сигнала или порта. В случай на прост скаларен тип текущата стойност се определя да бъде предишната стойност на този тип. Ако сигналът или портът са от съставен тип, всеки скаларен поделемент на композицията приема своята предишна стойност.

Възможно е да се преодолеят текущите стойности чрез изрично определяне на стойност по подразбиране при декларирането на сигнал или порт:

```
SIGNAL Enable  : BIT := '1';
PORT ( DataIn : BIT_VECTOR (0 TO 3) := "0000";
CLK : IN BIT := '0';
DataOut : OUT BIT_VECTOR (0 TO 3) := "1010");
```

Текущите стойности на сигналите и драйверите са определени по време на оптимизирането. Трябва да се отбележи, че стойността по подразбиране на изходен порт ще преодолее стойността по подразбиране на сигнала към който е свързан (тъй като в този случай той е порта, който определя драйвера на сигнала).

4.14.3. Несвързани портове

В много конструкции е обичайно изводите на устройството да бъдат оставяни несвързани. Например, обичайно е за конструкцията да се използват само изходите Q или Q-черта на тригерите. Обаче, VHDL изисква да има връзка с всеки порт на всеки непосредствен компонент. Има два начина да се реши това:

- Да се декларира допълнителен “фиктивен” сигнал и да се използва като връзка с несвързаните портове.
- Изрично да се укаже, че порта е несвързан чрез декларирането му като OPEN в непосредствения компонент.

Следователно е допустимо VHDL да използва сигнал, за да свърже неизползваните портове, но това не се препоръчва, тъй като увеличава сложността на VHDL (т.е. има повече сигнали).

Всеки порт от вида OUT, INOUT или BUFFER може да бъде оставен несвързан, като това изрично се упомене на по-високото йерархично ниво, чрез оператора “OPEN”.

```
[...]
COMPONENT FlipFlop
      PORT ( Data : IN : BIT;
             ResetL : IN : BIT := ‘1’;
             Clk : IN : BIT;
             Q : OUT : BIT,
             Qbar : OUT : BIT);
END COMPONENT;
[...]
FF:FlipFlop
PORT MAP ( Data => DataSignal,
           ResetL => ResetL,
           Clk => ClkSignal,
           Q => QSignal,
           Qbar => OPEN);
```

4.15. Пакети, библиотеки и конструктивни единици

4.15.1. Пакети

Пакетите осигуряват място за съхраняване на често използвани дефиниции, като им дават възможност да участват чрез съставни конструктивни единици. Пакетите се състоят от съвкупност от типове, константи и подпрограми. Пакетите са много полезно свойство на VHDL.

- Те позволяват свързани дефиниции да се държат заедно, като една конструктивна единица.
- Избягват дефиниции, които изрязват актуални конструктивни единици.
- Дават възможност дефинициите да бъдат разделени между съставните конструктивни единици, с което се избягва излишно дублиране.
- Налагат последователност между композициите и между сложните конструкции.

Пакетът може да бъде използван за деклариране на константи, типове, подтипове, подпрограми и деклариране на компоненти за вътрешно дефинирани величини. (Самите величини не могат да бъдат декларирани в пакети).

Пакетът е разделен на две части – декларация на пакета, която дефинира интерфейса и тяло на пакета, което дефинира детайли със закъснение. В случай, че няма детайли със закъснение тялото на пакета може да бъде пропуснато.

```
PACKAGE име на пакет IS
    поле за деклариране на пакета
END име на пакета;
PACKAGE BODY име на пакета IS
    поле за тялото на пакета
END име на пакета;
```

Пример за пакет може да бъде:

```
PACKAGE Definitions IS
    TYPE TriBit IS ('Z', 'X', '0', '1');
    CONSTANT TriBitHighZ : TriBit := 'Z';
    TYPE TriBitVector IS ARRAY (NATURAL RANGE <>) OF TriBit;
    FUNCTION ResolveTriBit (TriBits : TriBitVector) RETURN TriBit;
    SUBTYPE ResolvedTriBit IS ResolveTriBit TriBit;
END Definitions;

PACKAGE BODY Definition IS
    FUNCTION ResolveTriBit (TriBits : TriBitVector) RETURN TriBit IS
        BEGIN
        [...]
    END ResolveTriBit;
END Definitions;
```

В този пример пакет, наречен definitions, дефинира типа TriBit и TriBitVector, константата TriBitZ, подтипа ResolveTriBit и функцията ResolveTriBit. В този случай полето за деклариране на пакета определя интерфейса на функцията, докато тялото на пакета декларира инструментариума на функцията. Ако пакетът не съдържа подпрограми (процедури или функции), тогава тялото на пакета ще бъде празно и може да се пропусне.

Пакетите са като всички други VHDL конструктивни единици. Те трябва да бъдат анализирани успешно в библиотеката преди да могат да бъдат използвани.

Трябва да се отбележи, че телата на подпрограмите могат да бъдат включени в тялото на пакета, докато в интерфейсната част на пакета може да бъде включена само декларацията на интерфейса на подпрограмата.

4.15.2. Пакети USE и VISIBILITY

Позициите от параграфа могат да се използват като пред имената им се постави представка с името на пакета. Например, сигнала от тип TriBit би могъл да се декларира с:

```
SIGNAL Clk : Definitions.TriBit;
```

В повечето случаи конструктивната единица може да изисква използване на съставни дефиниции от пакета. Следователно най-простия път да се направи видима дефиницията в пакета е да се прибави USE към конструктивната единица:

```
USE WORK.Definitions.ALL;
ENTITY Fred IS
    PORT (...
```

Формата на USE е:

USE име на библиотека.име на пакет.ALL

Където име на библиотека е името на библиотеката, където е анализиран пакета. Име на пакет е името на пакета, а ALL показва, че всички съдържащи се в пакета дефиниции трябва да бъдат направени възможни за конструктивната единица.

4.15.3. Изрази за библиотека

Библиотеките във VHDL могат да бъдат класифицирани в две групи: работна библиотека и библиотеки за справка. Работна е библиотеката, в която се анализира композицията. Помощните библиотеки съдържат конструктивни единици, анализирани преди. Ако препоръчания пакет (или друга конструктивна единица) не е в работната директория, изразът за библиотека трябва да съдържа текущата конструктивна единица.

Изразът за библиотеки съдържа просто имената на библиотеки. Например, ако пакет наречен SystemDefinition е бил анализиран в библиотека наречена Global, тогава:

```
LIBRARY Global;
USE Global.SystemDefinitions.All;
```

4.15.4. Предварително дефинирани пакети

VHDL има няколко предефинирани пакета. Те позволяват по-голяма гъвкавост при проектирането и тестването. Чрез широк набор от стойностни типове, проектанта лесно би могъл да работи със състояния като висок импеданс, силна единица, силна нула, неопределено състояние, състояние на конфликт и др. В допълнение повечето производители подсигуряват други пакети, осигурявайки допълнителни възможности като аритметични изчисления, статистически анализ и градивни блокове от високо ниво като “Stacks” и “Queues”.

Типичен пример за такъв пакет е “Standard”. В него би могло да се видят предефинирани типове, като “BIT”, “CHARACTER”, Обхват на изменение за тип “INTEGER” и др. Може да се твърди също така, че поради дефинициите, които съдържа този пакет, той е утвърден като базов за повечето среди за проектиране.

Полезен, също така пакет е и “textio”, позволяващ работа с файлови структури.

```
USE Std.Textio.ALL
```

Някои пакети са също независими IEEE стандарти. Добър пример за това е standard_logic_1164. Този пакет декларира девет типове стойности, наречени STD_LOGIC и STD_LOGIC_VECTOR. Пакетът също дефинира прости операции от типа на AND, NOT и др.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
```

4.15.5. Конструктивни единици

Анализът проверява конструктивната единица за синтактични грешки и ако няма такива поставя анализираната композиция в библиотека. Библиотечна единица е всяка VHDL конструкция, която е възможно да бъде анализирана отделно. Има два класа библиотечни единици, първични и вторични (таблица 4.3).

Таблица 4.3

Първични единици	**Вторични единици**
Деклариране на пакета Деклариране на интерфейсната част Конфигуриране	Тяло на пакета Деклариране на архитектурната част

По всяко време в дадена библиотека може да има само една първична библиотечна единица с едно и също име. Вторичните единици трябва да бъдат анализирани в същата библиотека, както техните съответни първични конструктивни единици. (Това също означава, че първичните библиотеки трябва да бъдат анализирани първи).

Следователно библиотечните единици могат да се анализират отделно, което не означава, че те могат да бъдат анализирани в

произволен ред. Формалните правила управляващи реда на анализиране са:

- Вторична единица може да бъде анализирана само след съответната й първична единица.
- Всяка библиотечна единица, която препоръчва друга първична библиотечна единица може да бъде анализирана само след анализиране на първичната единица.
- Тялото на архитектурата, изисквано от конфигурацията трябва да бъде анализирано преди конфигурацията.

Когато се анализира йерархичната конструкция, анализите трябва да започнат от най-ниското ниво. Редът на анализиране може да бъде образно показан като обърнато йерархично дърво.

Практика е вторичните единици да се държат в същия файл като първичните. Тъй като по-голяма част от инструментариума за анализ на VHDL го извършва през първоначалния файл, това задоволява изискванията за ред на анализите (осигурявайки първичните конструктивни единици да се появят първи във файла).

ГЛАВА 5

Пограмируеми логически устройства

5.1. Комплексни програмируеми логически устройства (CPLD)

За реализиране на сложни функции с класическите PLD елементи са необходими множество макроклетки и логически елементи, което увеличава размера на матриците и значително намалява бързодействието им.

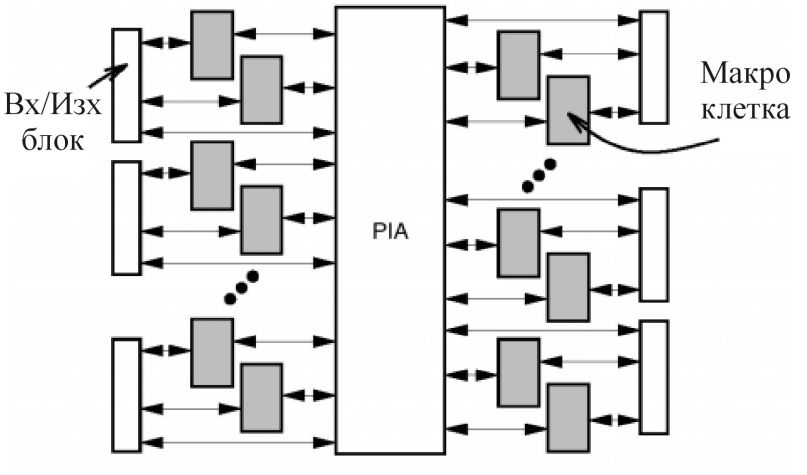
За избягване на този недостатък в така наречените сложни програмируеми елементи *(Complex Programmable Logic Devices-CPLD)* се използват матрици от програмируеми блокове и матрица с програмируеми междусъединения, които дават възможност за изпълнение на логически уравнения на няколко нива.

Сложните логически елементи комбинират бързодействието и удобството при използване на *PAL* елементи с по-голямата степен на интеграция, присъща на матричните кристали.

5.1.1. CPLD на фирмата *Altera*

Чиповете *на Altera* използват *EPROM* технология. Архитектурата им съдържа матрица от големи логически блокове *(LAB - Logic Array Block),* свързани посредством матрица от програмируеми междусъединения *(PIA - Programmable Interconnect Array*), както е показано на фиг. 5.1.

Матрицата от програмируеми междусъединения представлява глобална магистрала, която свързва входно-изходните изводи и логическите блокове. Сложните програмируеми елементи на *Altera* са групирани в няколко фамилии, наречени *MAX (Multiple Array matriX).* Всяка от тях предлага специфични архитектурни решения, количество логически блокове и входно-изходни изводи, осигуряващи голяма гъвкавост при реализацията на различни по сложност логически функции.



Фиг. 5.1. Обща архитектура на чипа

Логическите ресурси на две от фамилиите на *Altera* са дадени в таблица 5.1.

Таблица 5.1. Логически ресурси на EPM5000 и EPM7000

Фамилия	Брой LAB	Брой еквивалентни логически елементи
EPM5000	1-12	600-3750
EPM7000	1-16	2000-20000

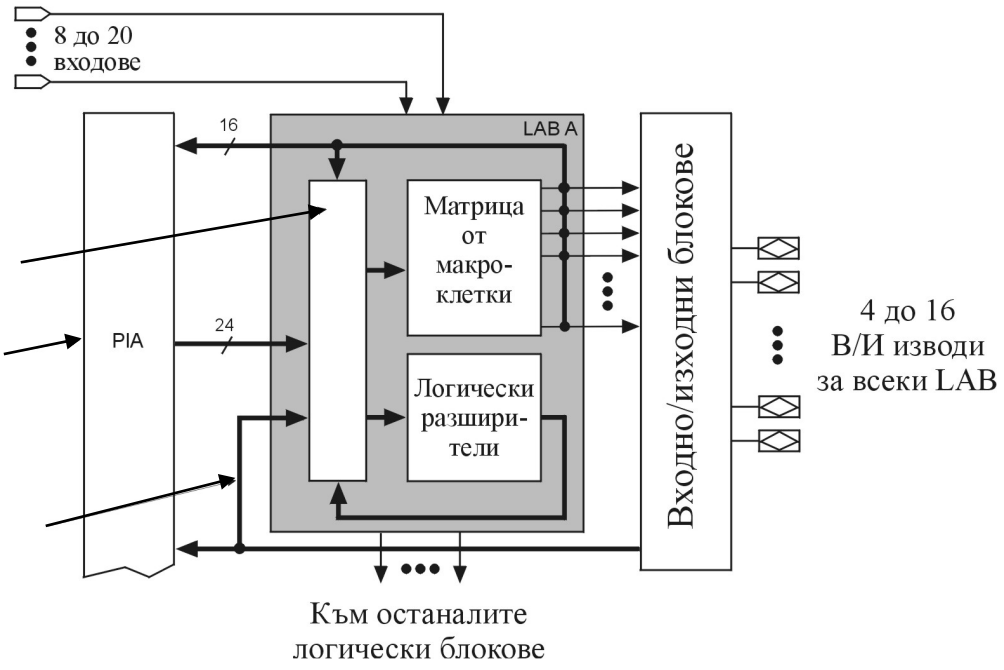
5.1.1.1. Фамилия MAX5000

Схемите от тази фамилия осигуряват 10 ns закъснение от извод до извод и максимална честота 125 MHz. Отделните представители на фамилията съдържат от един (за *EPM5Q32)* до няколко (за *EPM5064, EPM5U8, EPM5192)* логически блока.

На фиг. 5.2 е показан един от наличните логически блокове с входно-изходните му връзки. Както се вижда, блокът може да получава сигнали от фиксирани входове, от програмируемата свързваща матрица или от входно-изходния блок. Този подход позволява реализиране на логически функции с няколко нива на сложност. Допълнително увеличаване гъвкавостта за изпълнение на множество различни функции се осигурява от специфичната архитектура на логическия блок.

- **Логически блок**

  Всеки логически блок съдържа:
  - *матрица от макроклетки;*
  - *логически разширител.*

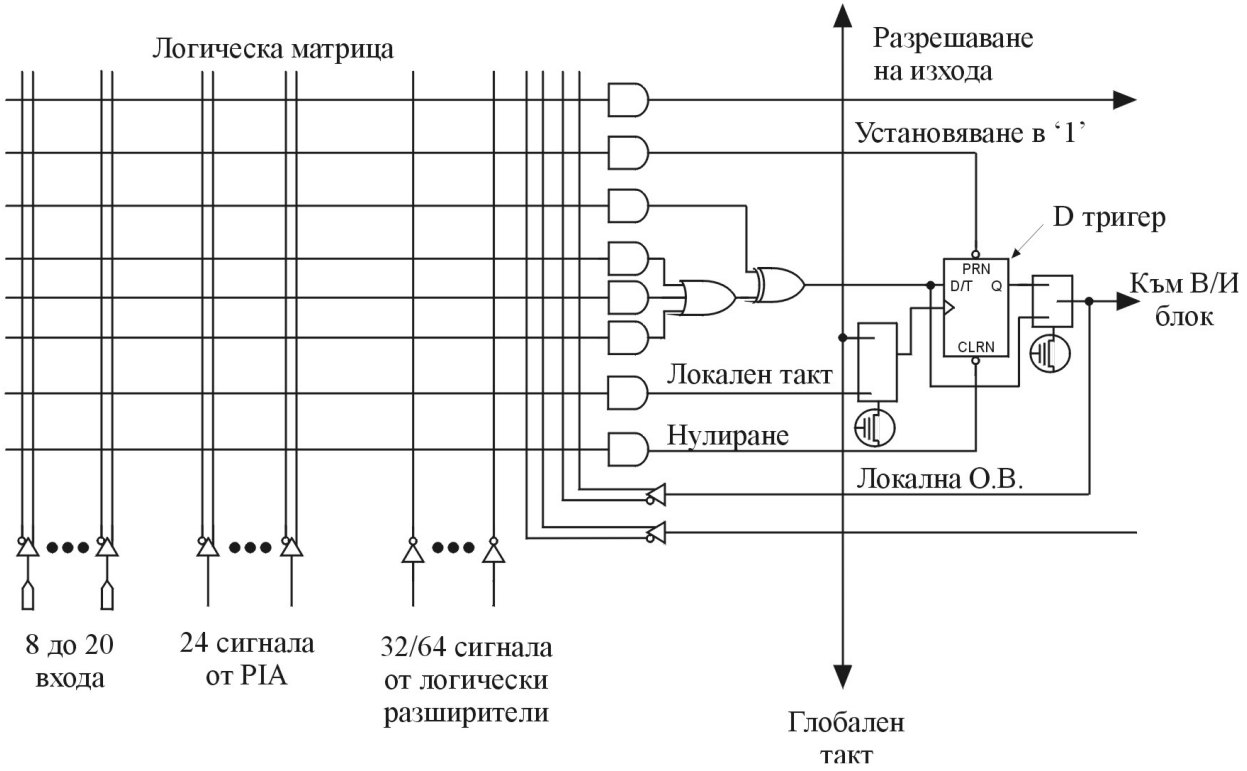


Фиг. 5.2. Логически блок на фамилията MAX5000

За отделните представители на фамилията логическият блок разполага с определен брой макроклетки.

Локалната свързваща матрица осигурява бързи връзки вътре в блока. Към нея са свързани всички изходи на макроклетки за съответния блок. Макроклетките захранват също и входно-изходния блок. Връзката с

останалите логически блокове се извършва посредством програмируемата свързваща матрица.



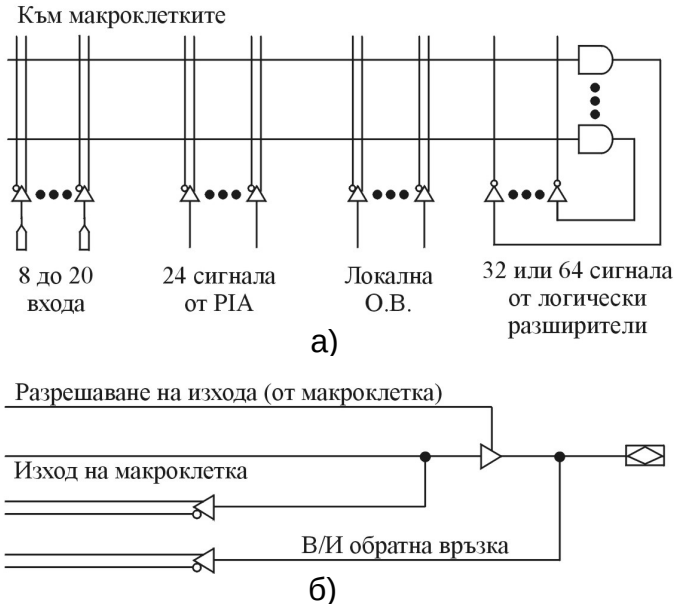
Фиг. 5.3. Структура на макроклетката заложена във фамилията MAX5000

Макроклетките съдържат основните ресурси за изграждане на логическите схеми. Всяка макроклетка има репрограмируема „И" и фиксирана „*ИЛИ*" матрица и тригер, който може да се конфигурира като *D, T, JK* или *SR.* Архитектурата на макроклетката е като тази на *PAL* (фиг. 5.3), но с по-малко логически произведения на регистър. Така логическият блок е по-ефективен, тъй като повечето логически функции не изискват много произведения.

б) Както се вижда от фигурата, макроклетката за MAX5000 съдържа само три „И" елемента, спрямо осем при *PAL.* Комбинаторните функции се реализират в логическата матрица. Трите произведения в последствие се сумират в „*ИЛИ*" матрицата и резултатът постъпва на единия вход на *XOR* елемент. Макроклетката има всички предимства като тази на *PAL* – програмируемо изменение на полярността в изхода, локален и глобален такт, нулиране и установяване в единица на тригера, разрешаване на изхода с три състояния за управление на входно-изходния блок.

Когато е необходимо да се реализира функция, съдържаща повече от три произведения, се използват *логически разширители.* Те представляват група от „И" елементи, които не са свързани към конкретна макроклетка (фиг. 5.4а).

а)



Фиг. 5.4. Логически разширители (а) и входно-изходен извод (б)

Разширителите могат да се използват, когато са необходими допъл-нителни ресурси (включително и за управление на регистрите). Те оси-гуряват гъвкавост при реализация на функции, изискващи много логически произведения.

- **Входно-изходен блок**

Всеки входно-изходен извод (фиг. 5.4б) може да се конфигурира индивидуално като вход, изход или двупосочен извод. Управлението се извършва от буфер с три състояния, по начин аналогичен на описания при *PAL.*

- **Програмируема свързваща матрица**

Второто ниво на йерархия позволява свързване между отделните логически блокове, посредством матрица от програмируеми връзки. Последната се състои от дълги опроводяващи сегменти, които минават в съседство с всеки логически блок. Матрицата на между-съединенията има фиксирано закъснение, осигурява едновременно постъпване на управляващите сигнали към тригерите и гарантира предсказуеми времеви характеристики.

5.1.1.2 Фамилия MAX9000

Схемите от тази фамилия използват 0.65 μm *EEPROM CMOS* технология като осигуряват логически еквивалент от 6 000 до 12 000 елемента при бързодействие 12 ns от извод до извод и 125 MHz тактова честота.

Те позволяват програмиране и препрограмиране без сваляне на чипа от платката, което облекчава разработката и настройката на проекта. Наличието на програмируем бит за сигурност гарантира пълна защита на проекта от копиране след окончателното му завършване.

Макроклетките разполагат с два изхода за независимо използване на комбинаторна и регистрова логика, което увеличава гъвкавостта при реализация на сложни функции.

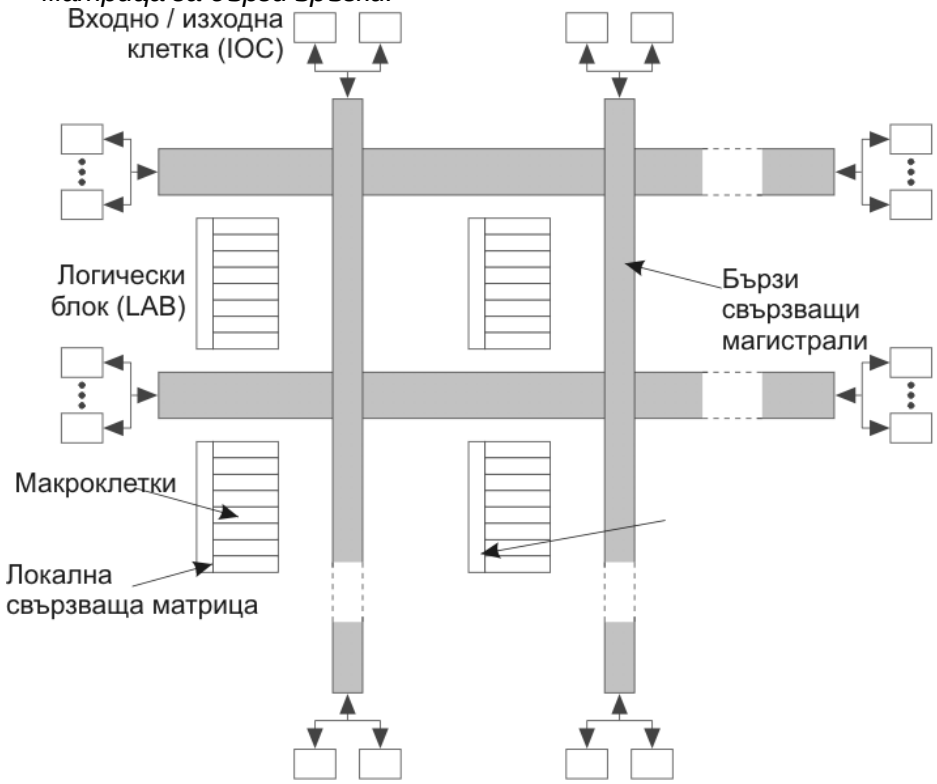
Фамилията MAX9000 позволява оптимизация между скорост и разсейвана мощност. Критичните сигнали използват максимално

бързодействие и пълна мощност, докато останалите могат да се програмират така, че една или повече макроклетки да работят с 50% от мощността си, като при това се добавя само незначително закъснение.

С логическите си възможности схемите от серията MAX9000 осъществяват интегриране на логически функции на системно ниво. С тях лесно се заместват *PAL, GAL, CPLD* и програмируеми логически матрици. Те са много удобни за прототипи на базови матрични кристали, тъй като са сравними с тях по бързодействие и входно-изходни ресурси.

На фиг. 5.5 е показана базисната им архитектура. Тя се състои от:

- *програмируеми логически блокове;*
- *входно-изходни клетки;*
- *матрица за бързи връзки.*

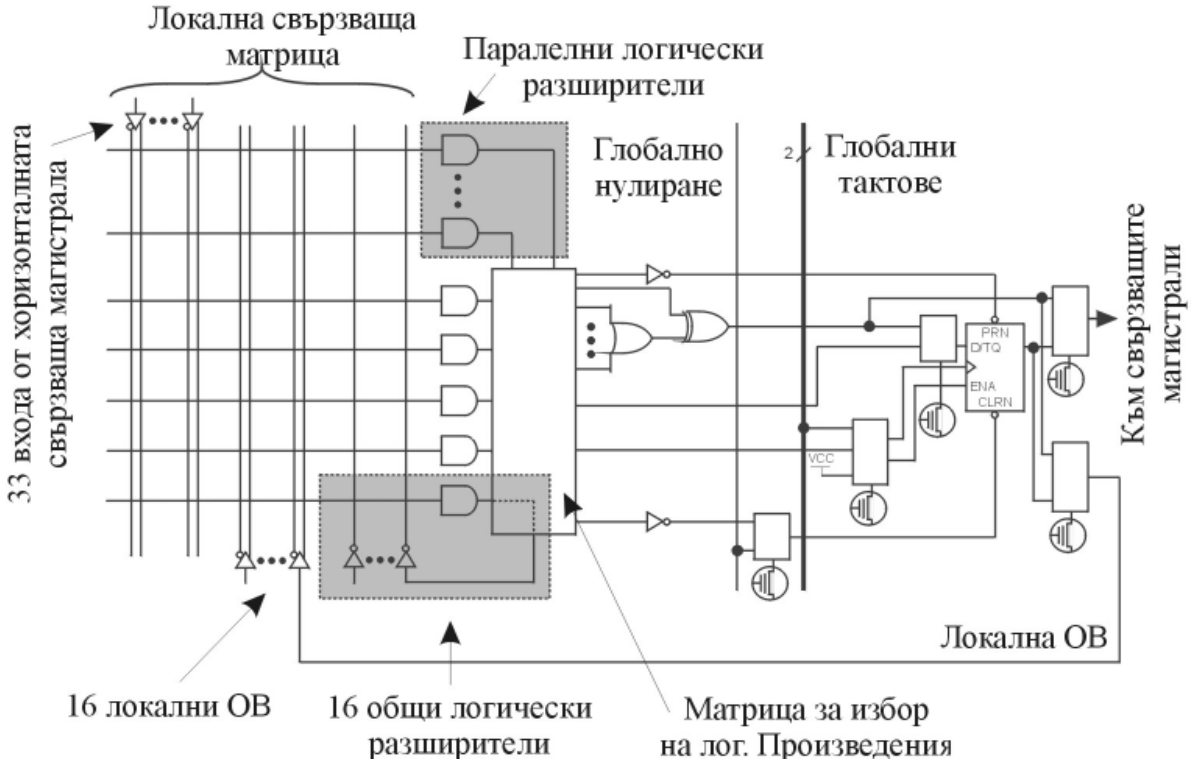


Фиг. 5.5. Базисна архитектура на фамилията MAX9000

Фамилията MAX9000 съдържа от 320 до 560 макроклетки, комбинирани в групи по 16 макроклетки за програмируем логически блок. Логическите блокове в *MAX9000* са подредени в матрица от редове и стълбове. Отделните блокове се свързват помежду си чрез матрицата за бързи връзки. Тя обхваща серия от хоризонтални и вертикални канали, разположени в непосредствена близост до блоковете. Изводите се управляват от входно-изходни клетки, намиращи се в края на всеки ред и стълб.

- **Програмируем логически блок**

Програмируемият логически блок на MAX9000 (фиг. 5.6) съдържа матрица от макроклетки, логически разширители и *локална матрица* за връзките в блока.



Фиг. 5.6. Програмируем логически блок

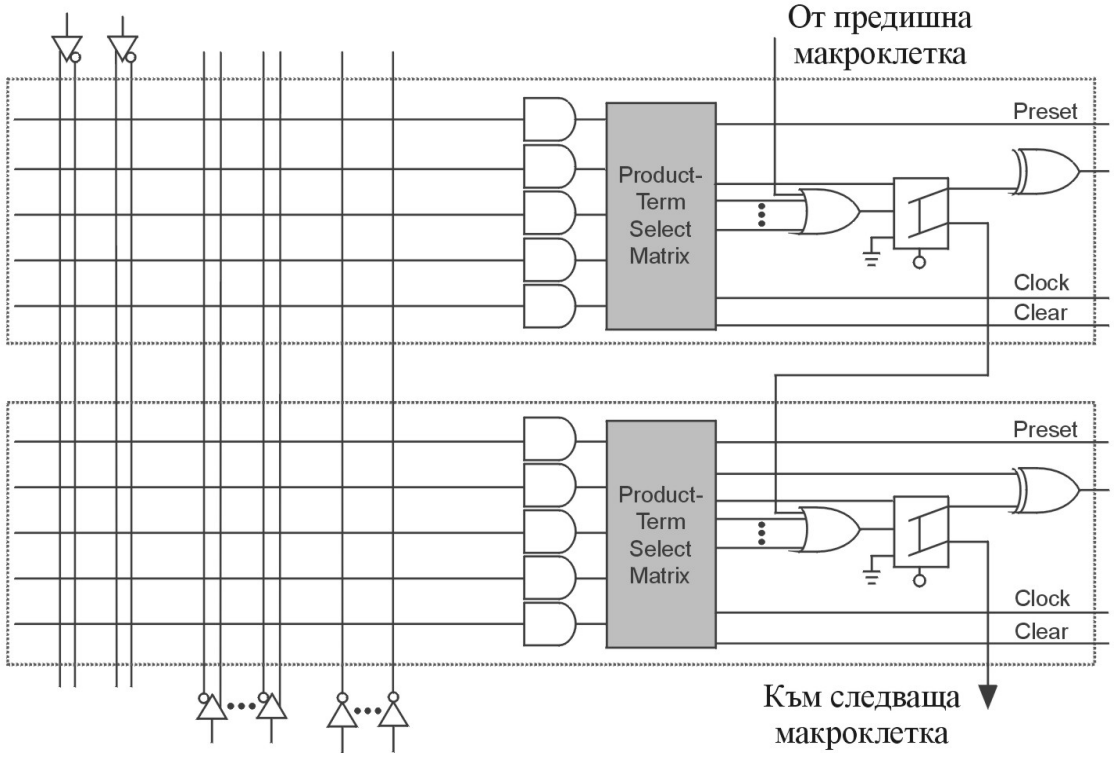
Програмируемият логически блок управлява директно редовете и стълбовете на матрицата за бързи връзки. Веднъж достигнати, тези канали осигуряват сигурно и бързо преминаване на сигналите към други блокове или входно-изходни клетки.

- **Макроклетка за MAX9000**

Макроклетката за MAX9000 съдържа: „И” матрица, блок за избор на „И” елементи от матрицата и програмируем регистър (фиг. 5.7). Тя може да се конфигурира едновременно за изпълнение на комбинаторни и последователни схеми.

Комбинаторната логика се реализира от пет „И” елемента за всяка макроклетка. Блокът за избор определя дали съответните „И” елементи се свързват към „*ИЛИ*” и *XOR* елементите за изпълнение на комбинаторни функции или към регистрите в макроклетката посредством сигналите за нулиране, установяване в единица, локалния такт и сигнала за разрешение на такта или изхода.

За регистрови функции всеки регистър може да се програмира индивидуално като *D, T, JK* или *SR* тригер. За чисто комбинаторните операции сигналите не минават през тригера.



Фиг. 5.7. Архитектура на макроклетката на фамилията MAX9000

- **Логически разширители**

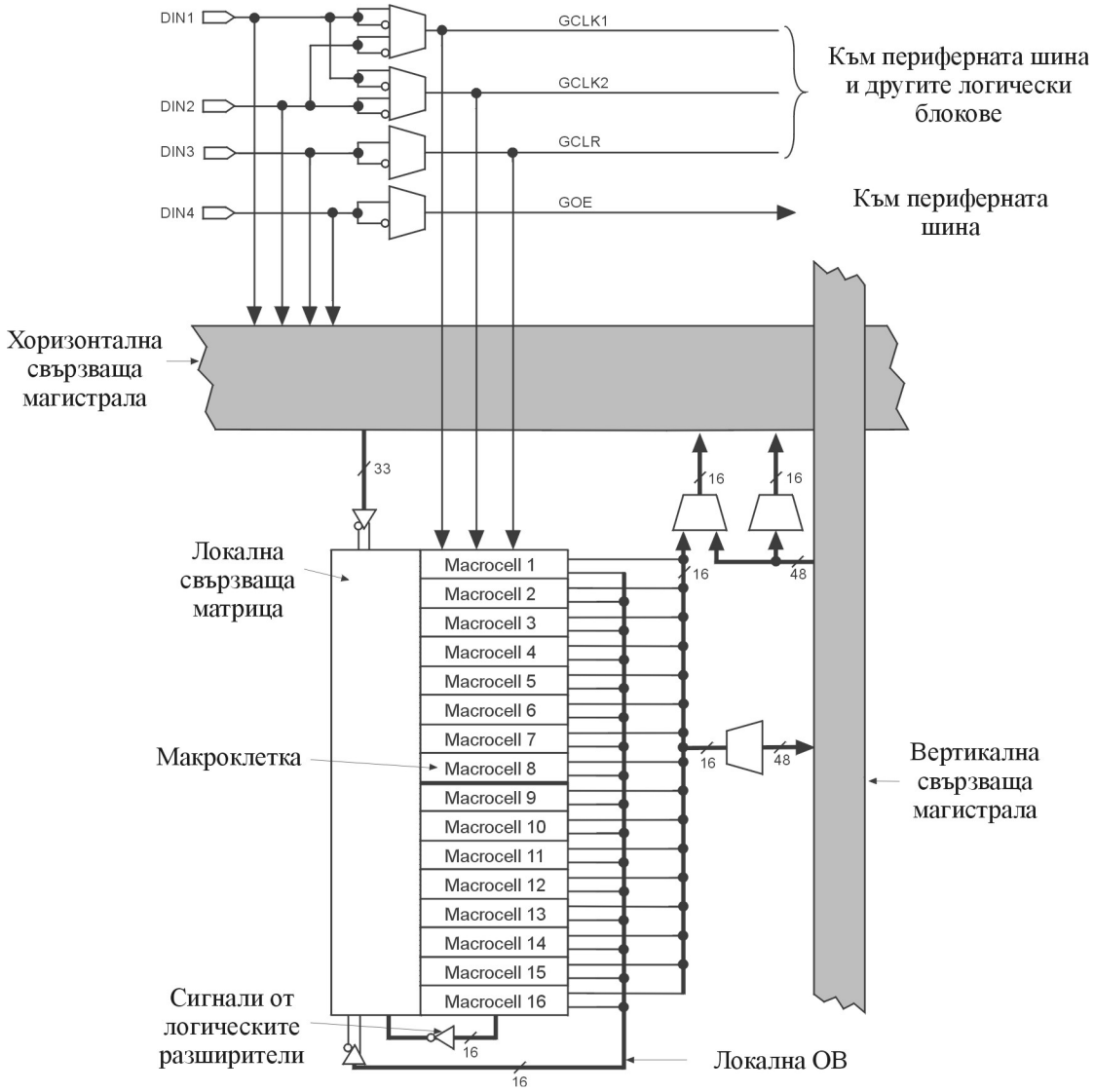
В някои случаи са необходими допълнителни „И” елементи за реализация на по-сложни функции. За тях се използват ресурси от други макроклетки или от логическите разширители. MAX9000 има два типа логически разширители: *разпределени* и *паралелни.* Те спомагат логиката да се синтезира с минимално възможните ресурси за получаване на максимално бързодействие.

Разпределените логически разширители представляват несвързани „И” елементи (по един за макроклетка), които връщат инвертиран сигнал към локалната матрица за използване от останалите макроклетки в блока. Всеки блок разполага с 16 разпределени разширители.

Паралелните логически разширители (фиг. 5.8) се формират от неизползвани „И” елементи, които могат да се свържат към съседни макроклетки за изпълнение на сложни логически функции.

- **Входно-изходни ресурси**

Всеки от изводите може да бъде конфигуриран като вход, изход или двупосочен. Регистърът в клетката може да се използва като входен за данни, с малки времена за установяване, или като изходен за данни с повишени изисквания към стръмността на изходния сигнал. Сигналите за нулиране, установяване, тактуване и разрешение на такта се разпространяват по периферна шина за данни, което осигурява едновременното им подаване към всички регистри в чипа. Тези сигнали се управляват или от същинските входове или от вътрешната логика.



Фиг. 5.8. Логически разширители

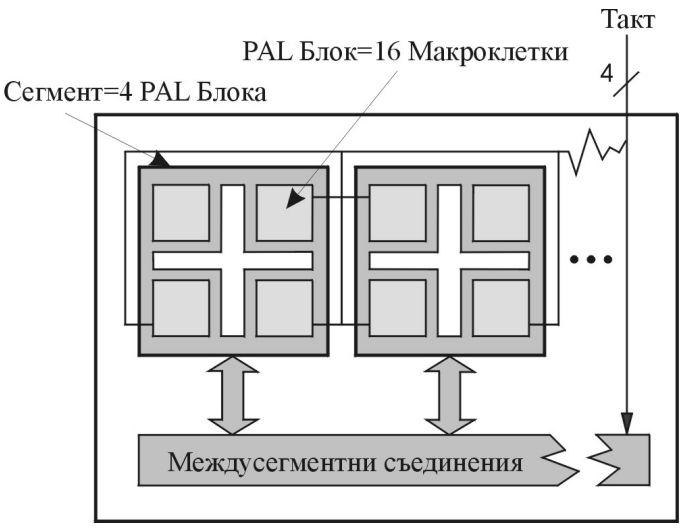
5.1.2. CPLD на фирмата Advanced Micro Devices (AMD)

Използват 0.35 μm *EECMOS* технология, която осигурява голяма плътност на елементите в чипа (от 128 до 512 макроклетки) и високо бързодействие – 7.5 ns закъснение от извод до извод и тактова честота до 125 MHz.

Йерархичната им архитектура е идеална за интегриране на няколко *PAL* елемента и за широка област приложения, включваща високоскоростни изчислителни операции, комуникации и управление. Те позволяват репрограмиране на самата платка.

Архитектурата на фамилията *MACH5* обхваща *PAL* блокове, свързани към две нива от междусъединения (фиг. 5.9). Всяка група от четири *PAL* елемента и собствените им ресурси за трасиране формират един сегмент. Второто ниво на връзки се осъществява от междусегментни съединения.

PAL блокът на *MACH5* (фиг. 5.10) съдържа логическа матрица макроклетки, разпределител на логиката и входно-изходна клетка. „И” матрицата захранва логическия разпределител, а той от своя страна разпределя подходящ брой логически произведения към определена макроклетка.



Фиг. 5.9. Архитектура на фамилията MACH5

Всяка макроклетка може да се програмира за изпълнение на комбинаторни или регистрови операции. За предаване на сигнали от входно/изходна клетка по локалната обратна връзка към матрицата или други *PAL* блокове се използват превключващи матрици.

5.1.3. CPLD на фирмата Lattice

Фирмата предлага фамилията pLSI с четири серии на базата на EEPROM технологията с различна производителност и степен на интеграция, и фамилията ispLSI – с възможност за вградено програмиране с JTAG интерфейс.

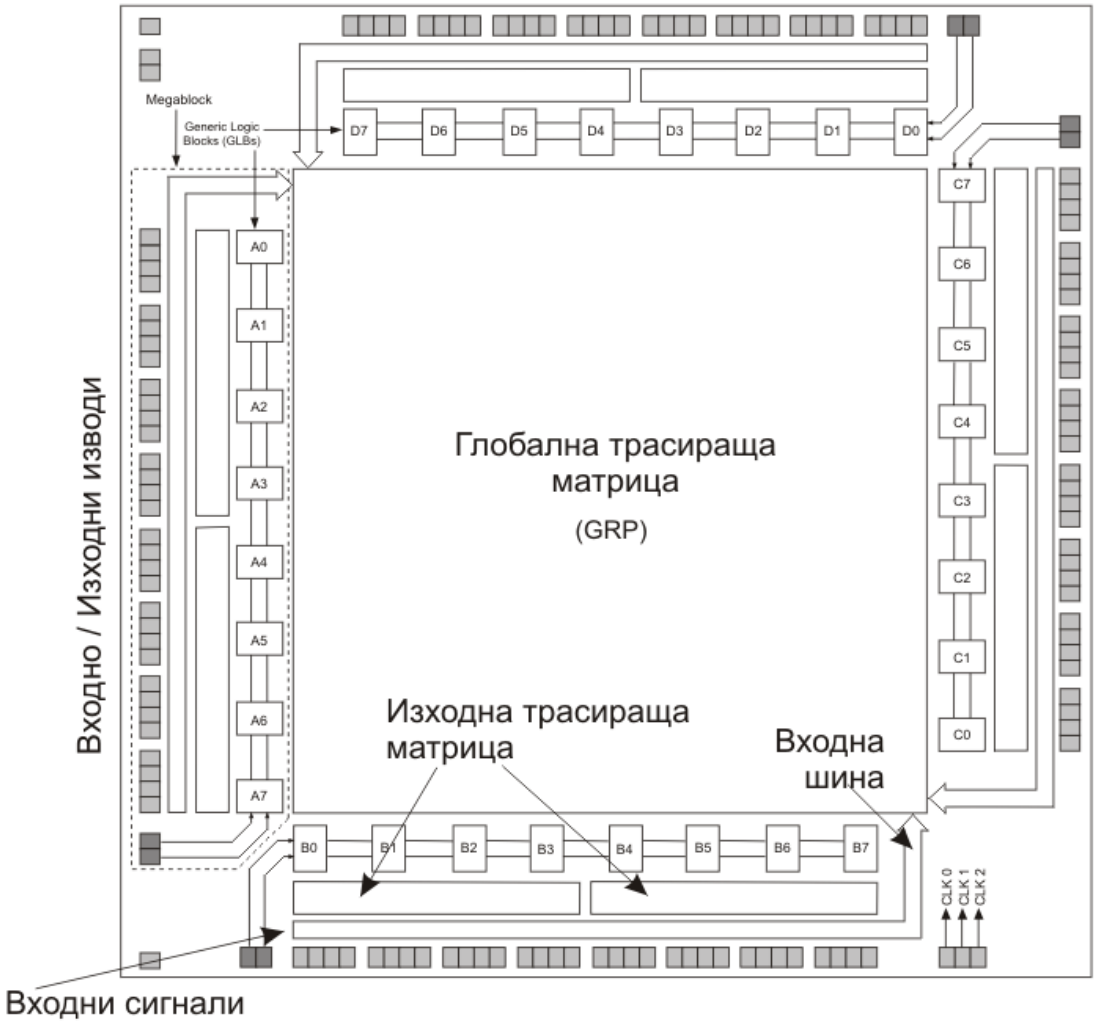
Фиг. 5.10. Логическа клетка на MACH5

Серията 1 000 има капацитет от 1 200 до 4 000 логическите елементи и закъснение от извод до извод 10 ns. Схемите в серията 2000 имат сравнително нисък капацитет - от 600 до 2 000 логически елемента, но предлагат най-високата тактова честота, закъснение от извод до извод 5.5 ns, както и най-високото съотношение между броя на наличните входно/изходни изводи и макроклетки.

Серията 3 000 включва CPLD схемите с най-висок капацитет - до 5 000 логически елемента и закъснение от извод до извод е 10 – 15 ns.

При серията 6 000 осем от PAL блоковете са заменени с два блока памет, които могат да се конфигурират като еднопортова или

двупортова RAM, или двупортова FIFO памет с организация {256 x 18} или {512 x 9}.



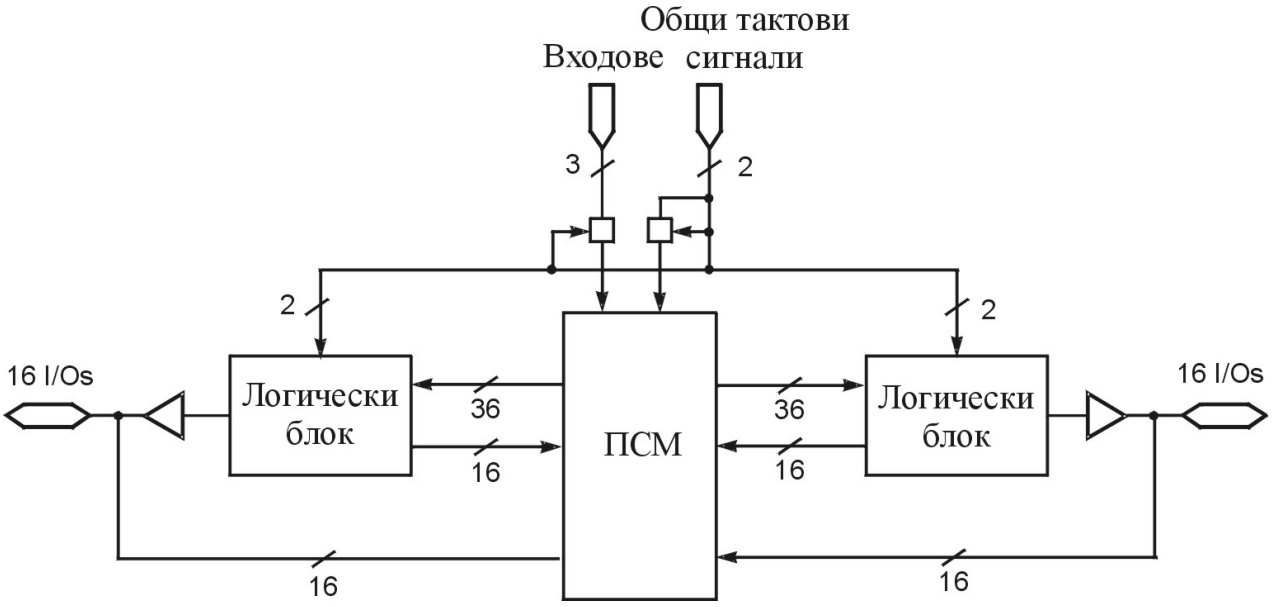
Фиг. 5.11. Архитектура на фамилиите pLSI и ispLSI

Обобщената архитектура на pLSI и ispLSI схемите е показана на фиг. 5.11. Тя включ-ва до 16 PAL блока, свързани към обща трасираща матрица (OTM) и през изходни трасиращи матрици към входно/изходните изводи. Всеки PAL блок включва AND матрица, продукт алокатор и 16 макроклетки. OTM позволява свързва-не на произволен вход и изход на два PAL блока помежду им и към произволен входно/изходен извод. Всички връзки (пътища) преминават през OTM и имат еднакво закъснение.

5.1.4. CPLD на фирмата Cypress

Фирмата Cypress предлага CPLD фамилията FLASH370i, базирана на EEPROM технология с възможности за вградено програмиране чрез JTAG интерфейс. Схемите от фамилията имат до 128 макроклетки, закъснение от извод до извод 8.5 - 10 ns и максимална тактова честота 167 MHz. Архитектурата на FLASH370i е показана на фиг. 5.12. Всяка

схема има 4 общи тактови сигнала и от 2 до 8 логически блока (ЛБ), свързани към програмируема свързваща матрица (ПСМ). Всеки ЛБ има по 8 или 16 фиксирани входно/изходни изводи с изходни буфери с 3-състояния.

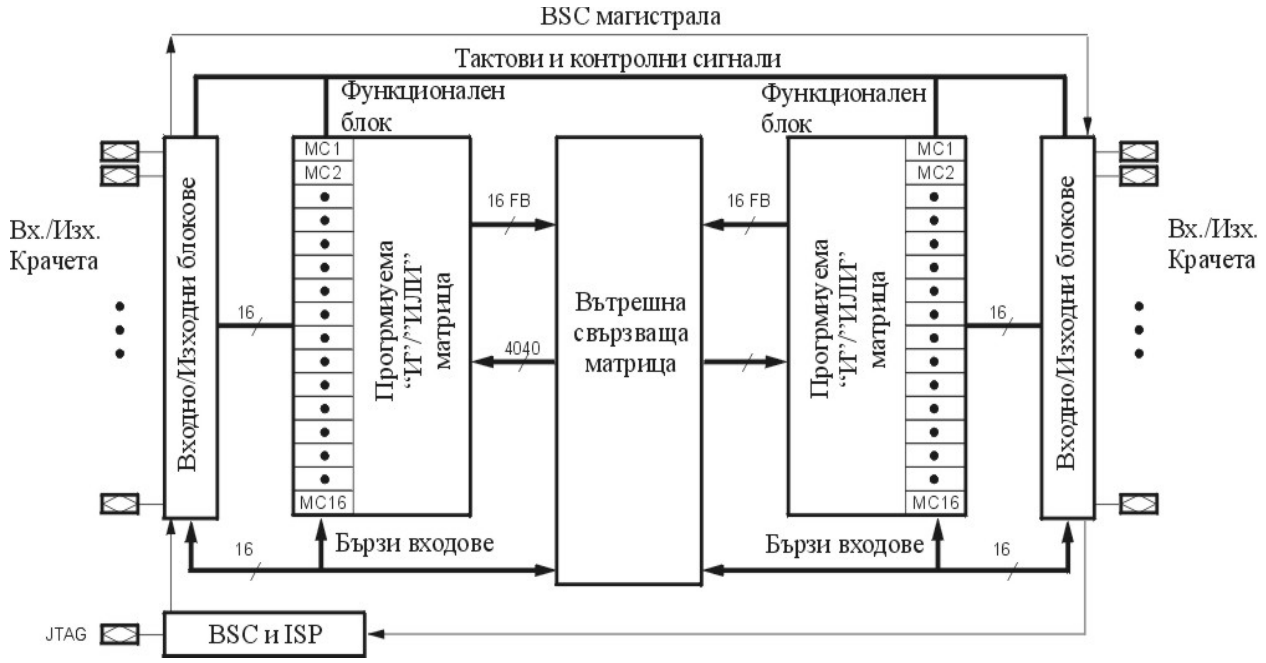


Фиг. 5.12. Архитектура на фамилията FLASH370i

Всеки ЛБ включва AND матрица {72 x 86}. Шест от наличните 86 продукти се използват за управление на макроклетките и входно/изходните блокове, а останалите 80 се преразпределят чрез продукт-терм алокатор между макроклетките, като се използват два метода. Първият включва гъвкаво разпределяне на наличните продукти от 0 до 16 между макроклетките (МК), като всяка МК получава точно толкова продукти, колкото са й необходими. Вторият се базира на използването на общи продукти в рамките на всяка група от 4 МК. При това времето за разпространение на сигналите не зависи от броя на използваните и общите продукти.

5.1.5. CPLD на фирмата Xilinx

CoolRunner2 е фамилия от бързи CPLD с ниска консумация. Вътрешната архитектура е традиционната CPLD архитектура, комбинираща макроклетките във функционални блокове (ФБ), свързани с глобална свързваща матрица, Xilinx Advanced Interconnect Matrix (AIM). ФБ използват конфигурация на програмируема логическа матрица (PLA), която позволява всички логически умножения да бъдат трасирани и споделени между всяка от макроклетките на ФБ. На фиг. 5.13 е показана архитектурата на чиповете CoolRunner2.



Фиг. 5.13. Архитектура на фамилията CoolRunner2

- **Функционален блок (ФБ)**

Всеки ФБ се състои от 16 макроклетки и 40 входни точки за сигнали с общо предназначение. Вътрешната логика е реализирана чрез PLA матрица с 56 логически произведения.

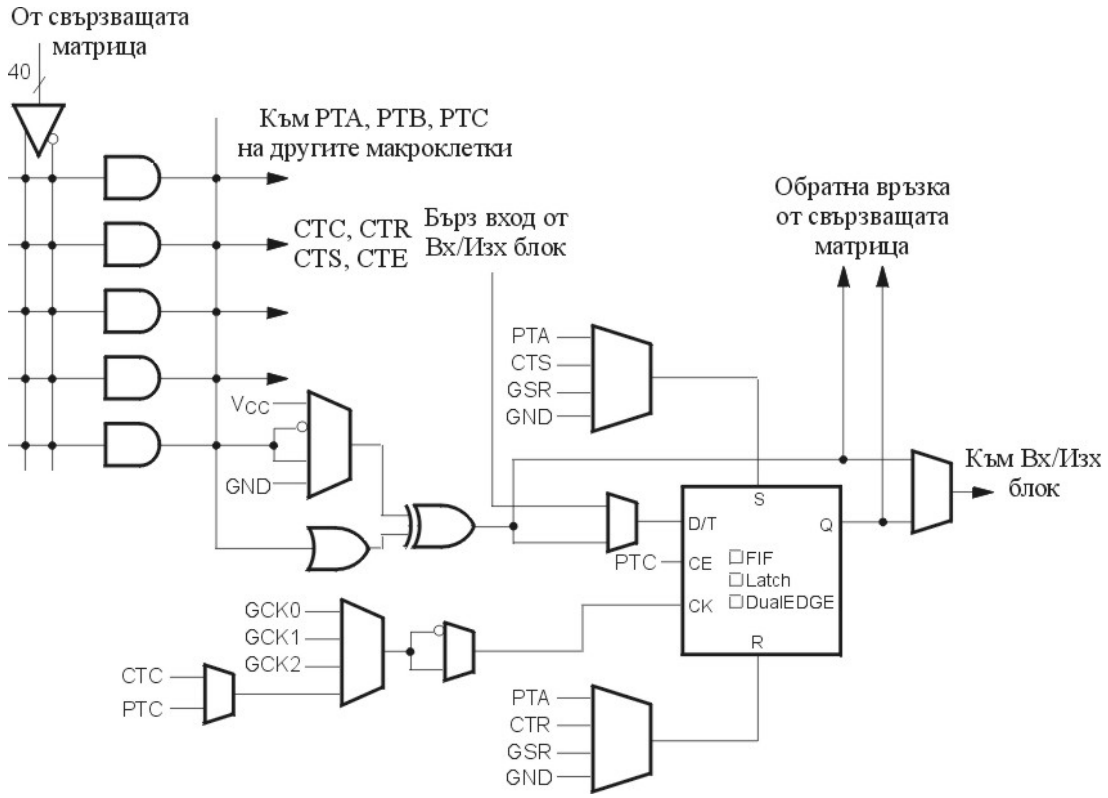
При CoolRunner2 логическите произведения са включени в PLA структурата. Тази структура е изключително гъвкава и много стабилна за разлика от ФБ с фиксирани или каскадно свързани продукт-терми.

При PLA структурата всяко логическо произведение може да бъде свързано към всеки от “ИЛИ” логическите елементи вътре в макроклетките на ФБ. Всяка една логическа функция може да има до 56 логически произведения (ЛП). Формираните логическите произведения биха могли да бъдат използвани от различни логически функции (до 16 в един ФБ), чрез реализацията на “ИЛИ” структури. Характерно за логическите произведения е, че имат еднакво закъснение в цялата структура на чипа.

- **Макроклетка**

Във всяка макроклетка могат да се задават множество от логически изрази, които включват до 40 входни сигнала, използвайки структурата от 56 ЛП, във всеки ФБ. Макроклетката след това може да комбинира крайния резултат с друг единичен ЛП резултат. Полярността на резултата от логическата операция също може да бъде зададен. Логическата функция може да бъде чисто комбинационна или последователностна (регистрова), като за целта елемента памет може да се използва като D или T синхронен тригер, или тригер превключващ по ниво. Всяка макроклетка може да се конфигурира по отношение на тактова честота, нулиране, установяване и разрешение за изход, на

ниво ФБ. Всеки тригер в макроклетката може да се конфигурира за тактуване по нарастващ, по спадащ или и по двата фронта (DualEDGE), което позволява или удвояване на трансфера на данни, или възможност да се използва по-малка тактова честота, което намалява консумацията. При избор на даден фронт за всяка макроклетка може да се избере по кой фронт да превключва. Общ вид на една макроклетка е показан на фиг. 5.14.



Фиг. 5.14. Архитектура на макроклетка

Всяка макроклетка има допълнителен сигнал за разрешение, когато е конфигурирана като D-тригер, което позволява съхранение на информацията независимо от тактовия сигнал. Контрол термите (КТ) се използват, когато една и съща логическа функция се изпълнява многократно от множество макроклетки. КТ и ЛП са достъпни за: тактуване от ФБ (CTC), асинхронно установяване от ФБ (CTS), асинхронно нулиране от ФБ (CTR) и разрешение за изход от ФБ (CTE).

Всеки тригер от макроклетките може да бъде конфигуриран като входен регистър или тригер превключван по ниво, като по този начин осъществява връзка между вх./изх. крачета и вътрешната свързваща матрица (AIM).

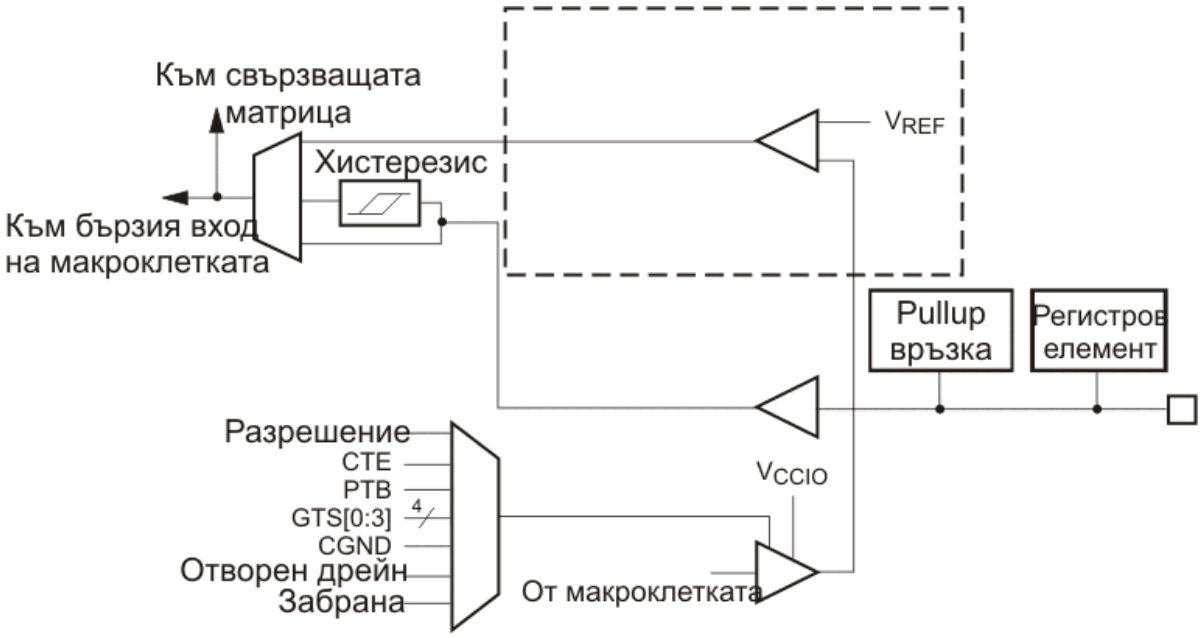
- **Вътрешна свързваща матрица (AIM)**

Предназначението на AIM е да реализира връзката между отделните ФБ и оптимизира закъснението на сигналите между тях. Чрез нея, се организира подаването на входните сигнали към всеки един от ФБ.

- **Входно/изходен блок**

Входно/изходните блокове са приемо-предаватели. Вх./Изх. блок може да бъде конфигуриран в зависимост от параметрите на сигнала. Като допълнение към това, сигналът през всеки вход може да минава и през тригер на Шмит. Това добавя известно закъснение, но в голяма степен намалява входните шумовете. Хистерезисът също така позволява устройството да се тактува от външен такт. Това е показано на фиг 5.15.

Изходите могат да бъдат управлявани директно, да са с три състояния или в схема “отворен дрейн”. Възможно е да се избере режим с повишено или намалено бързодействие.



Фиг. 5.15. Входно/Изходен блок

- **DataGATE**

Патентованата DataGATE технология е разработена, за да предостави допълнително намаляване на консумацията. Всяко вх./изх. краче има множество възможности за изключване, което може да блокира навлизането на нежелани сигнали. Този подход позволява допълнителна оптимизация на консумацията. Когато изходните сигнали не се променят, е възможно тяхното текущо състояние да се поддържа чрез функцията “bus hold”**.** На фиг. 5.16 е покана зависимостта на тока от честотата.

- **Глобални сигнали**

Глобалните сигнали - такт (GCK), установяване/нулиране (GSR) и разрешение за изход (GTS) са проектирани по един и същи начин. Този подход позволява приложният софтуер да използва максимално техните възможности. Всяка глобална функция се осигурява от съответния вариант на ПТ.

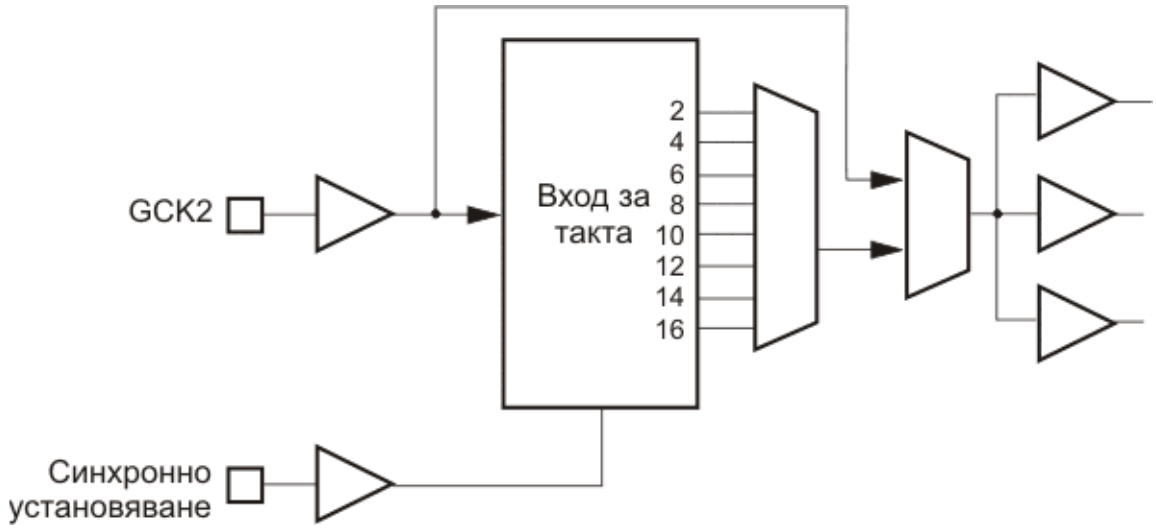
- **Допълнителни опции за тактовия сигнал**

- *Деление.* В архитектурата на CoolRunner-II CPLD са добавени елементи, позволяващи делението на един външен тактов сигнал на стандартни стойности. Опциите са – деление на 2, 4, 6, 8, 10, 12, 14, 16. Резултатът от делението е сигнал с коефициент на запълване 50% за всички възможни деления. Включеното синхронно нулиране гарантира липса на преминаващ тактов сигнал към глобалните възли. Сигналът също така е буфериран и управлява множество шини, осигуряващи минимално закъснение (фиг. 5.17).

Фиг. 5.16. Зависимост на тока на консумация от работната честота

- *DualEDGE.* Всяка макроклетка притежава възможност да удвоява входната си тактова честота. Възможността да се превключва по двата фронта е особено важна за голям брой интерфейсни приложения със синхронни памети, както и с определени I/O DDR приложения.

- *CoolCLOCK.* В допълнение към DualEDGE тригера, енергоспестяване може да се постигне от комбинацията от DualEDGE и деление на тактовия сигнал. Тази система се нарича CoolCLOCK и е проектирана, за да намали енергията консумирана от тактовия сигнал вътре в CPLD.



Фиг. 5.17. Верига за разпространение на такта

- **Сигурност на проекта**

По време на програмирането проектът може да бъде защитен от случайно презаписване или патентна кражба. Четири независими нива на сигурност са осигурени на чипа, като по този начин са елиминирани всички електрически или визуални опити за наблюдение на

конфигурацията. Битовете за сигурност могат да бъдат нулирани само чрез изтриване на цялото устройство.

5.1.6. Сравнителен анализ

- **Технология на производство**

Структурите CoolRunner на фирмата Xilinx се характеризират с т. нар. FZP технология. Това им придава редица предимства пред останалите CPLD структури (таблица 5.2).

Таблица 5.2 Традиционните технологии сравнени с технологията FZP

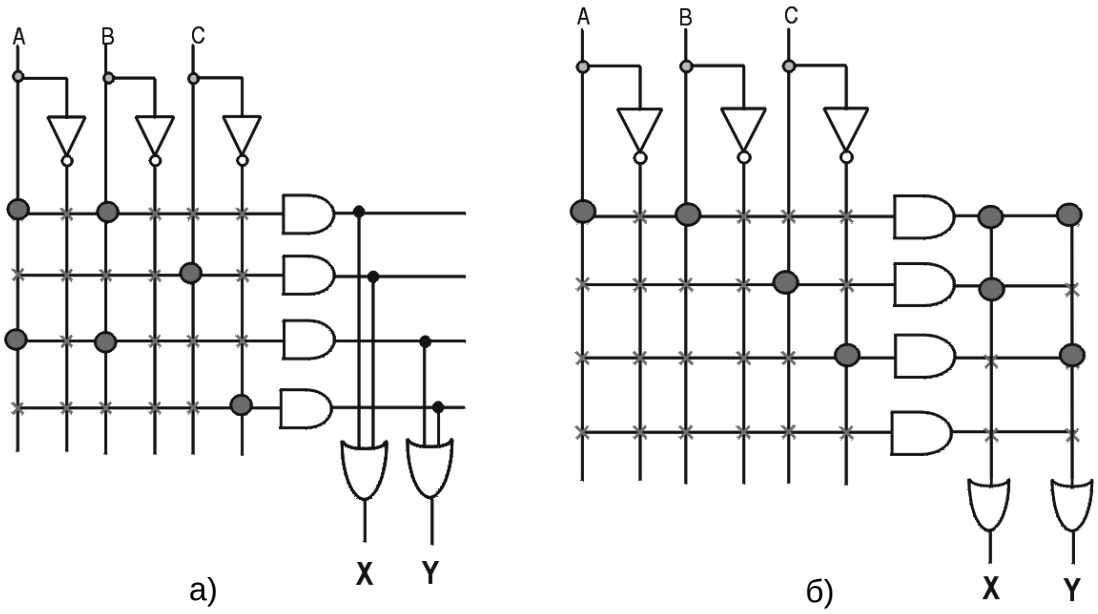
Традиционни технологии	**Технология FZP**
Обикновените CPLD структури използват биполярен усилвател при формиране на логическите произведения	При CoolRunner-II, логическите произведения се формират чрез CMOS технология.
Относително по-голяма консумирана мощност в standby режим	Теоретично е без консумация в standby режим
Консумацията е нелинейна функция от размера на чипа	Консумацията е в линейна функция от размера на чипа с малка стръмност
	Предоставя алтернативи за разработка на високочестотни проекти при запазване на относително малка консумация

- **Архитектура на логическата клетка**

За реализация на една и съща логическа функция, при PAL (фиг. 5.18 а) архитектурата (Lattice) са необходими повече логически произведения в сравнение с PLA (фиг. 5.18 б) архитектурата (CoolRunner-II). Наличието на второ програмируемо поле обуславя по-голяма гъвкавост при програмирането и по-голяма ефективност при използването на цялостния ресурс.

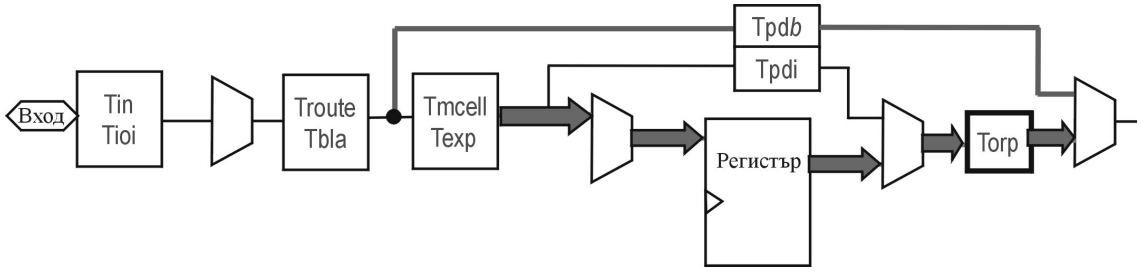
- **Времеви характеристики**

При реализацията на повечето проекти се използват методи за синхронно проектиране, което изисква наличието на регистрови структури. При CPLD структурите предлагани от фирмата Lattice, проектантите разполагат с бърза комбинационна логика, а при тези, предлагани от Xilinx, с по-бързи регистрови елементи. Като цяло CPLD структурите на фирмата Xilinx предоставят по-ефективни възможности при реализация.

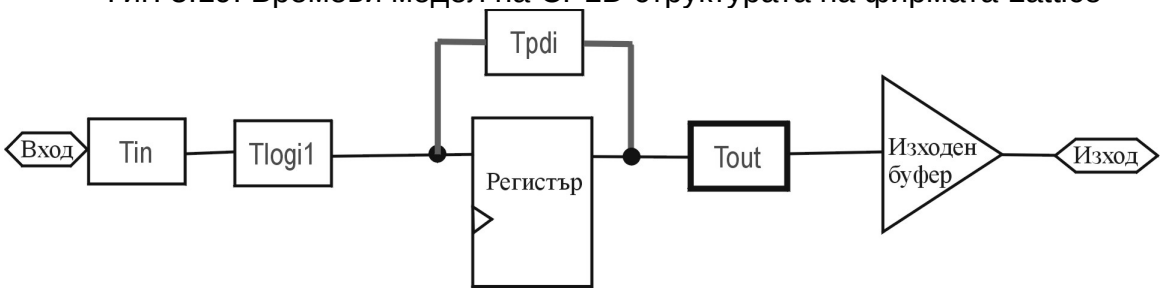


Фиг. 5.18. PAL структура (а) и PLA структура (б)

На фиг. 5.19 и фиг. 5.20 са показани времевите модели на CPLD структурите, представени от двете фирми. От диаграмата се вижда, че при Lattice, пътят T_{pdi} е по-кратък, което води до по-голямо бързодействие. Реалните проекти рядко биха се възползвали от това предимство. При реализация на синхронни структури, към пътят T_{pd} следва да се добави 1ns. Това увеличаване на общия път води до забавяне, което прави тези чипове по-бавни в сравнение със CPLD на Xilinx (CoolRunner II).



Фиг. 5.19. Времеви модел на CPLD структурата на фирмата Lattice



Фиг. 5.20. Времеви модел на CPLD структурата на фирмата Xilinx

- **Входно-изходни характеристики**

С добавянето на допълнителна поддръжка на интерфейсни стандарти, широка гама от компоненти биха могли да бъдат свързани по между си, без да се налагат допълнителни интерфейсни схеми. От друга

страна, добавянето на входен хистерезис позволява CPLD чиповете да обработват бавни аналогови сигнали. Това подобрява надеждността и намалява общия брой на използваните компоненти.

В таблица 5.3 е показано сравнение между CPLD структурите на фирмата Xilinx и фирмата Lattice.

Таблица 5.3 Сравнение между CPLD структурите на Xilinx и Lattice

CoolRunner-II	**ispMACH4000B/C**
Бързи входни регистри	Бързи входни регистри
Обратни връзки	Обратни връзки
Шинна организация	Шинна организация
Отворен дрейн	Отворен дрейн
LVCMOS 18 - 33	LVCMOS 18 – 33
1.5v I/O съвместимост	3.3v PCI съвместимост
HSTL1	Няма
SSTL 2-1,3-1	Няма
DataGATE	Няма
500mV Входен хистерезис	Няма
Програмируема “земя”	Няма

- **Вериги за разпределение на такта**

Наличието на вградени честотни делители и тригерни структури, превключващи по двата фронта на тактовият сигнал, оптимизират консумацията и предоставят възможности за допълнително ускоряване работата на проекта. Използвайки предимствата на CoolCLOCK структурата, заложена в CPLD чиповете на фирмата Xilinx, се постига запазване на честотните характеристики на чипа, като едновременно с това намалява консумацията.

В таблица 5.4 е показано сравнение между CPLD структурите на фирмата Xilinx и фирмата Lattice

Таблица 5.4 Ресурси за разпределение на такта при Xilinx и Lattice

CoolRunner-II	**ispMACH4000B/C**
Честотен делител (GCK2)	Няма
DualEDGE тригери за всяка макроклетка	Няма
CoolCLOCK	Няма

- **Защита на проекта**

Защитата на проекта е било винаги предимство в съвременните CPLD проекти. Разработването на методи за защита на повече от едно ниво, предлагат допълнителни възможности за подобряване на защитата, намалявайки на практика риска от нежелано прочитане на проекта до нулев.

В таблица 5.5 е показано сравнение между CPLD структурите на фирмата Xilinx и фирмата Lattice.

Таблица 5.5 Защита на проекта при CPLD структурите на Xilinx и Lattice

CoolRunner-II	**ispMACH4000B/C**
Разпределени 4 нива на сигурност	1 ниво

В таблица 5.6 е направено обобщено сравнение между параметрите на трите водещи в областта на CPLD структурите фирми.

Таблица 5.6 основни параметри при CPLD устройствата на Xilinx, Lattice и Altera

Параметър	**1.8V устройства**			**2.5V устройства**		
	Xilinx CoolRunner-II	**Lattice ispMACH4000C**	**Lattice ispMACH4000B**	**Lattice ispLSI 2000VL**	**Altera MAX7000B**	**Xilinx 9500XV**
Консумация в standby режим	<100 uA	2mA (256MC)	12mA (256MC)	45-178 mA	29-450 mA	14-92 mA
Консумация при 50 MHz (256MC)	15mA	20mA	30mA	205mA	254mA	150mA
T_{PD} (ns)/f_{MAX} (MHz)	3.5 / 333	2.5 / 400	2.5 / 350	5.0 / 180	3.5 / 303	3.5 / 287
$F_{MAX\ double\ rate}$	454 MHz	Не	Не	Не	Не	Не
Честотен делител	Да	Не	Не	Не	Не	Не
Тактуване по нарастващ и спадащ фронт фронт	Да	Не	Не	Не	Не	Не
Поддръжка на входно-изходни стандарти	LVTTL, LVCMOS, HSTL, SSTL	LVTTL, LVCMOS	LVTTL, LVCMOS	LVTTL, LVCMOS	LVTTL, LVCMOS, SSTL, GTL+	LVTTL, LVCMOS
Брой входно-изходни крачета	33-270	30-208	30-208	32-128	36-212	34-192
Входно-изходни банки (макс.)	4	2	2	1	2	2
Глобални тактуващи шини	3	4	4	3	2	3
Входен хистерезис (400 mV)	Да	Не	Не	Не	Не	Да
DataGATE	Да	Не	Не	Не	Не	Не
P-Term входове /макроклетки	56	80	80	20	32	90
Сигурност	4 нива	1 ниво	1 ниво	1 ниво	1 ниво	1 ниво
Chip Scale Package	Да	Не	Не	Да	Да	Да

5.1.7. Обобщение

- Логическият блок на фамилиите *MAX* на фирмата *Altera* използва логически разширител, удобен за изпълнение на функции с много произведения или регистри.
- *CPLD* схемите на *Xilinx* имат вградена логика за бърз пренос, която облекчава изграждането на аритметични устройства. Тези схеми разполагат с два типа логически блокове и позволяват оптимизиране на бързодействието и консумираната мощност.
- Вградените честотни делители и тригерни структури, превключващи по двата фронта на тактовият сигнал, оптимизират консумацията и предоставят възможности за допълнително ускоряване работата на проекта.
- Наличието на второ програмируемо поле, обуславя по-голяма гъвкавост при програмирането и по-голяма ефективност при използването на цялостния ресурс.
- С добавянето на допълнителна поддръжка на интерфейсни стандарти, широка гама от компоненти биха могли да бъдат свързани по между си, без да се налагат допълнителни интерфейсни схеми.
- Увеличаването на броя нива по отношение на защита на проекта повишава степента на защита, като свежда риска от копиране до нулев.
- Архитектурата на *AMD* обхваща *PAL* блокове, свързани с две нива на междусъединения.

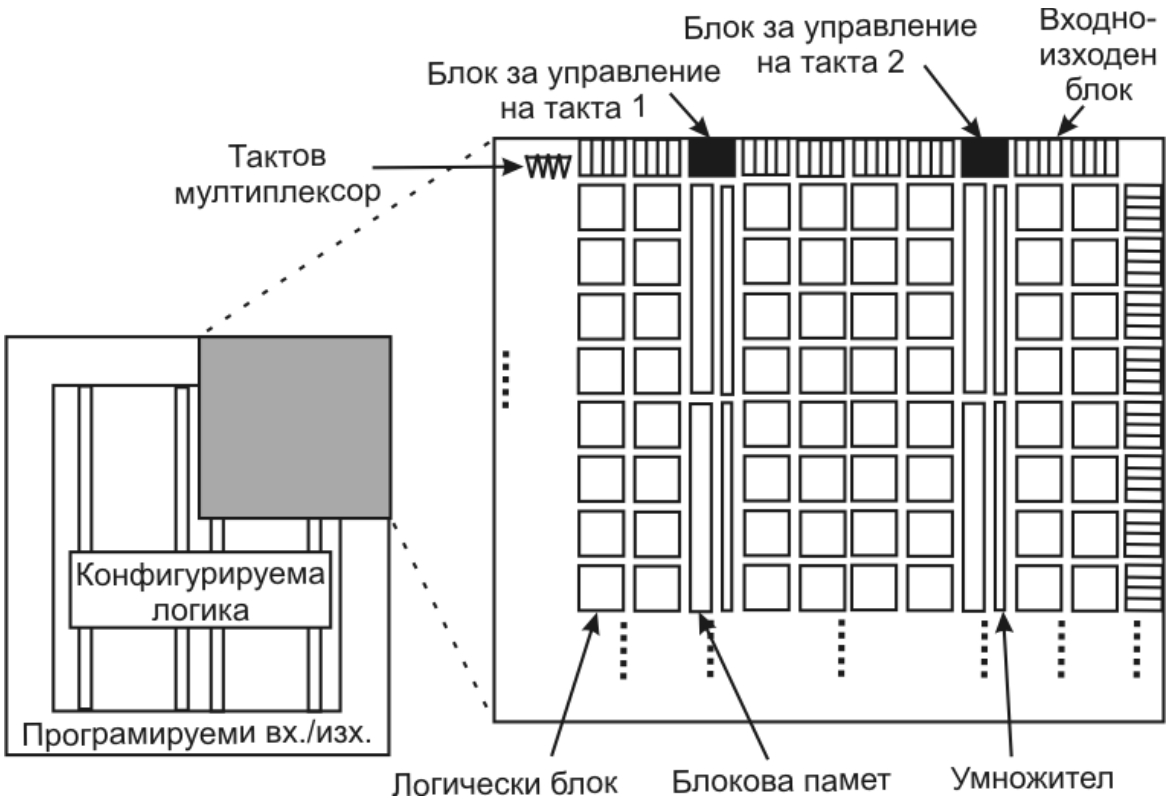
5.2. Програмируеми логически матрици (FPGA)

FPGA технологията е една от най-бързо развиващите се сфери в електронната индустрия. Това затруднява нейното обективно и безпристрастното представяне, тъй като то изисква отделните производители и продукти да се разглеждат в развитие. Тук се представят основните характеристики и особености на предлаганите продукти от водещите производители, както и техния капацитет (в потребителски логически елементи). Дадени са сравнителни оценки за логическите ресурси, типовете и количеството връзки за различните фамилии FPGA схеми.

Най-общо съществуват две категории FPGA, съответно базирани на *“SRAM”* и *“Antifuse”* програмируеми ключове. В първата категория лидери са фирмите *Xilinx* и *Altera* с основни конкуренти в лицето на фирмите Lucent Technology и Atmel. За втората категория основните производители са *Actel* и *Quicklogic*.

5.2.1. FPGA на фирмата Xilinx

FPGA чиповете, предлагани от фирмата Xilinx се разделят основно на две групи. Към първата група принадлежат фамилиите SpartanII, SpartanIIE, SpartanIII, а към втората – фамилиите Virtex.



Фиг. 5.21. Архитектура на фамилията Virtex-II

Характерно за двете групи е, че имат една и съща архитектура, като при Spartan базираните чипове, наличния ресурс е намален с цел реализация на чувствителни към цената устройства. Типичен представител на групата Virtex е Virtex-II. Фамилията Virtex-II представлява потребителски програмируеми масиви от логически елементи с разнообразни конфигуриращи елементи. Архитектурата на Virtex-II е оптимизирана за проекти с голяма степен на интеграция и с голяма скорост на работа. Както е показано на фиг. 5.21, програмируемият чип обединява входно-изходни блокове (IOB) и вътрешни конфигурируеми логически блокове (CLB).

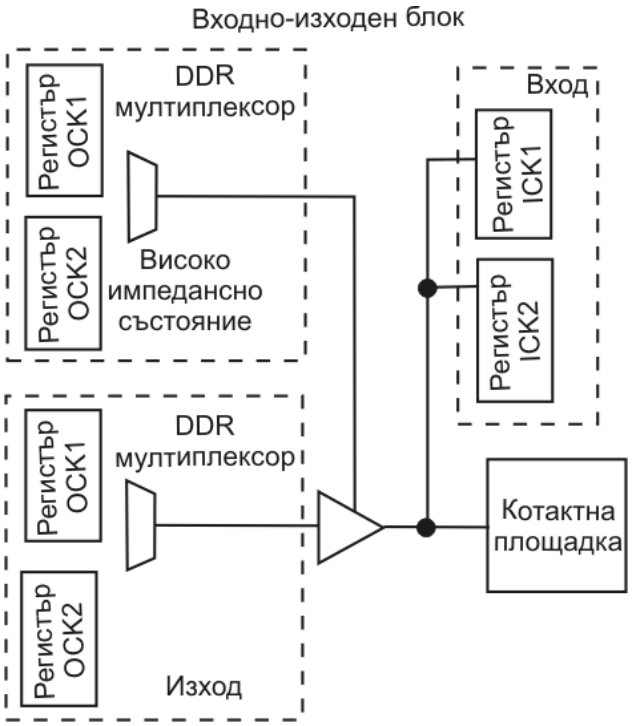
Програмируемите входно-изходни блокове осигуряват интерфейса между изводите на корпуса и вътрешната конфигурируема логика. Най-популярните и съвременни стандарти са поддържани от програмируемите входно-изходни блокове.

Вътрешната конфигурируема логика включва четири основни елемента, обособени в симетричен масив:

- *Конфигурируеми логически блокове* (КЛБ) – осигуряват функционални елементи за комбинационна и синхронна логика, включително основни запомнящи елементи. Буфери с високоимпедансно състояние присъединени към всеки КЛБ елемент управляват съответните сигнални магистрали.
- *Модули RAM памет* – осигуряват запомнящи елементи с обем 18-Кбита.
- *Умножаващи блокове* с размер на шината за данни 18 x 18 бита.

- *Блок за управление на тактовият сигнал* – осигурява самонастройка, напълно цифрово решения за компенсиране на закъснението при разпространение на цифровият сигнал, грубо и фино променяне на фазата на тактовият сигнал.

Ново поколение от програмируеми опроводяващи ресурси наречено “Active Interconnect Technology” свързва всички тези елементи. Главната опро-водяваща матрица е масив от опроводяващи ключове. Всеки програмируем елемент е свързан към превключваща матрица, позволяваща да се увеличат връзките към общата свързваща матрица. Взаимовръзките между елементите имат йерархична структура и са проектирани за високоскоростни проекти.



Фиг. 5.22. Структура на входно-изходния блок

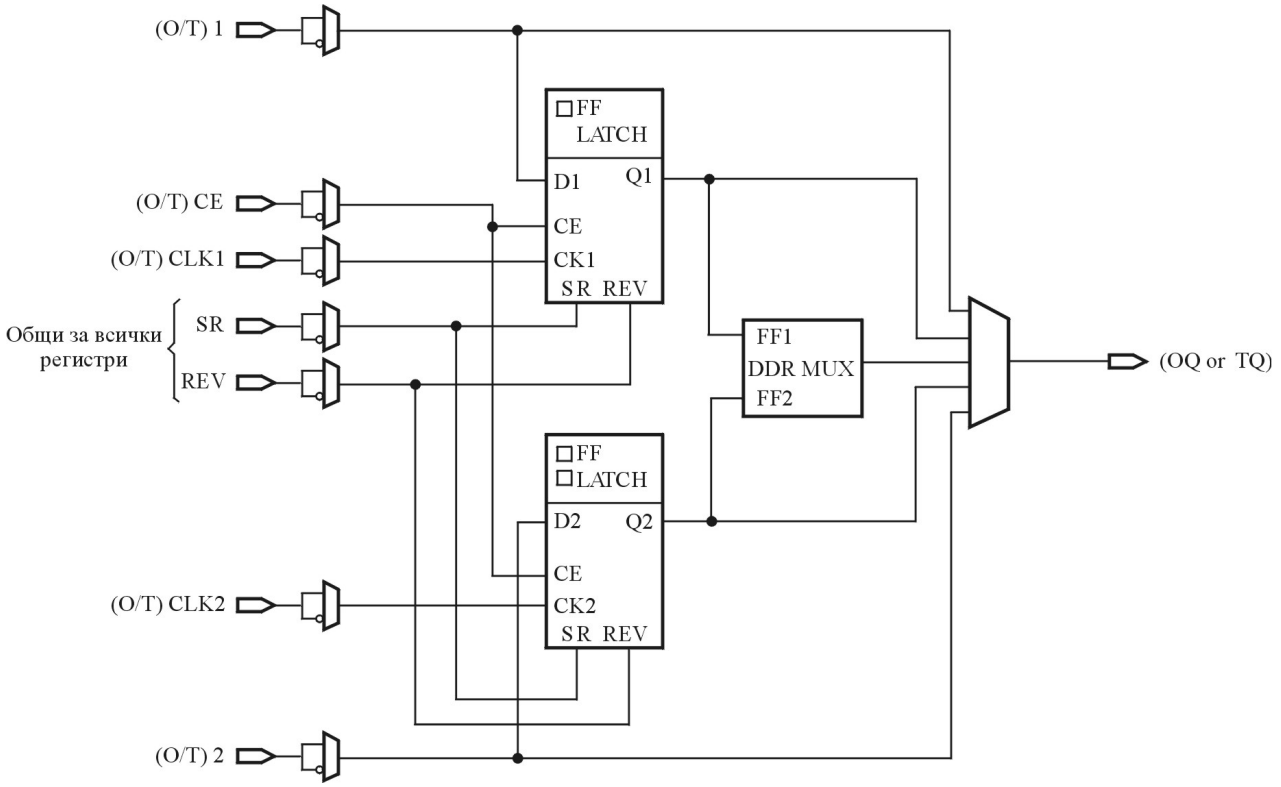
Всички програмируеми еле-менти, включително опрово-дяващите ресурси, са контролирани чрез стойности запаметени в клетки статична памет. Тези стойности се зареждат в клетките на паметта по време на конфигурирането и могат да бъдат презаредени така, че да се промени функцията на програмируемите елементи.

- **Входно-изходни блокове**

Входно-изходните блокове (ВИБ) във Virtex-II са обособени в групи по две или по четири по периферията на всяко устройство. Всеки ВИБ може да бъде използван като вход и/или изход за не диференциален вход-изход. Два ВИБ могат да бъдат използвани като диференциална двойка. Диференциалната двойка е винаги свързана към една и съща превключваща матрица, както е показано на фиг. 5.22.

Всеки запаметяващ елемент може да бъде конфигуриран като D-тригер управляван по фронт или като D-тригер управляван по ниво. На входа и на изхода могат да бъдат използвани един или два регистъра за данни с двойна скорост (DDR).

ВИБ включват шест запаметяващи елемента, както е показано на фиг. 5.23.



Фиг. 5.23. Структура на запаметяващия елемент

- **Конфигурируеми Логически Блокове (КЛБ)**

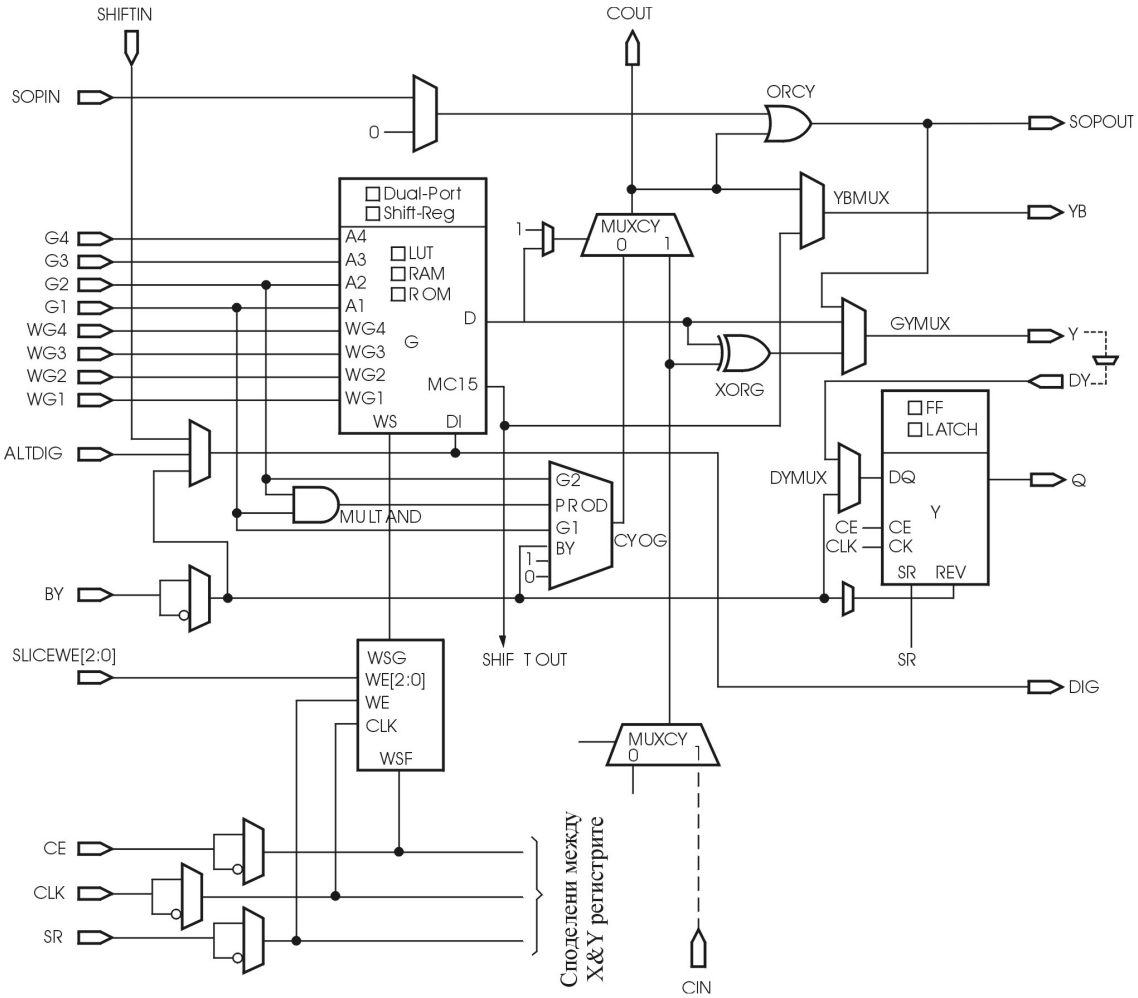
Ресурсите на КЛБ включват четири части (slice) и два буфера с високоимпедансно състояние. Всяка част е еквивалентна на останалите и съдържа:

- *два функционални генератора (F и G;)*
- *два запаметяващи елемента;*
- *аритметични и логически елементи;*
- *големи мултиплексори;*
- *обширни функционални възможности;*
- *верига за бърз пренос;*
- *хоризонтално свързана верига.*

Функционалните генератори F и G се конфигурират като четиривходова таблична логика (LUT), като шестнадесет битов преместващ регистър или като шестнадесет битова разпределена памет от типа SelectRam.

В допълнение, двата запаметяващи елемента са D-тригери управлявани по фронт или D-тригери управлявани по ниво.

Всеки КЛБ има вътрешни бързи междусвръзки и свръзка с превключваща матрица за достъп до главните опроводяващи ресурси.



Фиг. 5.24. Функционален генератор

Функционалните генератори на Virtex-II са изпълнени като четиривходова таблична логика. Четири независими входа са осигурени до всеки от двата функционални генератора в частта (F и G). Всеки от тези генератори има възможност да изпълни всяка булева функция с четири променливи. Поради това закъснението е независимо от изпълнената функция. Сигналите от функционалните генератори могат да излизат от частта (X или Y изход), може да им бъде променяна полярността (виж аритметична логика), могат да влизат в мултиплексора за пренос (виж логика за бърз пренос), да захранват D-входа на запаметяващ елемент или да отиват до входа на MUXF5 (фиг. 5.24).

В допълнение към базовата таблична логика, частите на Virtex-II съдържат логика (мултиплексорите MUXF5 и MUXFX), които комбинират функционалните генератори така, че може да се осигури всякаква функция от пет, шест, седем и осем променливи. Мултиплексора MUXFX е MUXF6, MUXF7 или MUXF8 в зависимост от частта в КЛБ. Избрана функция до девет променливи (мултиплексора MUXF5) може да бъде изградена в една част. Мултиплексора MUXFX може да бъде също MUXF6, MUXF7 или MUXF8, за да се осигури функция на шест, седем или осем променливи.

- **Блокова памет (Block SelectMap Memory)**

Блоковата памет има ресурс от 18 Кбита, в различни конфигурации от ширина и дълбочина, програмируема от 16К x 1 до 512 x 36 бита, а също и като двупортова RAM памет. Всеки порт е напълно синхронен и независим с възможност за три режима “четене по време на запис”. Поддържаните конфигурации за двупортова и еднопортова памет са показани в таблица 5.7.

Таблица 5.7 Поддържани конфигурации на бло**кова памет**

ROM	**Брой на логическите таблици**
16X1	1
32X1	2
64X1	4
128X1	8 (1 CLB)
256X1	16 (2 CLB)

Блокът за умножение е свързан с всеки блок памет. Той съдържа умножител 18 x 18 бита и е оптимизиран за операции базирани в съдържаната в блока памет. Умножителят може да бъде използван независимо от блока памет. Операциите четене/умножение/натрупване и структурите за цифрова филтрация са крайно ефективни.

Ресурсите на блоковата памет и умножителя са свързани към четири превключващи матрици, за да се осигури достъп до общите опроводяващи ресурси.

- Глобално тактуване

Блоковете за управление на такта (DCM) и буферите за мултиплексиране на общия тактов сигнал осигуряват завършено решение за проектиране на високоскоростни тактуващи схеми.

До 12 DCM блока са налични. За да се генерира външен или вътрешен тактов сигнал със стръмни фронтове, всеки DCM може да бъде използван за намаление на закъснението при разпространение на тактовия сигнал. DCM също осигурява фазово изместване на тактовият сигнал на 90, 180 и 270 градуса. Финото регулиране на фазовото изместване предлага възможност за преместване на 1/256 от периода на тактовият сигнал. Гъвкавият честотен синтез осигурява изходна тактова честота равна на всяко отношение M/N на входната тактова честота, където M и N са целочислени числа.

Устройствата Virtex-II имат 16 буфера за мултиплексиране на глобалният тактов сигнал, с осем тактови вериги на квадрант. Всеки буфер може да избере един или два тактови входа и да превключва от единият тактов сигнал към другия. Всеки DCM блок е способен да управлява до четири от шестнадесетте буфера за мултиплексиране на тактовия сигнал.

- Опроводяващи ресурси

Входно-изходните блокове, КЛБ, блоковата памет, умножителя, и DCM използват една и съща свързваща схема и един и същи достъп до глобалната опроводяваща матрица. Времевите модели са поделени, драстично подобрявайки предсказуемостта на поведението при високоскоростни проекти.

Има общо шестнадесет глобални тактови линии, като са налични по осем на квадрант. В добавка 24 вертикални и хоризонтални дълги линии на стълб или ред, а също вторично и локално опроводяване осигуряват бързи междусвързвания. Връзките във Virtex-II са буферирани, поради което са относително независими към капацитивни товари. Топологията е проектирана така, че да се минимизират паразитните влияния.

Хоризонталните и вертикалните опроводяващи ресурси за всеки стълб или ред включват:

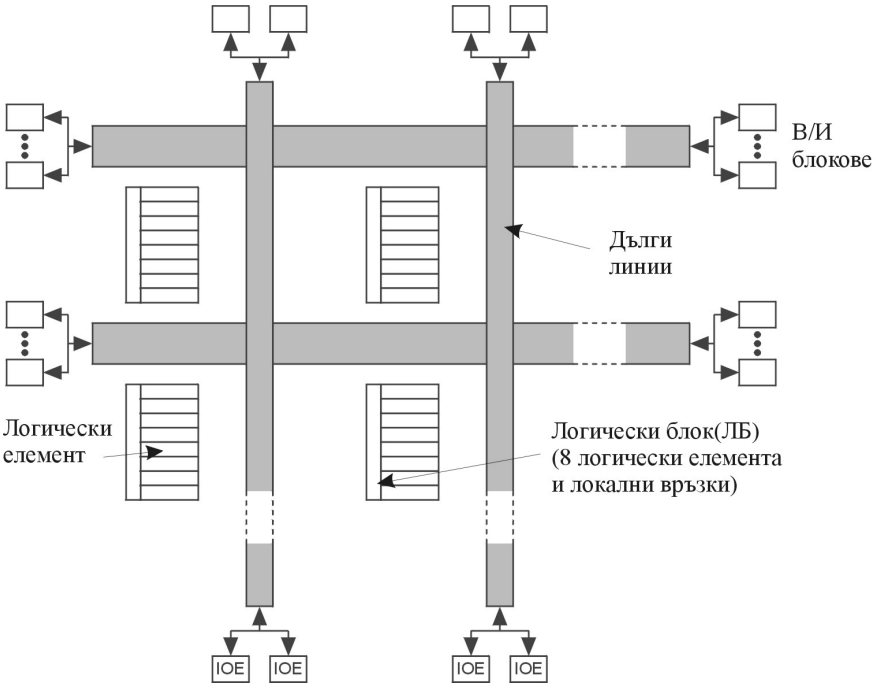
- 24 дълги линии;
- 120 шестнадесетични линии;
- 40 двойни линии;
- 16 директни свързващи линии (общо във всичките четири направления).

5.2.2. FPGA на фирмата Altera

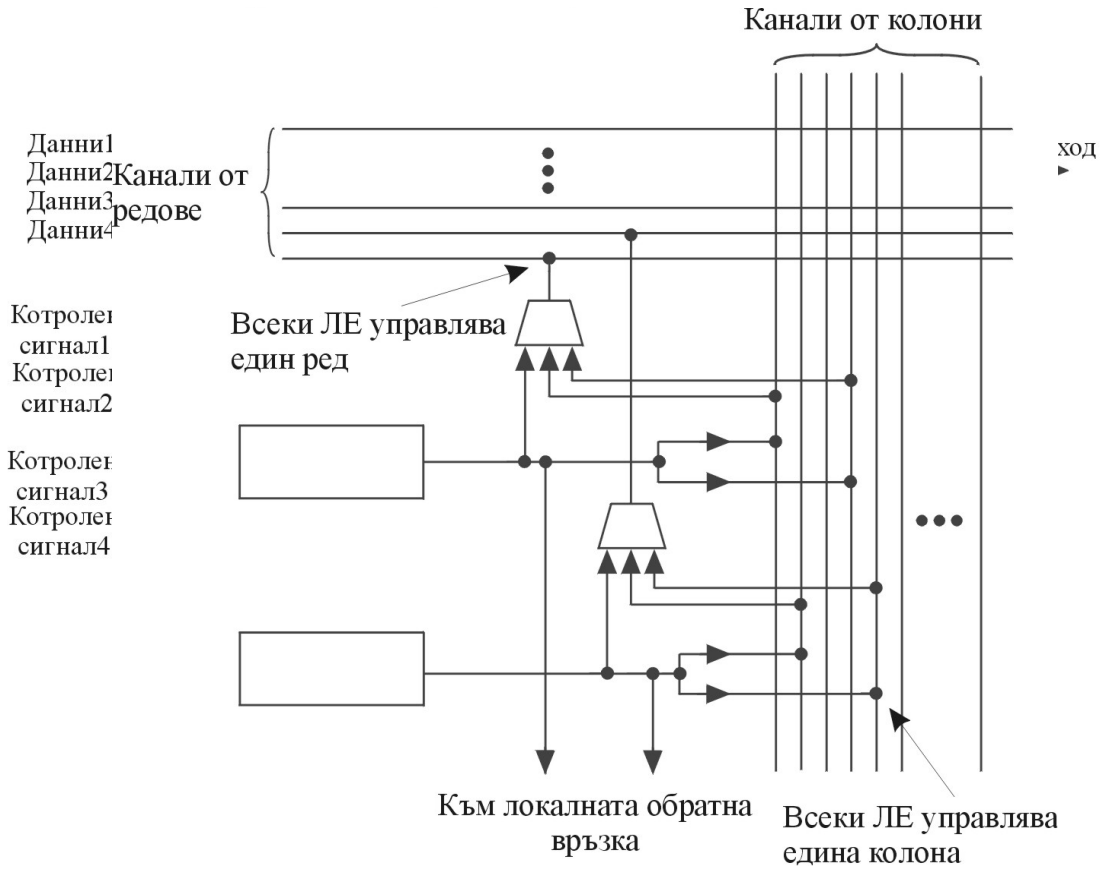
Архитектурата на фамилията FLEX8000 е показана на фиг. 5.25 и представлява комбинация между тази на FPGA и CPLD схемите. Фамилията притежава йерархична структура на три нива. Междинното ниво, подобно на CPLD схемите, представлява съвкупност от логически елементи (ЛЕ), групирани по 8 в логически блок (ЛБ). Броят на ЛБ варира от 26 до 162 за различните схеми, което предполага капацитет от 2 500 до 16 000 потребителски логически елемента.

Организацията на ЛБ е показана на фиг. 5.26. Той има локални връзки, които позволяват свързване на произволен вход и изход в рамките на ЛБ. Всеки ЛБ има четири общи управляващи линии, каскаден вход и изход, както и вход и изход за пренос. ЛБ са свързани към външните (дълги) линии, които позволяват реализиране на бързи връзки в рамките на цялата схема. Организацията на трасиращите ресурси, за разлика от тази на Xilinx, облекчава трасирането на връзките и прави схемите по-предсказуеми. Предсказуемостта на схемите се подпомага и от факта, че връзките между хоризонталните и вертикални дълги линии преминават през активни ключове (буфери), което води до намаляване на времето за разпространение на сигналите.

Организацията на ЛЕ е показана на фиг. 5.27. Базовият елемент е свръх бърза SRAM памет {16 x 1}, която позволява ЛЕ да реализира произволна функция на 4 променливи. Освен това ЛЕ съдържа един D-тригер с асинхронни “set” и “reset” сигнали, вградена логика за реализиране на ускорен пренос, както и каскаден елемент за реализиране на жично AND или OR логически функции.



Фиг. 5.25. Архитектура на фамилията FLEX8000



Фиг. 5.26. Организация на логическия блок (ЛБ)

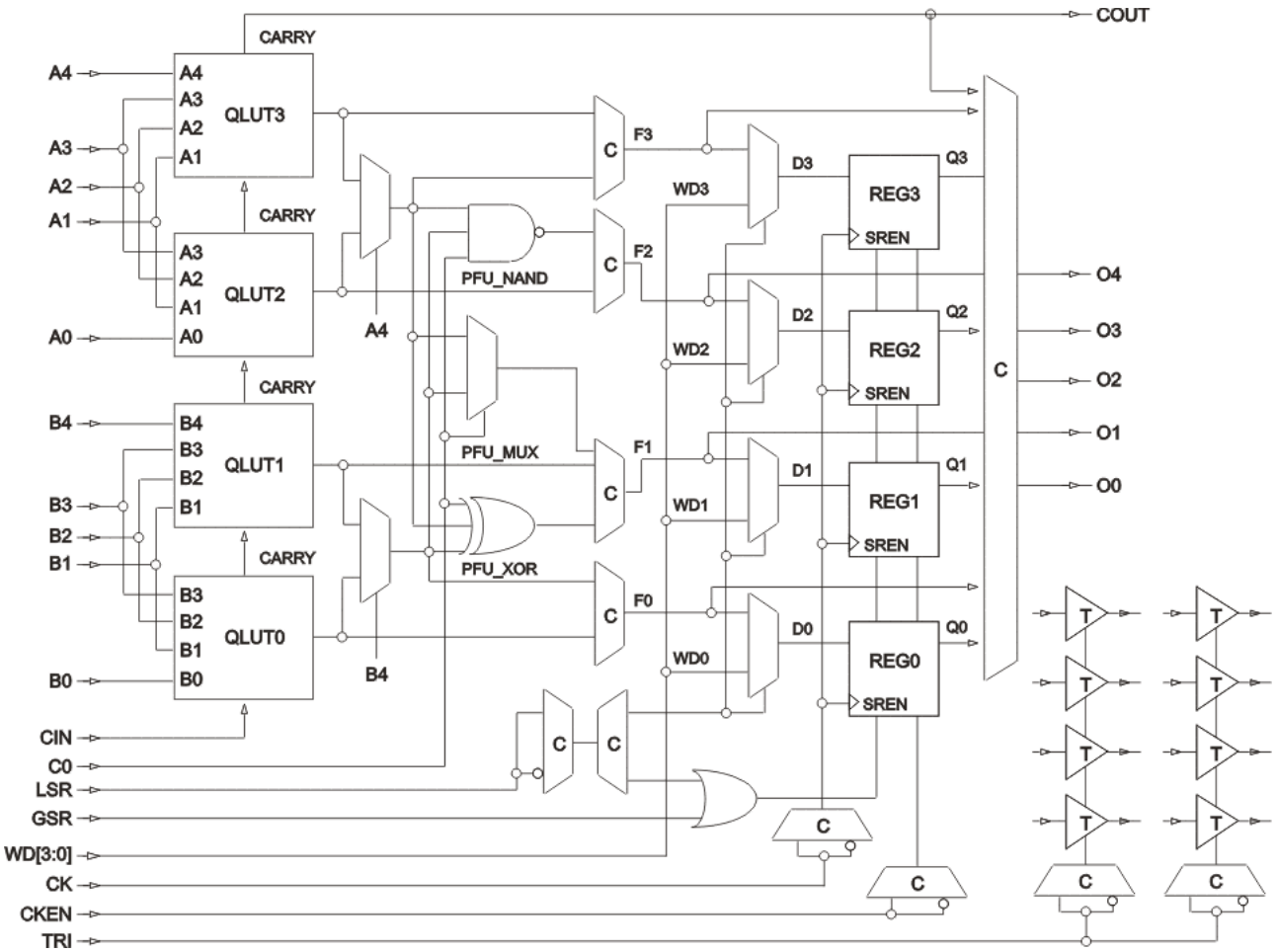
Архитектурата на фамилията FLEX8000 е подобрена при фамилията FLEX10K. Последната притежава всички възможности на

FLEX8000, плюс до 12 вградени блока памет (ВБП) във всеки ред на масива от ЛБ. Всеки ВБП представлява SRAM памет, която може да се конфигурира да работи като памет с произволен достъп и организация: {256 x 8}, {512 x 4}, {1K x 2} или {2K x 1}. Допълнително всеки ВБП може да се конфигурира да работи като комбинационна схема, която реализира умножение или друга функция, зададена чрез нейната таблица на истинност.

При FLEX10K са направени и някои подобрения в организацията на ЛЕ, по-съществените от които са наличието на вход за разрешение на тактовия сигнал на D-тригера, както и наличието на два отделни изхода, съответно към локалните връзки и към дългите линии, което позволява независимото използване на тригера. Фамилията FLEX10K също предлага по-богати трасиращи ресурси. Например, за най-малките масиви {3 x 24} те са по 144 и по 24 дълги линии за всеки ред и колона, докато при най-големите масиви {12 x 52} те достигат съответно до 312 и 24 дълги линии. Фамилията FLEX10K има капацитет до 160 000 потребителски логически елемента. Броят на тригерите и вх./изх. крачета достига съответно до 5 392 и 406.

5.2.3. FPGA на фирмата Lucent Technology (AT&T)

Архитектурата на фамилията ORCA (Optimized Reconfigurable Cell Array) е междинна и наподобява тази на FPGA схемите на фирмата Xilinx и Altera. Логическите блокове на ORCA се наричат програмируеми функционални блокове (ПФБ). Тяхната организация е показана на фиг. 5.28.



Фиг. 5.28. Програмируем функционален блок

ПФБ съдържа по 4 функционални блока (ФБ) – всеки с 4 входа, чрез които може да се реализира произволна логическа функция с 6 променливи, 2 функции с 5 променливи, или 4 функции с 4 променливи. Това се постига чрез наличието на общи входове между ФБ, което частично намалява гъвкавостта на ПФБ, но води до пестене на трасиращи ресурси. ПФБ на фамилията ORCA подобно на разгледаните продукти на фирмите Xilinx и Altera притежава вградена логика за ускорен пренос, както и възможност ПФБ да се конфигурира като еднопортова или двупортова RAM памет.

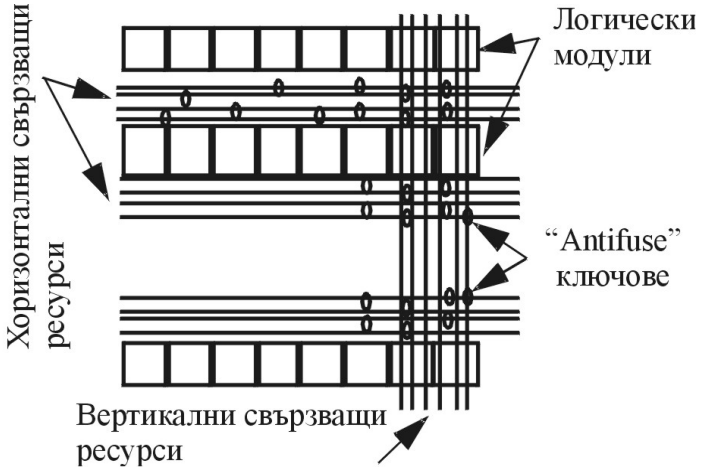
Трасиращите ресурси са организирани като 4-битови магистрали с активни ключове (буфери). Това осигурява постигане на по-висока ефективност и производителност при проектиране, тъй като типичните приложения често имат магистрална структура.

Архитектурата на фамилията ORCA е доразвита в сериите ORCA2, OR3C и OR3T, които имат по-висока степен на интеграция. ПФБ на тези серии е подобен на показания от фиг. 5.28, но има 8 ФБ всеки с 4 входа и 9 тригера, което му позволява да реализира 4 произволни логически функции с 5 променливи. Трасиращите ресурси са организирани в 9-битови магистрали с активни ключове. Борят на ПФБ е от 324 до 784, броят на тригерите е от 3 492 до 14 212, броят на вх./изх. крачета е от

292 до 612 и капацитетът е от 40 000 до 340 000 потребителски логически елементи или алтернативно до 185K RAM памет.

5.2.4. FPGA на фирмата Actel

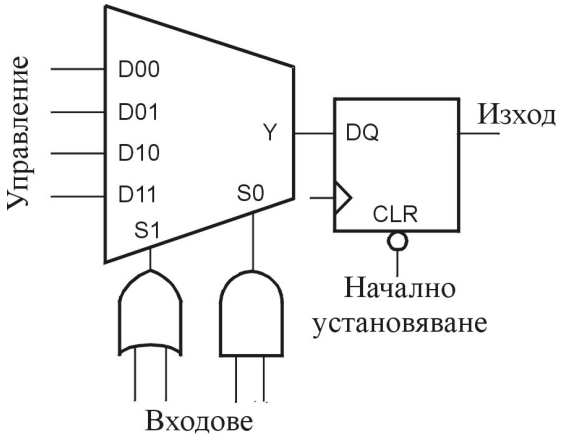
Фирмата Actel е най-големия производител на FPGA схеми на базата на “Antifuse” програмируеми ключове. Фирмата предлага три серии Act 1, 2 и 3, архитектурата на които е показана на фиг. 5.29. Логическите блокове са организирани по редове и разполагат с богати хоризонтални и вертикални ресурси за трасиране. Капацитетът на схемите е по-малък в сравнение с вече разгледаните FPGA фамилии, базирани на SRAM технологията. На фиг. 5.30 е показана организацията на логическия блок (ЛБ) на серията Acts. Той съдържа 2 двувходови логически елемента (AND и OR) и мултиплексор 4 към 1. Това позволява на ЛБ да реализира различни логически елементи с 5 входа. Около половината от наличните ЛБ допълнително съдържат D-тригер.



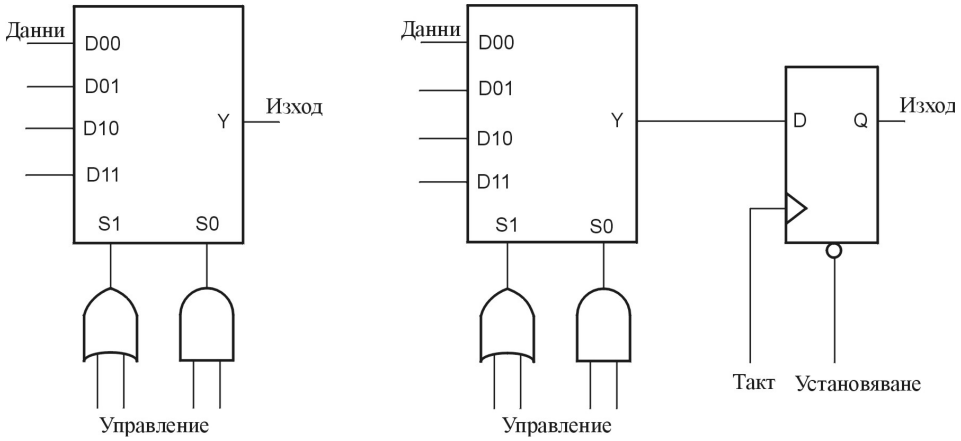
Фиг. 5.29. Архитектура на фамилията Act1,2 и 3

Трасиращите ресурси са организирани в хоризонтални канали, съдържащи сегменти с различна дължина, свързани чрез “antifuse” програмируеми ключове. В случай на връзка между ЛБ в различни редове на масива се използват и вертикалните трасиращи ресурси. Това води до непредсказуемост на Act схемите, тъй като закъснението на сигналите зависи от броя на използваните “antifuse” ключове за реализиране на връзката. Схемите разполагат с богат избор от сегменти с различна дължина и фирмата Actel е разработила ефективен алгоритъм, който стриктно гарантира постигане на зададено ограничение за броя на използваните “antifuse” ключове при трасиране на всяка връзка. Това води до увеличаване на производителността на схемите и ги прави по-предсказуеми.

Схемите от фамилията разполагат с до 20 вградени SRAM блока (256 бита) с време за достъп 5 ns, всеки един, от които може индивидуално да се конфигурира да работи като двупортова RAM с организация {32 x 8} или {64 x 4}. Броят на вх./изх. е до 250, капацитетът е до 40 000 потребителски логически елемента, а максималната тактова честота достига до 250 MHz.

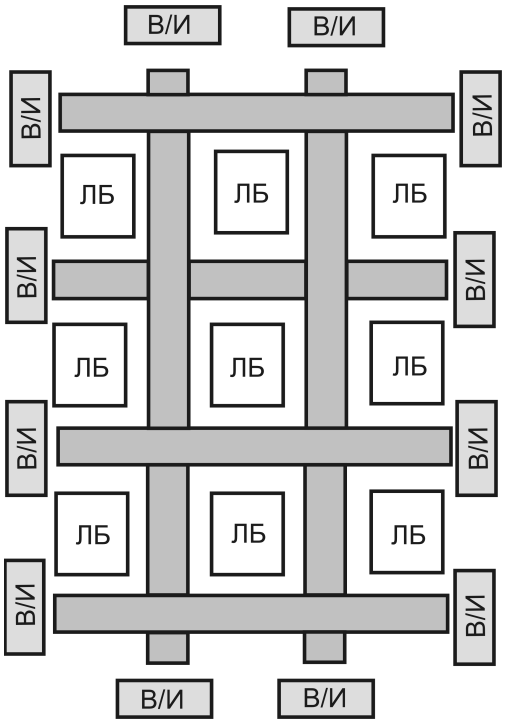


Фиг. 5.30. Структура на логическия блок



Фиг. 5.31. Организация на логическия блок на фамилията ES

Един от основните проблеми при “Antifuse” технологията е невъзможността наличните до 1 250 000 “antifuse” връзки, да се тестват от производителя, което води до това, че около 10% от схемите са неизправни след програмирането. Потребителят може да осигури тестово покритие около 75 + 95% чрез потребителско тестване на схемите, с което да намали процента на неизправните схеми до 2 - 5%. С вграждането на различни тест техники и режими за потребителско тестване този процес се облекчава значително и процентът на неизправните схеми се свежда до 0.1 - 1 %.



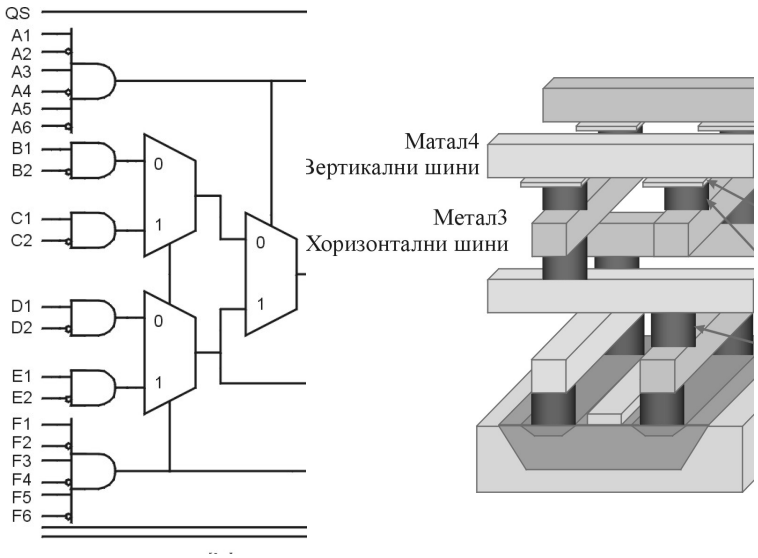
Фиг. 5.32. Архитектура на фамилията pASIC

От 1996 год. фирмата Actel предлага фамилията ES, базирана SRAM технология. Архитектурата на

логически блок на фамилията е показана на фиг. 5.31. Всеки логически блок (ЛБ) има три модула (функционални блока), чрез които може да реализира три произволни логически функции - 1 с два и 2 с три входа. Броят на ЛБ е до 16 384 като така наречените S-модули имат също D-тригер. Схемите имат до 32 вградени SRAM блока с обем 2К и време за достъп 6 ns, всеки един от които може да се конфигурира като двупортова RAM с различна организация. Трасиращите ресурси са организирани йерархично, като закъснението за различните нива варира от 0.8 до 12 ns. Броят на наличните вх./изх. крачета е до 640.

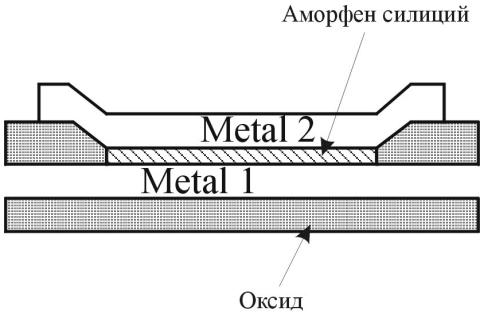
5.2.5. FPGA на фирмата Quicklogic

Основният конкурент на Actel в производството на FPGA схеми чрез “Antifuse” технология е фирмата Quicklogic. На фиг. 5.32 е показана архитектурата на фамилията pASIC, която е подобна на тази на фирмата Xilinx. Трасиращите ресурси включват само дълги линии, които в пресечните точки имат “antifuse” ключове.



а) б)

Фиг. 5.34. а) Логически блок на ASIC1 и pASIC2
б) Топология на трасиращите ресурси

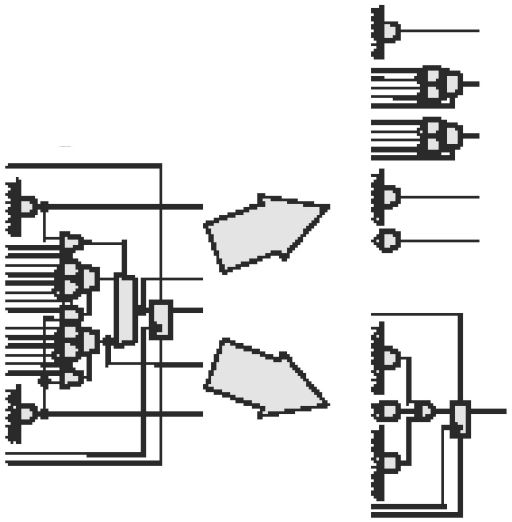


Фиг. 5.33. Antifuse разрез

Antifuse структурата на фирмата Quicklogic, наречена ViaLink, е показана на фиг. 5.33. Тя съдържа два метални слоя, разделени с изолиращ слой от аморфен силиций. След програмирането “ViaLink antifuse” ключовете имат съпротивление около 50 Ω. За сравнение съпротивлението на “PLICE antifuse” ключовете на фирмата Actel е около 300 Ω. ViaLink antifuse ключовете имат също около 5% по-ниско съпротивление от това на пас транзисторите и на транзисторите с плаващ гейт, използвани като програмируеми ключове съответно при SRAM и EEPROM технологията. “ViaLink” ключовете имат и по-нисък паразитен капацитет, както от “PLICE antifuse” ключовете, така и от SRAM и EEPROM технологията.

Логическият блок (ЛБ) на сериите pASIC1 и pASIC2 е показан на фиг. 5.34 а. Той е подобен на този на фирмата Actel, но със значително по-големи възможности и капацитет. Всеки ЛБ съдържа D-тригер с асинхронен set и reset.

Последният представител на фамилията pASIC е серията pASIC3. При нея организацията на ЛБ е подобрена (фиг. 5.35). Това позволява чрез един ЛБ да се реализират например един 16-входов AND елемент, два 6-входови и два 4-входови AND елемента, или два 6-входови AND елемента плюс два мултиплексора 2:1 или един мултиплексор 4:1. Тригерът може да се конфигурира да работи като D, T, JK или SR. Трасиращите ресурси са обогатени чрез използването на четири метални слоя, показани от фиг. 5.34 б.



Фиг. 5.35. Логически блок на pASIC3

В схемите от серията pASIC3 са вградени до 22 SRAM блока с 576 бита и време за достъп 5 ns, които могат да се конфигурират като двупортова RAM памет с различна организация. Броят на вх./изх. крачета е до 363, броят на тригерите е до 4 395, капацитетът на схемите е до 100 000 потребителски логически елемента, максималната тактова честота достига до 275 MHz. Последният показател е най-високия представен от всички разгледани производители.

Характерен за FPGA схемите на фирмата Quicklogic, както и при фирмата Actel, е проблемът с потребителското тестване. Той се състои в невъзможност за пълно тестване на схемите от производителя. Последното води до наличие на неизправни схеми след програмирането, откриването на които се извършва чрез потребителско тестване.

5.2.6. Сравнителен анализ

- *Ресурси* предлагани в чиповете на Xilinx и Altera
- ***Блокова памет***

Virtex архитектурите разполагат с реална синхронна двупортова RAM памет. Това дава възможност на проектантите за по-голяма гъвкавост, реализация на еднопортова памет, двупортова памет или памет с независими един от друг портове за четене и запис.

Flex10KE, предлаган като алтернатива от Altera, притежава класическа еднопортова блокова RAM памет. За реализация на

двупортова памет тук е небходимо да се използват два блока памет и допълнителна логика за управление на процесите при запис и четене.

- **Управление и обработка на тактовия сигнал**

Основни изисквания към веригите за обработка на такта са формиране и фазова корекция на входната поредица. Наличието на вериги за обработка на такта води до следните предимства:

- Премахване на нуждата от външни схеми за формиране на повече от една тактова поредица;
- Намаляване на цената на печатната платка, премахвайки нуждата от галванично разделяне на захранващата верига за DLL групата. Това изискване е задължително в случаите на PLL група.

Таблица 5.8. Сравнителна таблица на ресурсите на чиповете на Xilinx и Altera

Параметър	**Spartan-IIE**	**FLEX10KE ACEX1K**
Разпределена RAM	Да	--
Блокова RAM	Двупортова	Еднопортова
Управление на такта (PLL/DLL)	4	1
Входни/Изходни банки	До 8	1
Недиференциални входно-изходни стандарти	16 – LVTTL, LVCMDS3.3/2.5/1.8, PCI3.3V – 32/64 bit 33/66MHz, HSTL Class I,III & IV, AGP-2X, CTT, GTL, GTL+, SSTL2 Class I&II, SSTL3 Class I & II	5 – TTL, LVTTL, LVCMDS3.3/2.5, PCI3.3V – 32/64 bit 33/66 MHz
Диференциални входно-изходни стандарти	3 - LVDS, Bus LVDS, LVPECL	0
Прог. Товароносимост	Да	--
Споделени лог умножители	Да	--
Вградени BUS шини	Да	--
Преместващи регистри	Да	--

Таблица 5.9 PLL/DLL групи при чиповете на Xilinx и Altera

	Spartan-IIE	**FLEX10KE ACEX1K**
Начин на реализация	Цифров	Аналогов
Брой на PLL/DLL	4	1
Коефициент на умножение	1x, 2x	1x, 2x
Коефициент на делене	1.5, 2, 2.5, 3, 4, 5, 8, 16	--
Изместване на фазата	90, 180, 270 градуса	--

- **Поддръжка на входно-изходни стандарти**

Схемите от фамилията Virtex поддържат 19 входно-изходни стандарта. Това води до следните предимства:

- Реализация на диференциални и недиференциални стандарти;
- Реализация на високоскоростни интерфейси;
- Избягване на нуждата от външни приемо-предаватели;
- Опростяване проекта на ниво печатна платка;

- Намаляване на шума чрез използване на диференциални сигнали.

- **Организация на входно-изходните блокове**

Фирмата Xilinx предлага организация на разпределение в множество банки за своите чипове. Това води до следните предимства:

- Възможности за поддръжка на множество стандарти от един и същи интерфейс;
- Възможности за изпълнение на основни контролни функции в един комплексен проект;
- Надеждна системна връзка между множество различни специализирани схеми;
- Управление на вътрешна или външна памет, както и на системни интерфейси;
- Намаляване на необходимостта от използване на голям брой компоненти при разработката на проекти на ниво печатна платка;
- Създаване на условия за увеличаване на степента на интеграция и стабилност на реализираната печатна платка;
- Наличие на допълнителни функции на елементите, от които е изградена една LUT структура, което води до увеличаване на функционалността;
- Възможностите за реализация на проекти върху матрици разполагащи с по-комплексни логически структури водят до по-голяма гъвкавост при проектирането;
- Възможностите за използване на разпределени ресурси позволяват по-оптимална реализация на проекта в матрицата.

Таблица 5.10 Ресурси за реализирането на лог. операции при FPGA на Xilinx и Altera

	Spartan-IIE	**FLEX10KE ACEX1K**
Функция на LUT		
Капацитет на едно логическо ниво	4/5/6–входна функция (с използване на MUX5 и MUX6)	4-входна функция
Разпределена RAM	Да	--
Преместващ регистър	Да	--
Буфери с три състояния	Да	--
Аритметична логика		
Вградени вериги за пренос	2- Аритметични функции с до три операнда за едно логическо ниво	1- Аритметични функции с до три операнда за две логически нива
Ограничения на веригите за пренос	Няма	-Невъзможност за пресичане на блок паметта в средата на реда - Използване на допълнителни LAB при вериги по-дълги от 8

		логически елемента
Поддръжка на умножители	Да	--

5.2.7. Обобщение

- Блоковата памет дава възможност на проектантите за по-голяма гъвкавост, реализация на еднопортова памет, двупортова памет или памет с независими един от друг портове за четене и запис.
- Използването на DLL група води до намаляване цената на печатната платка, премахвайки нуждата от галванично разделяне на захран-ващата верига.
- DLL/PLL групите премахват нуждата от външни схеми за формира-
- не на повече от една тактова поредица.
- FPGA структурите поддържат множество стандарти чрез един и същи интерфейс.
- Този тип архитектури предоставят възможности за изпълнение на основни контролни функции в един комплексен проект, управление на вътрешна или външна памет, както и на системни интерфейси.
- Създават условия за увеличаване степента на интеграция и стабилност на реализираната печатна платка.
- Възможности за намаляване на шума чрез използване на диференциални сигнали.
- Те са надеждна системна връзка между множество различни специализирани схеми.
- Намаляване на необходимостта от използването на голям брой компоненти при разработката на проекти на ниво печатна платка.

Литература

1. Василева Т., *Автоматизация на проектирането на специализирани интегрални схеми*, Технически университет, София, 1997
2. Василева Т., В. Чумаченко, *Машинно проектиране на интегрални схеми и електронни възли*, Техника, София, 1999
3. Манолов Е., *Аналогови интегрални схеми. Схемотехника и проектиране*, Технически университет – София, София, 2002
4. Charles Sweeney, *Hardware Design Methodologies*, Celoxica Limited, 2002
5. Hastings A., *The Art of Analog Layout*, Prentice Hall, NJ 07458, 2001, ISBN 0-13-087061-7
6. Saint C., J. Saint, *IC Layout Basics – A Practical Guide*, McGraw-Hill, 2002, ISBN 0-07-138625-4
7. Geiger R., P. Allen, N. Strader, *VLSI Design Techniques for Analog and Digital Circuits*, McGraw-Hill, 1990, ISBN 0-07-023253-9
8. Allen P., D. Holberg, *CMOS Analog Circuit Design*, Oxford University Press, 1987, ISBN 0-19-510720-9
9. Gray P., P. Hurst, S. Lewis, R. Meyer, *Analysis and Design of Analog Integrated Circuits*, 4th ed., John Wiley & Sons, 2001, ISBN 0-471-32168-0
10. Weste N., K. Eshraghian, *Principles of CMOSVLSI Design – A System Perspective*, 2nd ed., Addison-Wesley, 1992, ISBN 0-201-53376-6
11. Johns D., K. Martin, *Analog Integrated Circuit Design*, John Wiley & Sons,1997, ISBN: 0-471-14448-7
12. Baker J., H. Li, D. Boyce, *CMOS Circuit Design, Layout, and Simulation*, IEEE Press, USA, IEEE Order Number: PC5689, 1998, ISBN 0-07-002469-3
13. Clein D., *CMOS IC Layout – Concepts, Methodologies, and Tools*, Newnes, 2000, ISBN 0-7506-7194-7
14. Grebene A., *Bipolar and MOS Analog Integrated Circuit Design*, John Wiley & Sons, 1984, ISBN 0-471-08529-4
15. Razavi B., *Design of Analog CMOS Integrated Circuits*, McGraw-Hill, 2000, ISBN 0-07-237371-0
16. Daly J., D. Galipeau, *Analog BiCMOS Design: practices and pitfalls*, CRC Press, 2000, ISBN 0-8493-0247-1
17. Perry D., *VHDL*, 3rd ed., McGraw – Hill Education, 1998, ISBN 0-0704943-3-9
18. Smith D., *HDL Chip Design*, Doone Publications, 1997, ISBN 0-9651934-3-8
19. Нанчева-Филиопова, Кр., М. Христов, В. Христов, И. Панайотов, *Използване на (v)HDL за синтез на електронен хардуер*, София, 2004, ISBN 954-9518-21-3
20. *Platform FPGAs Take on ASIC SoCs*, Xcell journal , Issue 43, Summer 2002, pp70-73

21. Cadence Online Documentation, Version 2.0, 2000-2002, Cadence Design Systems, Inc.
22. QuickLogic, Data Book CD, 2000
23. Xilinx Data Book, Third Quarter 2003, rev.8
24. Synopsys On-Line Documentation (SOLD) v 2003.3
25. Дончев Б., *Изследване и проектиране на специализирани цифрови интегрални схеми с програмируеми матрици*, Дисертационен труд, София, 2003
26. Михов Г. С., *Цифрова схемотехника*, Технически университет, София, 1998
27. Шойкова Е., С. Цанова, Д. Колев, И. Пандиев, *Методология за проектиране на електронни схеми с PSpice*, София, 2000, ISBN 954-9952-17-7
28. Шойкова Е., *Синтез на активни филтри*, София, 2000, ISBN 954-9952-19-3
29. Шойкова Е., И. Пандиев, *PSpice макромодели на операционни усилватели*, София, 2000, ISBN 954-9952-18-5
30. Holmes C., M. Willoughby, *VHDL Language Course* – Rutherford Appleton Laboratory, 1997
31. Армстронг Дж. Р., *Моделирование цифровых систем на языке VHDL*, Мир, 1992
32. Mazor S., P. Langstraat, *A guide to VHDL*, Kluwer Academic Publishers, 1992
33. Иванов Н., *Съвременни програмируеми прибори*, София, 2003
34. Гиздарски Е., *Проектиране с програмируема логика*, Авангард Принт ООД, Русе, 1998

WEB Sites:

Cadence Design Systems, Inc.: **http://www.cadence.com**

Synopsys documentation:
http://www.synopsys.com/support/support.html

Xilinx Data Sheets:
http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp

Altera Data Sheets: **http://www.altera.com/literature/lit-index.html**

Actel Application Notes:
http://www.actel.com/techdocs/appnotes/index.html

Actel Data Sheets: **http://www.actel.com/techdocs/ds/**

Quick Logic Documentations:
http://www.quicklogic.com/documentlistnew_.asp?sMenuID=212

Модули за дистанционно обучение:
http://ecad.tu-sofia.bg/education/courses.html

Методологии и среди за проектиране: **www.eedesign.com**

Съдържание

Увод 3

1. Методи за автоматизирано проектиране в електрониката.....5
1.1. Начини за представяне на интегрални схеми.....5
1.2. Етапи при проектирането на микроелектронни изделия.....6
1.3. Основни изисквания към системите за автоматизирано проектиране 10
1.4. Обобщение.....13

2. Въведение в CADENCE 14
2.1. Организация на базата данни в DFII.....14
2.1.1. Технологична библиотека.....17
2.1.2. Библиотеки в DFII.....17
2.1.3. Структура на библиотеките.....17
2.2. Въвеждане на електрическата схема.....19
2.3. Симулация на схеми.....21
2.4. Топологично проектиране.....24
2.4.1. Топологично проектиране на клетка.....24
2.4.1.1.Автоматично генериране на топология на стандартна клетка от представяне schematic в представяне layout.....25
2.4.1.2.Ръчно изчертаване на топология на клетка.....25
2.4.1.2.1. Напълно ръчно изчертаване.....26
2.4.1.2.2. Полу-ръчно изчертаване.....27
2.4.1.2.3. Автоматично изчертаване.....27
2.4.2.Проектиране на топология на интегрална схема.....27
2.4.2.1.IC Craftsman.....28
2.4.2.2.Silicon Ensemble (SE).....28
2.4.2.2.1. Импортиране на файловете.....30
2.4.2.2.2. Инициализиране.....30
2.4.2.2.3. Разполагане на входно-изходните стандартни клетки 31
2.4.2.2.4. Захранващи шини.....31
2.4.2.2.5. Разполагане на ядрото.....32
2.4.2.2.6. Глобално опроводяване.....33
2.4.2.2.7. Окончателно опроводяване.....33

3. Среда за проектиране Synopsys.....36
3.1. Инструменти за симулация.....36
3.1.1.Scirocco (VHDL симулация).....36
3.1.1.1. VHDL Analyzer.....36

3.1.1.2. Scirocco-i Compiler....36
3.1.1.3. VirSim....37
3.1.2.VCS™ 37
3.1.2.1. VCSi....37
3.1.2.2. VCS MX....37
3.1.3.CoCentric SystemStudio....37
3.2.Инструменти за синтез....38
3.2.1. Design Compiler....38
3.2.1.1.DC Professional....39
3.2.1.2.DC Expert....39
3.2.1.3.DC Ultra 40
3.2.1.4.DC Ultra Plus....40
3.2.1.5.DFT Compiler....40
3.2.1.6.Design Compiler FPGA....40
3.2.2. Design Analyzer....40
3.2.2.1.HDL Compiler 42
3.2.2.2.FPGA Compiler II 43
3.2.2.3.Behavioral Compiler 44
3.2.2.3.1.Входни данни на Behavioral Compiler 44
3.2.2.3.2.Синтез с Behavioral Compiler....45
3.2.2.3.3.Времеразпределяне 46
3.2.2.3.4.Разпределяне на хардуера 47
3.2.2.3.5.Определяне на пътя на данните и синтез на краен автомат. 47
3.2.2.3.6.Изходни данни на Behavioral Compiler 48
3.2.2.3.7.Архитектурен анализ с BCView™ 48
3.3.Инструменти за анализ....49
3.3.1.TimeMill 49
3.3.2.PrimeTime 50
3.3.3.PrimeTime SI 50
3.3.4.PathMill 50
3.3.5.PathMill Plus 50
3.3.6.AMPS 51
3.3.7.Arcadia 52
3.3.8.RailMill 52
3.3.9.PowerMill 52
3.3.10.PrimePower 52
3.3.11.Power Compiler....53
3.4.Инструменти за Layout....53
3.4.1.Apollo II 53
3.4.2.Columbia 53
3.4.3.JupiterXT 54
3.4.4.Astro 54
3.5.Продуктите Astro....54

3.5.1.Floorplan Compiler.....55
3.5.2.Mars-Xtalk 55
3.5.3.Mars-Rail 56
3.5.4.NanoSim 56
3.5.5.Saturn 56
3.5.6.Proteus 56
3.5.7.HERCULES 56
3.5.8.Star-RCXT 57
3.5.9.Star-MTB 57
3.5.10.Star-SimXT™ 57
3.5.11.Cosmos 57
3.6.Инструменти за проверка 57
3.6.1.Formality 57
3.6.2.Продукти на Test Compiler 58
3.6.2.1.TetraMAX 58
3.6.2.2.BSD Compiler 58
3.6.2.3.Vera 58
3.6.2.4.Magellan.....59
3.7.Инструменти за многократно използване и създаване на IP.....59
3.7.1.DesignWare 59
3.7.2.DesignWare Developer 59
3.7.2.1.Milkyway 60
3.7.2.2.Library Compiler.....60
3.8.Други инструменти.....64
3.8.1.SaberDesigner 64
3.8.2. Вътрешен език за хардуерно описание.....65

4. Език за описание на хардуер VHDL 66
Въведение.....66
4.1.Историческо развитие на VHDL 66
4.2.Интерфейсно и архитектурно тяло – ENTITY и ARCHITECTURE 66
4.3.Типове данни 67
4.3.1. Скаларни типове данни 67
4.3.2. Целочислени типове 68
4.3.3. Реални типове данни 68
4.3.4. Изброими типове 68
4.3.5. Физически типове 69
4.3.6. Съставни типове данни 70
4.3.6.1.Масиви 70
4.3.6.2.Масиви и константи 71
4.3.6.3.Записи 72
4.3.7.Типове ACCESS 72
4.3.8.Типове FILE 72

4.3.9.Подтипове 72
4.3.10.Конверсионен тип 73
4.4.Оператори 73
4.4.1.Логически оператори 74
4.4.2.Оператори за сравнение 74
4.4.3.Оператор IF 74
4.4.4.Оператор Case 74
4.4.5.Оператор NULL 75
4.4.6.Оператор Loop 75
4.4.7.Цикъл WHILE 75
4.4.8.Цикъл FOR 75
4.4.9.LOOP 76
4.4.10.Оператор EXIT и оператор NEXT 76
4.5.Изрази и оператори 76
4.6.Сигнали и променливи 76
4.7.Атрибути 77
4.8.Процедури и функции 77
4.8.1.Процедура 77
4.8.2.Функция 78
4.9.Краен автомат 78
4.10.Структурен VHDL 79
4.10.1.Интерфейсна част 79
4.10.2.Деклариране на компонент 79
4.10.3.Конфигурационна част 80
4.11.Тестови VHDL описания 83
4.12.Тестова установка 83
4.12.1.Стилове на VHDL тестовата установка 84
4.12.2.Тестова установка управлявана от файл 84
4.12.3.Командно ориентирана тестова установка 84
4.12.4.Самостоятелна тестова установка 84
4.13.Конфигурации 84
4.13.1.Прости конфигурации 84
4.13.2.Изрази за конфигуриране на архитектури 85
4.13.3.Конфигуриране на портове 86
4.14.Рангове, стойности по подразбиране и отворени портове....86
4.14.1.Рангове 86
4.14.2.Стойности по подразбиране 88
4.14.3.Несвързани портове 89
4.15.Пакети, библиотеки и конструктивни единици 89
4.15.1.Пакети 89
4.15.2.Пакети USE и VISIBILITY 91
4.15.3.Изрази за библиотека 91
4.15.4.Предварително дефинирани пакети 92
4.15.5.Конструктивни единици 92

5. Програмируеми логически устройства 94
5.1.Комплексни програмируеми логически устройства (CPLD).....94
5.1.1.CPLD на фирмата Altera 94
5.1.1.1.Фамилия MAX5000 95
5.1.1.2.Фамилия MAX9000.....97
5.1.2.CPLD на фирмата Advanced Micro Devices (AMD).....101
5.1.3.CPLD на фирмата Lattice 102
5.1.4.CPLD на фирмата Cypress 103
5.1.5.CPLD на фирмата Xilinx 104
5.1.6.Сравнителен анализ 109
5.1.7.Обобщение 113
5.2.Програмируеми логически матрици (FPGA).....113
5.2.1.FPGA на фирмата Xilinx 113
5.2.2.FPGA на фирмата Altera 119
5.2.3.FPGA на фирмата Lucent Technology (AT&T) 121
5.2.4.FPGA на фирмата Actel 123
5.2.5.FPGA на фирмата Quicklogic 125
5.2.6.Сравнителен анализ 126
5.2.7.Обобщение 129

Литература 130