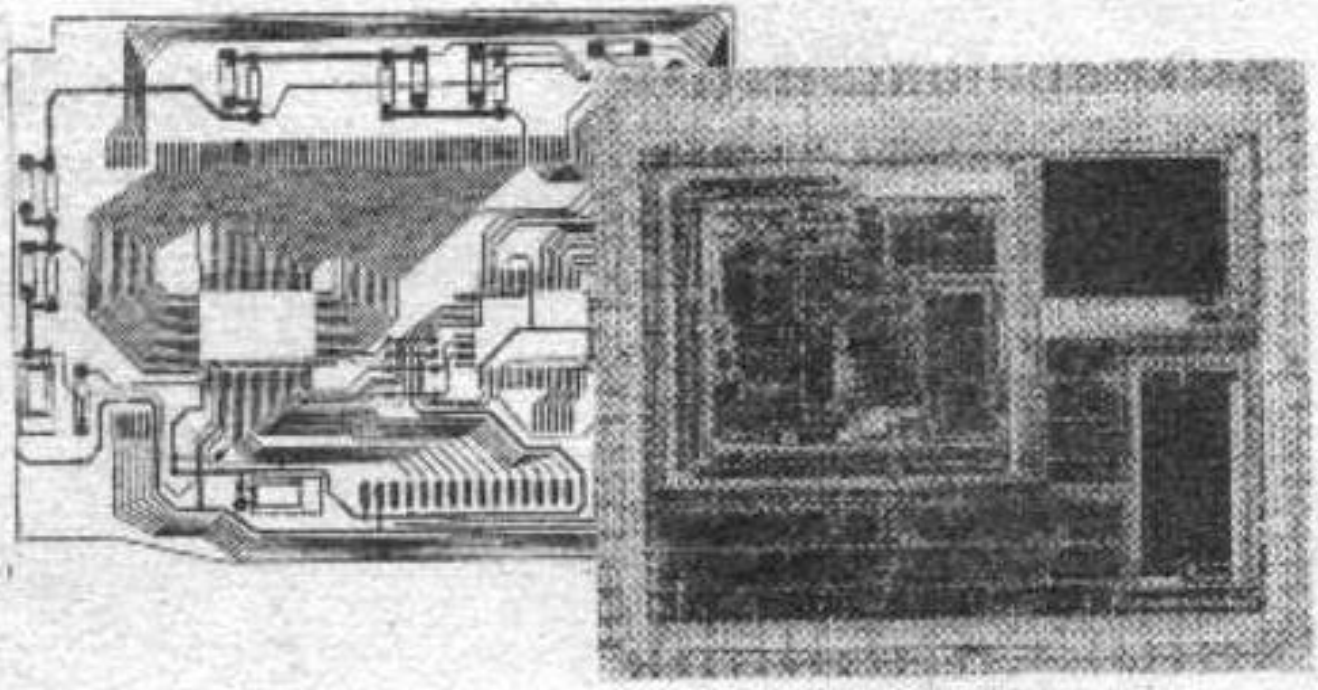
МАШИННО ПРОЕКТИРАНЕ НА ИНТЕГРАЛНИ СХЕМИ И ЕЛЕКТРОННИ ВЪЗЛИ

Ръководство за лабораторни упражнения

Доц. д-р инж. Таня К. Василева

Гл. ас. инж. Василий П. Чумаченко



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ

СОФИЯ, 1996

Целта на Ръководството за лабораторни упражнения по МАШИННО ПРОЕКТИРАНЕ НА ИНТЕГРАЛНИ СХЕМИ И ЕЛЕКТРОННИ ВЪЗЛИ е да запознае студентите с методологията на йерархичното проектиране чрез използване на съвременни средства за автоматизация. В него се разглеждат проблемите, които се решават на различните етапи от проектирането, както и специфичните средства, използвани на всеки един от тях. Упражненията са насочени към придобиване на практически опит за работа с VHDL като език за описание на апаратната част, с автоматичен синтез на логиката, логическа и аналогова симулация, автоматизирана разработка на топологията на интегрални схеми или печатни платки, както и със средствата за проверка на проекта и подготовката за производство на изделието.

Ръководството е предназначено за студентите от четвърти курс – профил Електронно уредостроене към Факултет по Електронна Техника и Технологии при Технически университет – София. То може да бъде полезно и на всички интересуващи се от проектиране на цифрови интегрални схеми и печатни платки с автоматизирани средства.

Средствата, техниката и програмното осигуряване, необходими за разработката и подготовката за отпечатване на настоящото ръководство бяха осигурени от проект TEMPUS JEN2605 към лаборатория "Симулационно Моделиране в Индустрията" към Технически Университет - София.

ISBN 954-438-179-1

Таня Крумова Василева
Василий Платонович Чумаченко

СЪДЪРЖАНИЕ

на база на логическото проектиране. Булевите уравнения се трансформират в електрическа схема като се вземат в предвид изискванията за бързодействие и консумация.

Физическо проектиране: Всички елементи от схемата и връзките между тях се трансформират в геометрично представяне. В резултат се получават послойните чертежи на топологията на схемата.

Проверка на проекта: Изследва се разработената топология за съответствие с правилата за проектиране и с изискванията на производството. След като се отстранят всички грешки, от топологията се извлича електрическата схема заедно с паразитните елементи и чрез ресимулация се проверява съответствието на физическата реализация със зададените изисквания за правилно функциониране.

Производство и тестване: След проверките топологията е готова за производство, затваряне в корпус и тестване.

Цикълът на проектиране на интегралните схеми обхваща множество итерации във всеки стадий и между отделните стадии. На всяка стъпка се създава ново представяне на системата, което се анализира и подобрява итеративно докато се изпълнят изискванията на спецификацията.

Целта на средствата за автоматизация, използвани в процеса на проектирането, е да минимизират броят на итерациите и да намалят времето за проектиране като обезпечат проект без грешки, чиято функционалност и параметри съответстват на изискванията на спецификацията.

В настоящото ръководство ще се разглеждат проблемите, които се решават на различните нива на абстракция при йерархичното проектиране в електрониката, както и автоматизираните средства, използвани на всеки един от тях, за да се проектира качествено изделие, изпълняващо изискванията на техническото задание.

Контролни въпроси

① Обяснете термините: йерархичност, описание на проекта, нива на абстракция.

② Дефинирайте типовете описания, използвани при проектиране.

③ Обяснете ролята на синтеза в процеса на проектиране.

④ Посочете етапите при проектиране на интегрални схеми и задачите, които се решават на всеки един от тях.

⑤ Каква е ролята на средствата за автоматизация при проектирането?

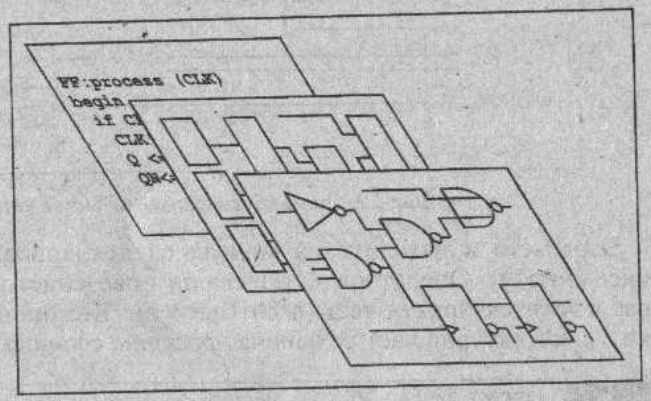
⑥ Начертайте диаграмата на Гайски като със стрелки посочите преходите за трансформация между отделните описания при синтеза.

МОДЕЛИРАНЕ НА ЕЛЕКТРОННИ СХЕМИ С VHDL

2

Ключови думи

- VHDL
- модел
- поведение
- структура
- симулация

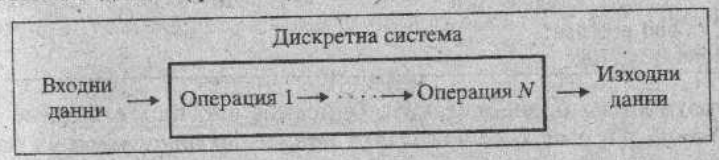


2.1 Основни концепции на езика VHDL

Езика VHDL е предназначен за описание и симулация на цифрови устройства. Той покрива множество нива на абстракция – от отделния логически елемент до цялостна хардуерна система. VHDL предоставя средства за описание, както на поведението така и на структурата на моделираната система.

2.1.1 Поведенчески модели

Поведенческия модел представлява функционална интерпретация на дадена система. Апаратната част на едно цифрово устройство може да се разглежда като *дискретна система*. Поведението на такава система се описва като набор от операции, прилагани върху преминаващите през системата данни (фиг. 2.1).

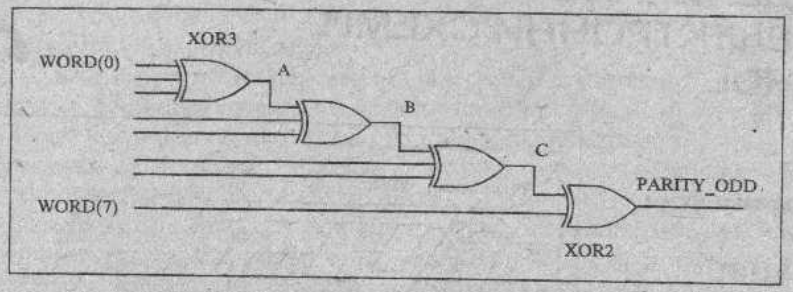


Фиг. 2.1. Дискретна система

При създаването на поведенчески VHDL модели операциите се описват с *процеси*, а връзките между тях – със *сигнали*. Процесите представляват програми съставени от последователни VHDL оператори. По време на симулация всички процеси се изпълняват едновременно.

На следния пример е показан поведенчески модел на схема за генери-

ране на бит за четност (фиг. 2.2). Изходният сигнал PARITY_ODD има стойност '0' когато броя на единиците е четен и '1' – когато е нечетен.



Фиг. 2.2. Схема за генериране на бит за четност

Входовете и изходите на модела са декларирани в интерфейсната част (entity). Операторите, описващи поведението на модела се намират в архитектурната част (architecture). Комбинацията от интерфейсна и архитектурна част се нарича *проектна единица*.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity PARITY is
   port (WORD : in std_logic_vector(0 to 7);
         PARITY_ODD : out std_logic);
end PARITY;
architecture BEHAVIOR of PARITY is
begin
   process (WORD)
      variable ODD: std_logic;
   begin
      ODD :='0';
      for i in 0 to 7 loop
         ODD := ODD xor WORD(i);
      end loop;
      PARITY_ODD <= ODD;
   end process;
end BEHAVIOR;
```

Разновидност на поведенческото описание е т. нар. описание тип *поток от данни*. При него се използват *паралелни оператори* за описание на поведението:

```
architecture DATAFLOW of PARITY is
begin
   PARITY_ODD <= WORD(0) xor WORD(1) xor WORD(2) xor WORD(3) xor
                 WORD(4) xor WORD(5) xor WORD(6) xor WORD(7);
end DATAFLOW;
```


2.1.2 Структурни модели

Структурният модел на една система отразява нейното декомпозиране на функционално свързани части. В терминологията на VHDL частите на системата се наричат *компоненти*. Връзките се осъществяват посредством сигнали, които влизат и излизат от компонентите през *портове*. Структурните VHDL модели са гъвкаво средство за описание на йерархията на проекта.

По-долу е показана архитектурната част на структурен модел на генератора за четност от фиг. 2.2. При описанието са използвани компонентите XOR3 и XOR2, чиито модели са предварително разработени.

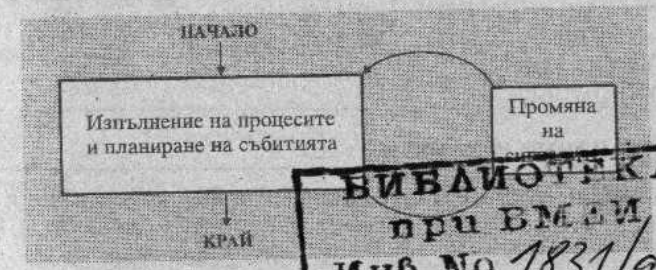
```
architecture STRUCTURAL of PARITY is
component XOR3
   port (I1, I2, I3 : in std_logic; O1 : out std_logic);
end component;
component XOR2
   port (I1, I2 : in std_logic; O1 : out std_logic);
end component;
signal A, B, C : std_logic;
begin
   U1: XOR3 port map (I1=>WORD(0), I2=>WORD(1), I3=>WORD(2), O1=>A);
   U2: XOR3 port map (I1=>A, I2=>WORD(3), I3=>WORD(4), O1=>B);
   U3: XOR3 port map (I1=>B, I2=>WORD(5), I3=>WORD(6), O1=>C);
   U4: XOR2 port map (I1=>C, I2=>WORD(7), O1=>PARITY_ODD);
end STRUCTURAL;
```

2.1.3 VHDL симулация

Преди да се симулират VHDL моделите трябва да бъдат компилирани и записани в *библиотека* (Приложение 1, фиг. 1).

VHDL симулаторите работят на принципа на *дискретно събитийно - моделиране*. Състоянието на модела може да бъде наблюдавано само на границите на *дискретни интервали* от време. Процесите в модела могат да променят изходите си само ако има промяна на някои от входните им сигнали. Промяната в стойността на сигнал се нарича *събитие*.

На фигура 2.3 е илюстриран цикличния процес на VHDL симулация.



Фиг. 2.3. Симулационен цикъл

Първа фаза. Изпълняват се процесите и се ***планират*** промените, които ще настъпят в изходните им сигнали.

Втора фаза. Извършват се планираните промени на сигналите и се определя кои процеси са засегнати от тях. Те ще бъдат изпълнени при следващия цикъл.

При VHDL симулацията винаги съществува закъснение между присвояването и промяната на стойността на даден сигнал.

За да се симулира и верифицира един VHDL модел е необходимо да бъде създадена *тестова установка*. Тя представлява проектна единица която включва в себе си тествания модел като компонент, подава към него входни въздействия, събира получените резултати и евентуално ги сравнява с зададени коректни стойности. Тестовата установка за разглеждания генератор на четност е следната:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity PARITY_TEST is
end PARITY_TEST;
architecture TESTBENCH of PARITY_TEST is
component PARITY
   port (WORD : in std_logic_vector(0 to 7);
         PARITY_ODD : out std_logic);
end component;
signal WORD : std_logic_vector(0 to 7);
signal PARITY_ODD : std_logic;
for all: PARITY use entity work.PARITY(BEHAVIOR);
begin
   UUT: PARITY port map (WORD => WORD, PARITY_ODD => PARITY_ODD);
   STM: process
   begin
      WORD <= "00000000"; wait for 10 ns;
      WORD <= "00000001"; wait for 10 ns;
      WORD <= "00000011"; wait for 10 ns;
      WORD <= "00000111"; wait for 10 ns;
      WORD <= "11111111"; wait for 10 ns;
   end process;
end TESTBENCH;
```

2.2 Типове данни и обекти

В езика VHDL всеки обект (например променлива или сигнал) притежава тип, който се задава при декларирането на обекта. VHDL е език със строга типизация, т.е. обект от определен тип може да приема стойност само от същия тип. Преобразуването от един тип в друг става

само в явен вид с помощта на специално написани за целта функции.

2.2.1 Обекти

Съществуват три класа обекти: *сигнали*, *променливи* и *константи*. Стойностите на *константите* се задава при деклариране им и след това не могат да бъдат променяни. *Променливите* в VHDL имат същия смисъл като и в езиците за програмиране. Те са локални за процеса, в който са декларирани. *Сигналите* са подобни на променливите, но съществува една принципна разлика. При присвояване сигнала не променя веднага стойността си, а се изчаква определено време. Продължителността на това изчакване се определя от стойността записана след думата after в оператора за сигнално присвояване. Ако такава стойност не е указана се приема, че сигналът ще се промени след един *делта интервал*, т.е. в следващия симулационен цикъл.

Общият формат за деклариране на обекти е следният:

клас списък-от-имена-на-обекти : *тип* := *стойност*;

Класът на обекта е constant, signal или variable. *Типовете* ще бъдат разгледани в следващия раздел. Полето *стойност* е задължително само за константите, за сигнали и променливи то има смисъла на начална стойност. Ето някои примери за деклариране на обекти:

```
variable FB,Q0,Q1 : std_logic;
signal CLK: bit := '0';
constant N: Integer := 32;
signal DOUT: std_logic_vector (31 downto 0);
```

2.2.2 Типове данни

VHDL предоставя няколко базови или *скаларни* типове данни и механизъм за конструиране на *съставни* типове. Скаларните типове включват *числови*, *физически* и *изброими* типове. Съставните типове са *масиви* и *записи*. Въз основа на базовите типове могата да бъдат дефинирани нови потребителски типове. В езика съществуват и няколко предварително дефинирани типове данни - bit, bit_vector, time, boolean и др. Техните декларации се съдържат в пакета standard.

Общия формат на оператора за деклариране на тип е:

type *име-на-тип* is *дефиниция-на-тип*;

Подобен на него е операторът за деклариране на *подтип* на вече създаден тип. В следващите раздели са дадени множество примери, поясняващи използването на тези оператори.

Целочислени типове

Декларацията на целочислен тип задава в какви граници се менят стойностите на обектите от този тип. В пакета standard е деклариран един целочислен тип – integer и два негови подтипа – на натуралните и на положителните числа:

```
type integer is range -2147483648 to 2147483647;
subtype natural is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;
```

Физически типове

Физическите типове са числови типове, служещи за представяне на физически величини като време, напрежение и т.н. Освен диапазон на допустимите стойности могат да се укажат и мерни единици, кратни на базовата. Важен пример за такъв тип е предварително дефинирания тип за време – time, който се използва при VHDL симулацията:

Изброими типове

Изброимият тип представлява подредено множество от символи и идентификатори.

```
type segments is ('a','b','c','d','e','f','g',dp);
type boolean is (false, true);
type bit is ('0','1');
```

Масиви

Масивите представляват индексирани набори от еднотипни елементи. Ако при декларирането на типа е указан диапазона на индекса на масива то такъв тип се нарича *ограничен*. В случай, че в декларацията на типа не са указани граници за индекса то типа е *неограничен*. Това позволява създаване на универсални типове данни, каквито са например предварително дефинираните string и bit_vector. Размерността на обектите от неограничен тип се указва при декларацията им:

```
type string is array (positive range <>) of character;
type bit_vector is array (natural range <>) of bit;
signal DATABUS : bit_vector (63 downto 0);
constant ERRMSG : string := "Simulation error";
```

Адресирането на елемент от масив се извършва като след името на масива в скоби се укаже индекса на елемента.

Записи

Както и в езици като Pascal и C, *записът* представлява структура от именовани елементи, които могат да са от различни типове. Обикновенно записите се използват при описания на по-абстрактните нива в йерархията на VHDL проекта. Адресирането на поле от запис се извършва като се укажат имената на обекта и полето, разделени с точка.

Атрибути

Атрибутите представляват допълнителна информация асоциирана със декларираните типове и обекти. Достъпа до атрибутите се осъществява като след името на типа или обекта се укаже името на атрибута и

като разделител се използва апостроф. Съществат множество предварително дефинирани атрибути. Тук ще бъдат разгледани само най-важните от тях.

За произволен скаларен тип T са дефинирани следните атрибути:

Атрибут	Резултат
T'left	Лява граница на T
T'right	Дясна граница на T

За всеки едномерен масив A са дефинирани следните атрибути:

Атрибут	Резултат
A'range	Диапазон на индекса на A
A'length	Дължина на диапазона на индекса на A

За всеки сигнал S са дефинирани следните атрибути:

Атрибут	Резултат
S'Event	true ако в текущия симулационен цикъл е променена стойността на S
S'Last_Value	Показва предпоследната стойност на S

Тип *std_logic*

Два от разгледаните до тук предварително дефинирани типове данни имат най-пряко отношение към моделирането на хардуер - bit и bit_vector. Обектите от тези типове могат да имат стойности '0' или '1' и служат за описание на логически сигнали и магистрали.

Когато двузначната логика не е достатъчна за адекватно описание на поведението на моделираната схема се използват типовете std_logic и std_logic_vector. Многозначната логика описава състоянията на възлите в схемата като комбинации от логически нива и сили. Типа std_logic е дефиниран със следния набор от стойности:

Стойност	Име на състоянието	Типично приложение
'U'	Неинициализирано	Стойност по подразбиране
'X'	Неопределено, силно	Конфликти в магистрали и др.
'0'	Силна нула	Транзистор към земя
'1'	Силна единица	Транзистор към захранване
'Z'	Висок импеданс	Изход на буфер с 3 състояния
'W'	Неопределено, слабо	Терминатор на шина
'L'	Слаба нула	Резистор към земя
'H'	Слаба единица	Резистор към захранване
'-'	Без значение	Автоматичен синтез на схеми

Тези типове, както и операциите със тях са декларирани в пакета std_logic_1164.

Оператори и изрази

Изразите в VHDL са подобни на изразите в традиционните езици за програмиране. Те представляват формули, които съдържат обекти, ли-

терали и извиквания на функции свързани помежду си с оператори.

Логическите оператори **and, or, nand, nor, xor** и **not** имат операнди от тип bit, std_logic или boolean и също така едномерни масиви от тези типове. Резултата е от типа, от който са и операндите.

Операторите за сравняване =, /=, <, <=, > и >= трябва да имат операдни от един и същ тип. Резултата е от тип boolean.

Аритметичните оператори +, –, * и / работят с операнди от целочислени, реални и физически типове.

В случай на необходимост могат да бъдат дефинирани нови оператори, а също така да се разширят дефиниционните области на вече съществуващи оператори.

2.3 Структурно описание

Структурното VHDL описание на една електронна схема описава от какви компоненти се състои тя и как тези компоненти са свързани помежду си.

2.3.1 Базови оператори за структурно описание

Този тип описание се базира на два оператора: оператор за деклариране на компонент и оператор за включване на компонент в схемата.

Следния пример представлява структурен VHDL модел на двубитов брояч на Джонсън. Схемата е съставена от два D-тригера – U1 и U2. Модела на тригера (DFF) е описан в друг файл и трябва да се компилира преди модела на брояча.

```
library IEEE;
use IEEE.std_logic_1164.all;
-- Интерфейсна част на проектната единица JOHNSON
entity JOHNSON is
   port (CLOCK, RESET : in std_logic;
         Q0, Q1 : inout std_logic);
end JOHNSON;
-- Архитектурна част на проектната единица JOHNSON
architecture STRUCTURAL of JOHNSON is
   component DFF -- Декларация на компонента D-тригер
      port (D, CLK, RSTN : in std_logic; Q, QN : out std_logic);
   end component;
   signal FB : std_logic;
begin
-- Включване на два компонента DFF с имена U1 и U2
   U1: DFF port map (FB, CLOCK, RESET, Q0, open);
   U2: DFF port map (Q0, CLOCK, RESET, Q1, FB);
end STRUCTURAL;
```

Оператора за деклариране на компонент задава имената и типа на портовете на компонента. Неговият формат е следния:

component *име-на-компонент*
  port (*списък-на-портовете*);
end component;

След като компонента е деклариран той може да бъде включен в описанието на проектната единица чрез следния оператор:

етикет : *име-на-компонент* port map *свързващ-списък*;

Етикетът е задължителен елемент от оператора за включване на компонент. *Свързващият списък* представлява списък на сигналите от проектната единица, към които е свързан компонента. Съответствието между тези сигнали и портовете на компонента се задава от местоположението им в *свързващия списък* и *списъка на портовете*. Ако даден порт трябва да остане несвързан на съответното място в *свързващия списък* се поставя думата open.

2.3.2 Регулярни стуктури

Много от електронните схеми съдържат повтарящи се еднотипни компоненти. За съкратено описание на такива регулярни структури се използва оператора generate. Формата на този оператор е следния:

етикет : *схема-на-генериране* generate
  оператори-за-включване-на-компоненти
end generate *етикет*;

Схемата-на-генериране може да бъде два типа: *for-схема* или *if-схема*.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity REG33 is
   port( DIN : in std_logic_vector (1 to 32);
         PTYIN, CLEAR, LOAD : in std_logic;
         DOUT : out std_logic_vector (1 to 32);
         PTYOUT : out std_logic);
end REG33;
architecture STRUCT_GEN of REG33 is
component DFF
   port (D, CLK, RSTN : in std_logic; Q, QN : out std_logic);
end component;
begin
G1: for N in 1 to 33 generate
   G2: if N < 33 generate
     U1: DFF port map (DIN(N), LOAD, CLEAR, DOUT(N), open);
   end generate G2;
   G3: if N = 33 generate
     U33: DFF port map( PTYIN, LOAD, CLEAR, PTYOUT, open);
   end generate G3;
end generate G1;
end STRUCT_GEN;
```

В зависимост от избраната схема оператора generate реализира итеративно или условно включване на компоненти. В горния пример е илюстрирано използването на generate за описание на 33-битов регистър. Разрядите от 1 до 32 са свързани към входната шина DIN и към изходната DOUT. Последният бит е свързан към вход PTYIN и към изход PTYOUT.

2.4 Поведенческо описание

В VHDL съществуват две нива, на които проектанта трябва да опише поведението на системата: последователно ниво и паралелно ниво. Последователното ниво включва програмирането на поведението на всеки един от процесите съставящи модела. Паралелното ниво включва дефинирането на връзките между процесите и по-специално комуникациите между тях. Връзката между тези две нива се осъществява от оператора process:

```
[етикет:] process [ (активиращ-списък) ]
    декларации
begin
    последователни-оператори
end process;
```

Последователните оператори описават поведението на процеса или с други думи действията, които се извършват когато процеса се изпълнява. Изпълнението на процеса настъпва в резултат на промяна на сигнали указани в *активиращия списък* или в оператора wait. Резултатите от изпълнението се отразяват в изходните сигнали на процеса. Всички процеси, които са *активни* в даден момен се изпълняват *паралелно* по отношение на симулационното време.

2.4.1 Последователни оператори

Последователните оператори могат да бъдат използвани в тялото на процес и при описание на функции и процедури. Те са подобни на операторите използвани в езиците за програмиране с изключение на оператора за сигнално присвояване и оператора wait.

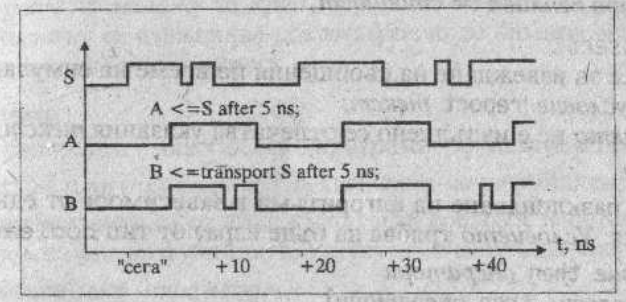
Оператор за сигнално присвояване

По време на VHDL симулацията за всеки сигнал се съхранява списък на бъдещите промени в стойността му. Този списък се нарича *драйвер* - на сигнала и се състои от *транзакции*. Оператора за сигнално присвояване добавя една или няколко транзакции към драйвера на дадения сигнал. Той има следния формат:

име-на-сигнал <= [transport] *стойност* [after *закъснение*];

Параметъра *закъснение* е опционален и показва след колко време трябва да настъпи промяната в стойността на сигнала. Ако този параметър е пропуснат то транзакцията се планира за следващия симулационен цикъл.

Съществуват два типа закъснения – инерционно и транспортно (фиг. 2.4). Транспортното закъснение е аналогично на закъснението при разпространение на сигнал по проводник. Инерционното закъснение служи за описание на активни компоненти чиито изходи не реагират на прекалено кратки входни въздействия.



Фиг. 2.4 Типове закъснения

Оператор *wait*

Операторът wait превключва съдържащият го процес в неактивно състояние. Също така той може да укаже условията, при които процеса отново ще стане активен.

Операторът wait не може да се използва при описанието на процес, за който в оператора process е указан *активиращ списък*.

Съществуват следните варианти на **wait**:

wait;
: Процесът няма да се активира до края на симулацията;

wait on *активиращ-списък*;
: Процесът се активира при промяна на сигнал от *активиращия списък*. Този вариант е еквивалентен на поставяне на активиращ списък в заглавния ред на оператора **process**;

wait until *условие*;
: Процесът се активира при изпълнение на зададеното *условие*;

wait for *време*;
: Процесът се активира след изтичане на указаното *време*.

Последните три варианта на оператора могат да се комбинират за формиране на по-сложни условия. На следващия пример е показан процес, който се изпълнява при промяна на сигналите A и B, но само ако по това време сигнала ENABLE е във високо ниво.

```
And_process: process
begin
   wait on A, B until Enable = '1';
   T <= A and B after 2 ns;
end process;
```

Оператор за присвояване

Това е класически оператор за даване на стойност на променлива:

име-на-променлива := *стойност*;

Оператор *assert*

Използва се за извеждане на съобщения по време на симулация.

assert *условие* report *текст;*

Ако *условието* не е изпълнено се отпечатва указания *текст*.

Оператор *if*

Служи за разклоняване на алгоритъма в зависимост от едно или няколко условия. *Условието* трябва да бъде израз от тип boolean.

if *условие* then *оператори*
[elsif *условие* then *оператори*]
[else *оператори*]
end if;

В следващия пример операторът **if** е използван за описание на тригер управляван по ниво:

```
DLATCH: process
begin
   if ENABLE = '1' then
      Q <= D after 10 ns;
      QN <= not D after 10 ns;
   end if;
   wait on D;
end process;
```

Оператор *case*

Оператора case определя кои оператори ще се изпълнят на базата на стойността на *управляващия израз*:

case *управляващ-израз* is
when *стойност* [| *стойност*] => *оператори*
...
[when others => *оператори*]
end case;

Управляващия израз трябва да бъде или от дискретен тип или да е едномерен масив. Операторите след others се изпълняват когато стойността на управляващия израз не съвпада с нито една от явно изброените *стойности*.

Оператор *loop*

Операторът loop се използва за организиране на цикли. Съществуват три негови разновидности:

loop *тяло-на-цикъла* end loop;
Безкраен цикъл. От него се излиза с **exit**;

while *условие* loop *тяло-на-цикъла* end loop;
Цикълът се изпълнява докато *условието* има стойност true ;

for *брояч* in *начало* to *край* loop *тяло-на-цикъла* end loop;
Цикълът се изпълнява докато *брояча* се намира между *началната* и *крайната* стойност.

Подпрограми

VHDL разполага с два типа подпрограми - функции и процедури:

function *име* (*параметри*) return *тип-на-резултата*;
Параметрите са от клас signal или constant и не могат да бъдат променяни от функцията. Резултатът се връща чрез името ѝ;

procedure *име* (*параметри*);
Параметрите са от клас signal, constant или variable. При декларирането им се указва дали са входни, изходни или двупосочни.

2.4.2 Паралелни оператори

Паралелните оператори представляват съкратен начин за описание на прости процеси. Тяхното използване прави VHDL моделите по кратки и прегледни.

Оператор за паралелно сигнално присвояване

По начина на записване този оператор не се отличава от последователния оператор за сигнално присвояване. Той е еквивалентен на процес с един единствен оператор за присвояване.

Например операторът

```
Y <= A and B after 2 ns;
```

е еквивалентен на

```
process (A, B) begin
   Y <= A and B after 2 ns;
end process;
```

Условен оператор за паралелно сигнално присвояване

Това е комбинация от оператор **if** и оператори за присвояване:

име-на-сигнал <= *стойност*1 when *условие*1 else
...
*стойност*N-1 when *условие*N-1 else
*стойност*N;

Пример:

```
MUX_OUT <= 'Z' after 2 ns when ENA='0' else
     IN_1 after 5 ns when SEL='1' else
     IN_2 after 5 ns;
```

Еквивалентният оператор **process** изглежда по следния начин:

```
process (IN_1, IN_2, ENA, SEL)begin
   if ENA='0' then MUX_OUT <= 'Z' after 2 ns;
   elsif SEL='1' then MUX_OUT <= IN_1 after 5 ns;
   else MUX_OUT <= IN_2 after 5 ns;
   end if;
end process;
```

Селективен оператор за паралелно сигнално присвояване

Това е комбинация от оператор **case** и оператори за присвояване:

with *управляващ-израз* select
име-на-сигнал <= *стойност*1 when *списък*1,
...
*стойност*N-1 when *списък*N-1;

```
with ALU_FUNCTION select
   RESULT <= OP1 + OP2 when ALU_ADD,
             OP1 - OP2 when ALU_SUB,
             OP1 and OP2 when ALU_AND,
             OP1 or OP2 when ALU_OR;
```

Еквивалентният оператор process би изглеждал по следния начин:

```
process (ALU_FUNCTION)begin
   case ALU_FUNCTION is
   when ALU_ADD => RESULT <= OP1 + OP2;
   when ALU_SUB => RESULT <= OP1 - OP2;
   when ALU_AND => RESULT <= OP1 and OP2;
   when ALU_OR  => RESULT <= OP1 or OP2;
   end case;
end process;
```

Контролни въпроси

① Каква е разликата между поведенчески и структурен VHDL модел?

② От какви части се състои проектната единица?

③ Обяснете разликата между типовете std_logic и bit.

④ В кои случаи се използва транспортен модел на закъсненията и в кои – инерционен?

⑤ По какво се различава сигналното присвояване от присвояването на стойност на променлива?

ТЕМА № 1

МОДЕЛИРАНЕ С VHDL

1. Цел

Придобиване на практически опит за създаване и симулация на VHDL проекти.

2. Задание

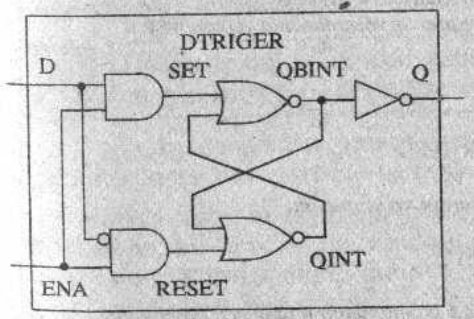
Да се довърши разработването на поведенчески VHDL модел на D-тригер. Да се провери коректността на модела посредством симулация.

3. Задачи за изпълнение

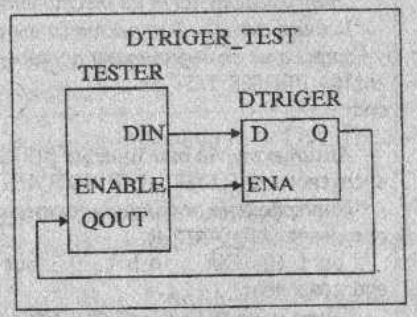
① Даден е файл с VHDL описание на D-тригер и *тестова постановка* за симулацията му. Запознайте се със съдържанието на този файл и се уверете, че разбирате смисъла на VHDL конструкциите в него.

Разглеждания тригер (фиг. 1) има вход за данни D, изход Q и управляващ вход ENA. При подаване на логическа единица на входа ENA в тригера се записва информацията от входа D.

Модела на тригера е описан в *проектната единица* DTRIGER. За този модел са разработени две *архитектури* – DATAFLOW и BEHAVIOR. Те са от тип *поведенческо описание*, но използват различни изразни средства. Архитектурата DATAFLOW е изградена с *конкурентни оператори за сигнално присвояване*, които описват преминаването на *потока от данни* през компонентите на схемата. В архитектурата behavior тригера е моделиран посредством *процес*, който се активира при промяна на сигналите ENA или D.



Фиг. 1. Схема на D тригер



Фиг. 2. Тестова постановка за симулация на D тригер

Тестовата постановка за проверка на модела на D-тригера е реализирана като проектна единица с име DTRIGER_TEST (фиг. 2). Тя включва в себе си компонента UUT_DTRIGER, на който съответства модела на тригера и процеса STM, който осигурява необходимите входни въздействия. Подаването на входни сигнали към модела се извършва *с последователни оператори за сигнално присвояване.*

```
-- Интерфейсна част на проектната единица DTRIGER.
entity DTRIGER is
    port (D, ENA : in bit; Q:out bit);      -- Входно / изходни сигнали.
end DTRIGER;

-- Архитектурно описание на DTRIGER - първи вариант.
architecture DATAFLOW of DTRIGER is
    signal SET, RESET, QINT, QBINT : bit; -- Вътрешни сигнали.
begin
    SET <= ENA and D;              --
    RESET <= ENA and not D;        -- Конкурентни оператори за
    QINT <= RESET nor QBINT;       -- сигнално присвояване.
    Q <= not QBINT;                --
end DATAFLOW;

-- Архитектурно описание на DTRIGER - втори вариант.
architecture BEHAVIOR of DTRIGER is
begin
    -- Процеса се активира при промени в ENA или Q.
    process (ENA, D) begin
        if ENA = '1' then          -- Последователни оператори.
            Q <= D;                --
        end if;
    end process;
end behavior;

-- Интерфейсна част на тестовата установка DTRIGER_TEST.
-- Всички сигнали в проекта са вътрешни за тестовата установка и
-- затова не са дефинирани портове.
entity DTRIGER_TEST is
end;

-- Архитектурно описание на DTRIGER_TEST.
architecture TESTER of DTRIGER_TEST is
-- Интерфейсна дефиниция на тествания компонент.
component UUT_DTRIGER
    port (D, ENA : in bit; Q : out bit);
end component;
-- Сигнали за връзка между тествания компонент и процеса генериращ
-- входните въздействия .
signal DIN, ENABLE, QOUT : bit;
```



```
-- Този оператор показва коя проектна единица ще бъде тествана.
for all: UUT_DTRIGER use entity work.DTRIGER(BEHAVIOR);
begin
    -- Установяване на връзка между портовете на тествания компонент и
    -- сигналите на тестера.
    UUT: UUT_DTRIGER port map (D=>DIN, ENA=>ENABLE, Q=>QOUT);

    -- Процес -генератор на входни въздействия (тестови вектори).
    STM : process begin
        -- Всеки вектор се формира с оператори за сигнално присвояване.
        DIN <='0'; ENABLE <='0' after 10 ns,'1' after 15 ns,'0'after 20 ns;
        -- Оператора wait осигурява подаването на тестовия вектор към
        -- компонента и изчаква установяването на изходите.
        wait for 50 ns;
        DIN <='1'; ENABLE <='0' after 10 ns,'1' after 15 ns,'0'after 20 ns;
        wait for 50 ns;
        DIN <='0'; ENABLE <='0' after 10 ns,'1' after 15 ns,'0'after 20 ns;
        wait;                          -- Край на симулацията.
    end process STM;
end TESTER;
```

② Симулирайте архитектурата BEHAVIOR на D-тригера и проверете адекватността на модела като наблюдавате резултатите от симулацията в графичен (прозорец **Wave**) и табличен вид (прозорец **List**). Симулацията да се извърши в продължение на 500 ns. Обърнете внимание, че промените на изходния сигнал QOUT настъпват веднага след подаване на високо ниво на разрешаващия вход ENABLE. Такова поведение е характерно за *функционалните* модели. При тях не се специфицира бързодействието на моделиранта схема и симулатора работи с нулеви закъснения (*делта закъснение* според терминологията на VHDL).

③ Променете входния VHDL файл, така че в тестовата постановна за модела на тригера да се използва архитектурата DATAFLOW. Повторно компилирайте и симулирайте проекта.

Преди всяка нова симулация не забравяйте да рестартирате симулатора с команда **File => Restart**.

Анализирайте резултатите в прозорците **Wave** и **List**. Има ли нещо нередно в поведението на сигнала QOUT?

④ Корегирайте входните данни, така че описанието на архитектурата DATAFLOW да съответства на схемата, показана на фиг. 1. Компилирайте и симулирайте корегираната схема.

Наблюдавайте резултатите в прозореца **List**. Обърнете внимание на това, че момента на настъпване на всички събития е 0 ns, а се увеличава само номера на *цикъла на симулация* (т.е. между събитията има "делта" закъснение). Подобно поведение е характерно за случаите когато моде-

ла на схемата започва да "осцилира" и е индикация за грешка в описанието. В дадения случай причината е в нулевото закъснение на логическите елементи и наличието на обратни връзки.

Добавете реални закъснения за логическите елементи (от порядъка на 2-5ns), компилирайте и симулирайте схемата до 500 ns и анализирайте резултатите.

Въпроси и задачи

① Да се разработи поведенчески модел на D-тригер, управляван по нарастващ фронт на тактовия сигнал CLK. Закъснението от фронта на тактовия сигнал до установяване на изхода Q да бъде 10 ns.

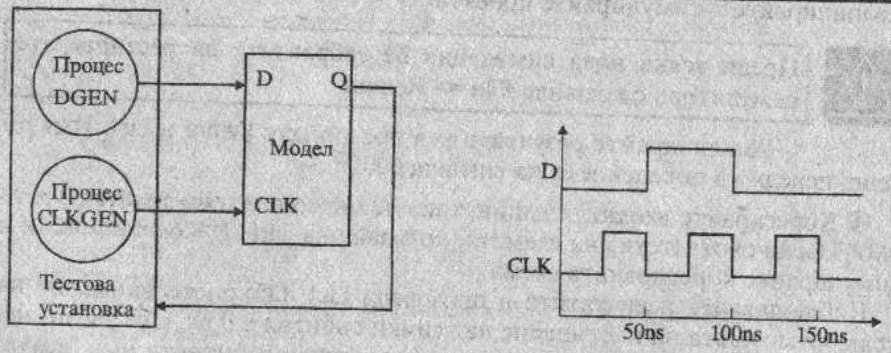
За да се провери дали е на лице нарастващ фронт на сигнала CLK се използва следният оператор:

```
if (CLK = '1' and CLK'event) then ... else ... end if;
```

② Да се разработи тестова установка за проверка на модела от предишната задача, съгласно схемата на фиг. 3. Входните въздействия да се подават от два отделни процеса: DGEN за данните и CLKGEN за тактовия сигнал. Времедиаграмите на входните сигнали са показани на фиг. 3.

Генерирането на тактовия сигнал може да се опише по следния начин:

```
CLK_gen : process
begin
   CLK <= not CLK;
   wait for 25 ns;
end process;
```



Фиг. 3. Тестова установка и входни въздействия (към задача 2)

ТЕМА № 2

ПРОЕКТИРАНЕ НА СТРУКТУРЕН VHDL МОДЕЛ НА ДВУБИТОВ СУМАТОР

I. Цел

Целта на упражнението е да се създаде модел на двубитов суматор.

II. Задание

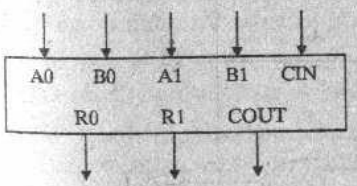
Да се разработи и верифицира описание на суматор, който събира две двубитови числа A и B и входен пренос CIN (фиг.1). Резултатът е двубитова сума S и изходен пренос COUT.

III. Задачи за изпълнение

① Попълнете таблицата на истинност на суматора. Сигналите A0 и A1 представляват съответно младшия и старшия бит на числото A. По аналогичен начин са означени и битове на числата B и S.

A1	A0	B1	B0	CIN	S1	S0	COUT
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
...							
1	1	1	1	1	1	1	1

② Запознайте се с интерфейстната част на модела на суматора, чийто символ е показан на фиг. 1.



Фиг. 1. Двубитов суматор

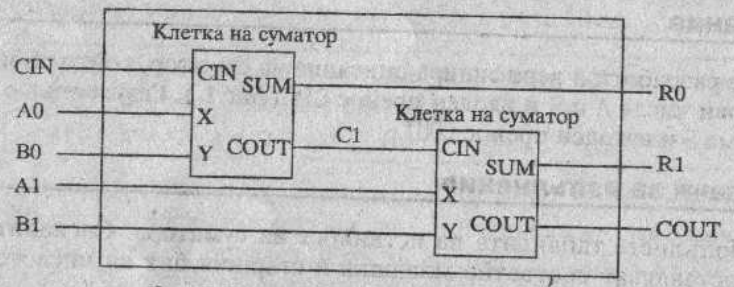
```
library IEEE;
use IEEE.std_logic_1164.all;

entity ADDER2 is
   port (A0, B0, A1, B1, CIN : in std_logic;
         S0, S1, COUT : out std_logic);
end ADDER2;
```

Обърнете внимание, че е използван тип std_logic, а не bit. Типът std_logic дава по-прецизно описание на състоянията на сигналите.

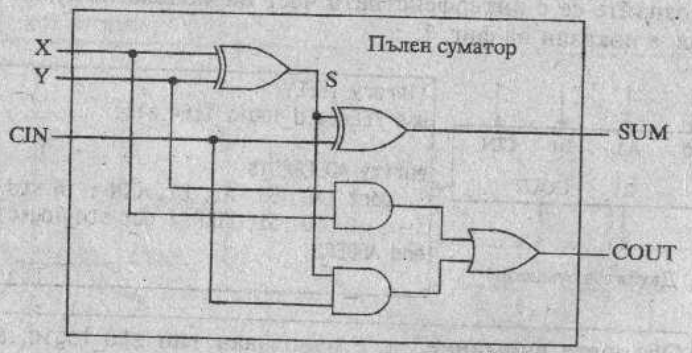
③ Запознайте се с архитектурната част на модела. При описанието на схемата на суматора, показана на фигура 2, е използван структурен подход.

```
architecture STRUCTURAL of ADDER2 is
   component ADDER1
      port (X, Y, CIN: in std_logic; SUM, COUT: out std_logic);
   end component;
   signal C1: std_logic;
begin
   U1: ADDER1 port map (X=>A0, Y=>B0, CIN=>CIN, SUM=>S0, COUT=>C1);
   U2: ADDER1 port map (X=>A1, Y=>B1, CIN=>C1, SUM=>S1, COUT=>COUT);
end STRUCTURAL;
```



Фиг. 2. Блокова схема на двубитов суматор

④ Съставете интерфейсната част на модела на еднобитов суматор (фиг. 3). Името на модела трябва да бъде ADDER1, а имената на портовете - както е указано на схемата. Тези изисквания се налагат поради това, че моделът ADDER1 вече е използван като компонент при описанието на двубитовия суматор.



Фиг. 3. Принципна схема на пълен суматор

⑤ Съставете архитектурната част на модела ADDER1, като използвате

схемата от фиг. 3. Използвайте описание тип *поток от данни*. Името на архитектурата да бъде dataflow. Закъсненията на елементите са както следва: изключващо ИЛИ – 5ns, И – 5ns, ИЛИ – 7ns. Запишете модела във файл с име ADDER1.VHD.

⑥ Разучете тестовата установка за симулация на модела ADDER2:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ADDER2_TEST is
end ADDER2_TEST;

architecture TESTER of ADDER2_TEST is
   component TESTED_MODEL
      port (A0, B0, A1, B1, CIN: in std_logic; S0, S1, COUT: out
std_logic);
   end component;
   signal A0, B0, A1, B1, CIN, S0, S1, COUT: std_logic;
   for all: TESTED_MODEL use entity WORK.ADDER2;
begin
   UUT: TESTED_MODEL port map
      (A0=>A0, B0=>B0, A1=>A1, B1=>B1, CIN=>CIN, S0=>S0, S1=>S1,
COUT=>COUT);
   STM: process begin
      A0 <='0'; A1 <='0'; B0 <='0'; B1 <='0'; CIN <='0';
      wait for 100 ns;
      A0 <='0'; A1 <='0'; B0 <='0'; B1 <='0'; CIN <='1';
      wait for 100 ns;
      A0 <='0'; A1 <='0'; B0 <='0'; B1 <='1'; CIN <='0';
      wait for 100 ns;
      A0 <='1'; A1 <='1'; B0 <='1'; B1 <='1'; CIN <='1';
      wait for 100 ns;
   end process STM;
end TESTER;
```

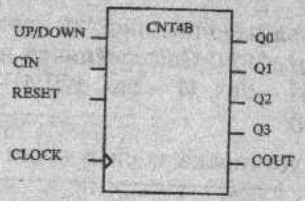
⑦ Добавете към тестовата установка още въздействия, които да изчерпят всички възможни комбинации от входни данни на суматора.

⑧ Компилирайте файловете ADDERTST.VHD, ADDER2.VHD и ADDER1.VHD. Има ли значение реда, в който се извършва компилацията?

⑨ Симулирайте архитектурата TESTER и проверете коректността на разработените модели. Определете бързодействието на суматора.

Въпроси и задачи

① Да се проектира четирибитов двоичен реверсивен брояч (фиг. 4). Типът и функциите на входните и изходните сигнали са следните:



Фиг. 4. Символ на брояч

- ◊ Q0 – Q3 – Изходи. Показват състоянието на брояча и се изменят от 0000 до 1111.
- ◊ CLOCK – Вход за тактов сигнал. Превключ-ването на брояча става по нарастващия фронт;
- ◊ RESET – Вход за нулиране. При високо ниво на този вход броячът преминава в състояние 0000 синхронно с тактовия сигнал;
- ◊ UP/DOWN – Вход за управление на режима. При високо ниво на този вход броячът е в режим на събиране, а при ниско – в режим на изваждане;
- ◊ CIN – Входен пренос. Броячът променя състоянието си само когато на този вход е подадено високо ниво;
- ◊ COUT – Изходен пренос. Когато броячът преминава от състояние 1111 в състояние 0000 (режим на събиране) или от 0000 в 1111 (режим на изваждане) на този изход се установява 1 в продължение на един период на тактовия сигнал. CIN и COUT се използват за каскадно свързване на няколко брояча.

Да се напише структурно VHDL описание на брояча, като се използват следните готови комоненти:

```
component LIBXOR -- Сума по модул 2
        port (A, B: in std_logic; Y: out std_logic);
      end component;
component LIBDFF
-- D тригер, управляван по нарастващ фронт на тактовия сигнал CK.
        port (D, CK: in std_logic; Q, QZ: out std_logic);
      end component;
component LIBINV -- Инвертор
        port (A: in std_logic; Y: out std_logic);
      end component;
component LIBMUX21 -- Мултиплексор 2 към 1. При S=0 : a=>y, при s=1 : b=>y.
        port (A, B, S: in std_logic; Y: out std_logic);
      end component;
component LIBAN2 -- Двувходов "И"
        port (A, B: in std_logic; Y: out std_logic);
      end component;
```

Да се проектира тестова установка и чрез симулация да се докаже работоспособността на схемата.

ТЕМА № 3

ПРОЕКТИРАНЕ НА ПАРАМЕТРИЧЕН *N*-БИТОВ СУМАТОР

1. Цел

Целта на упражнението е да се придобият знания за създаване на параметризирани VHDL модели и практически умения за използването им в по-сложни проекти.

2. Задание

Да се разработи универсален модел на суматор с последователен пренос, който събира две *N*-битови числа A и B и входен пренос CIN. Резултатът е *N*-битова сума S и изходен пренос COUT. Да се провери правилното функциониране на модела чрез симулация.

3. Задачи за изпълнение

① Запознайте се с даденото по-долу описание на модела ADDERN и на тестовата постановка TESTBENCH. Обърнете внимание на използваните VHDL конструкции: масиви, функции, дефинирани от потребителя типове, оператор generate и др.

Интерфейсна част

Съгласно заданието, суматора трябва да има следните входно / изходни сигнали:

Сигнал	Функция	Разрядност	Посока
A	събираемо	N	вход
B	събираемо	N	вход
CIN	входен пренос	1	вход
S	сума	N	изход
COUT	изходен пренос	1	изход

Портовете A, B и S са от тип std_logic_vector. Това са едномерни масиви, чиито елементи са от тип std_logic. Размерността на масивите и респективно разрядността на описваните от тях сигнали се задава от параметъра N, дефиниран в оператора generic. Конкретната стойност на N се указва чрез оператора generic map в схемата, която ще използва модела ADDERN като компонент.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity adderN is
   generic(N : integer);
   port (a, b : in std_logic_vector(N downto 1);
         cin  : in std_logic;
         s    : out std_logic_vector(N downto 1);
         cout : out std_logic);
end adderN;
```

Архитектурна част

N-битов суматор с последователен пренос може да се конструира от N еднобитови суматора (фиг. 1). VHDL реализацията на подобен параметризиран модел се базира на оператора generate. Той е подобен на операторите за цикъл, но за разлика от тях описва повтарянето на компоненти в структурното описание на схема, а не повтарянето на действия във времето. Описанието на архитектурата на суматора е следното:

```
-- Структурна архитектура на N-битов суматор.
architecture STRUCTURAL of ADDERN is
-- Като градивен елемент се използва еднобитов суматор.
   component ADDER1
      port (X, Y, CIN : in std_logic; SUM, COUT : out std_logic);
   end component;
   signal C : std_logic_vector(N downto 0);
-- За еднобитовия суматор се използва архитектура DATAFLOW на
-- модела ADDER1.
   for all : ADDER1 use entity work.ADDER1(DATAFLOW);
begin
   C(0) <= CIN;
   COUT <= C(N);
-- Еднобитовия суматор се повтаря N пъти.
   GEN: for I in 1 to N generate
      ADD: ADDER1 port map
         (X=>A(I), Y=>B(I), CIN=>C(I-1), S=>S(I), COUT=>C(I));
   end generate;
end STRUCTURAL;
```

Модел на еднобитов суматор.

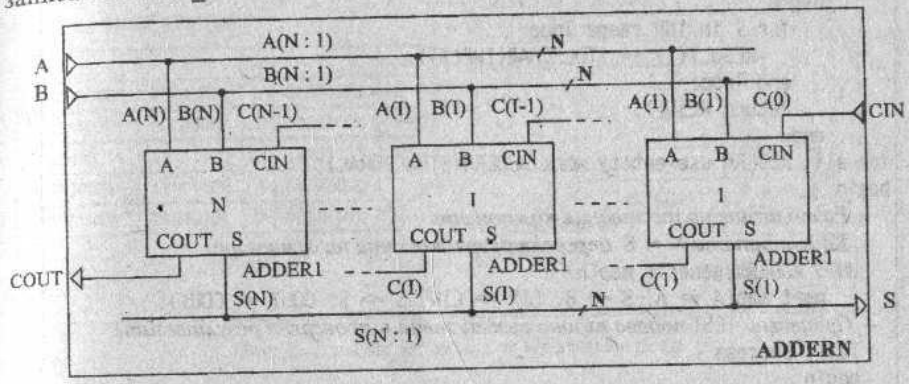
За еднобитовия суматор се използва модела ADDER1 от Тема 2.

Тестова постановка

За тестването на модела е реализиран 8-битов суматор. Той е включен като компонент в тестова установка с име TESTBENCH (фиг. 2). Конкретната разрядност на суматора е указана в оператора generic map (N). Тестовите вектори са записи от тип TEST_RECORD. Те са организирани в масива TEST_VECTORS.

Самото тестване се извършва от процеса TEST. Той взема един запис

от масива TEST_VECTORS, подава го към суматора и след 100ns проверява дали получените сума и пренос съответстват на очакваните стойности, записани в TEST_VECTORS.



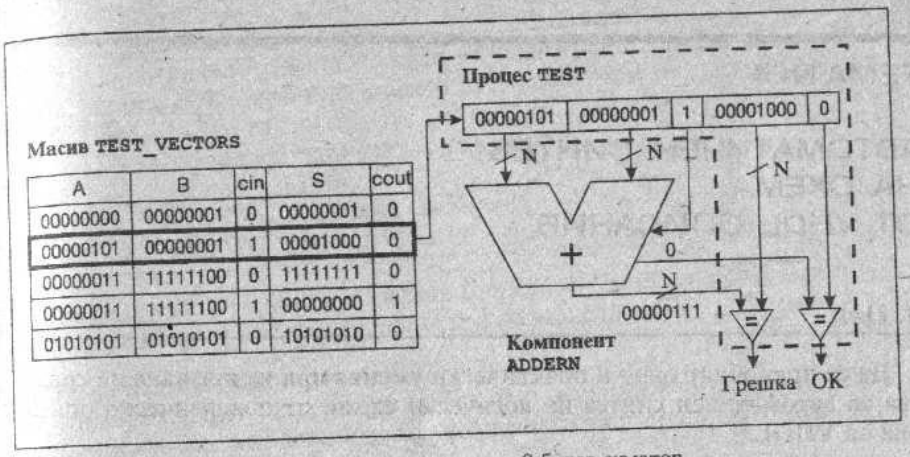
Фиг. 1. Схема на N-битов суматор с последователен пренос

```
library IEEE;
use IEEE.std_logic_1164.all;
entity TESTBENCH is end;
architecture ADDER8 of TESTBENCH is
component ADDERN
   generic(N : integer);
   port (A, B : in std_logic_vector(N downto 1); CIN : in std_logic;
         S : out std_logic_vector(N downto 1); COUT : out std_logic);
   end component;
   constant N : integer := 8;
   signal A, B, S : std_logic_vector(N downto 1);
   signal CIN, COUT  : std_logic;
   type TEST_RECORD is record
      A, B, S : std_logic_vector(N downto 1); CIN, COUT : std_logic;
   end record;
   type TEST_ARRAY is array(positive range <>) of TEST_RECORD;
-- Масива test_vectors дефинира тестовите вектори за суматора
   constant TEST_VECTORS : TEST_ARRAY := (
   (A=>"00000000", B=>"00000001", CIN=>'0', S=>"00000001", COUT=>'0'),
   (A=>"00000101", B=>"00000001", CIN=>'1', S=>"00001000", COUT=>'0'),
   (A=>"00000011", B=>"11111100", CIN=>'0', S=>"11111111", COUT=>'0'),
   (A=>"00000011", B=>"11111100", CIN=>'1', S=>"00000000", COUT=>'1'),
   (A=>"01010101", B=>"01010101", CIN=>'0', S=>"10101010", COUT=>'0'),
   (A=>"00000000", B=>"00000000", CIN=>'0', S=>"00000000", COUT=>'0'));
-- Масива STDL2CHAR се използва при преобразуването на std_logic в символи
   type STDLOGIC_TO_CHAR is array(std_logic) of character;
   constant STDL2CHAR : STDLOGIC_TO_CHAR := (
      'U' => 'U', 'X' => 'X', '0' => '0', '1' => '1', 'Z' => 'Z',
      'W' => 'W', 'L' => 'L', 'H' => 'H', '-' => '-');
```

```
  -- Функцията STDLV2STR преобразува std_logic_vector в стринг
   function STDLV2STR(INP : std_logic_vector) return string is
      variable RESULT : string(INP'range);
      begin
         for I in INP'range loop
            RESULT(I) := STDL2CHAR(INP(I));
         end loop;
         return RESULT;
      end;
for all: ADDERN use entity work.ADDERN(STRUCTURAL);
begin
  -- Разполагане на тествания компонент.
  -- Константата N = 8 определя разрядността на суматора.
   UUT: ADDERN generic map(N)
      port map(A => A, B => B, CIN => CIN, S => S, COUT => COUT);
  -- Процесът TEST подава входни въздействия и проверява резултатите.
   TEST: process
   begin
      for I in TEST_VECTORS'range loop
         -- Подаване на един тестов вектор.
         A <= TEST_VECTORS(I).A;
         B <= TEST_VECTORS(I).B;
         CIN <= TEST_VECTORS(I).CIN;
         -- Изчаква установяване на изходите.
         wait for 100 ns;
         -- Проверка на резултата.
         assert (S = TEST_VECTORS(I).S)
            report "Получената сума " & STDLV2STR(S) &
            " се различава от очаквания резултат:"&
            STDLV2STR(TEST_VECTORS(I).S);
         assert (COUT = TEST_VECTORS(I).COUT)
            report "Получения пренос " & STDL2CHAR(COUT) &
            "се различава от очаквания резултат:" &
            STDL2CHAR(TEST_VECTORS(I).COUT);
      end loop;
      wait;
   end process;
end;
```

② Компилирайте файловете ADDERN.VHD и ADDNTEST.VHD. Симулирайте библиотечната единица TESTBENCH и проверете работоспособността на модела. За симулацията използвайте командата run forever. Обяснете на какво се дължи полученото съобщение за грешка. Определете бързодействието на суматора.

③ Като използвате модела ADDERN, съставете описание на тестова постановка за 16-битов суматор. Напишете тестови вектори, които да активират критичния път в схемата. От резултатите от симулацията определете бързодействието на суматора.



Фиг. 2. Тестова установка за 8-битов суматор

Въпроси и задачи

① Да се довърши процедурата INT2STDLV, която да преобразува числа от тип integer в масиви от тип std_logic_vector:

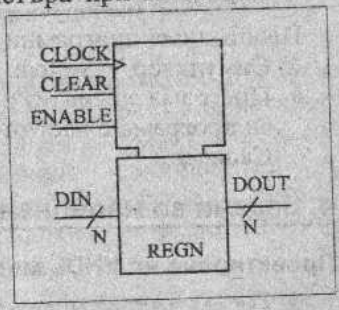
```
procedure INT2STDLV (INT : in integer; STDLV : out std_logic_vector) is
   begin ... end;
```

② Проектирайте тестова установка за 8-битов суматор, която да проверява автоматично всички възможни комбинации от входни сигнали.

Вместо масив с тестови вектори използвайте вложени цикли и прецедурата INT2STDLV, разработена в предишната задача.

③ Съставете параметричен модел на N-битов регистър (фиг. 3). Входните данни DIN(1-N) се записват в регистъра при ниско ниво на сигнала ENABLE и нарастващ фронт на CLOCK. Регистъра се нулира при ниско ниво на входа CLEAR. Като запомнящи елементи използвайте D-тригери управлявани по фронт.

④ Проектирайте тестова установка за проверка на модела, разработен в предишната задача. Използвайте масив с тестови вектори и автоматична проверка на разултатите.



Фиг. 3. N-битов регистър

ТЕМА № 4

АВТОМАТИЧЕН СИНТЕЗ НА СХЕМА ОТ VHDL ОПИСАНИЕ

1. Цел

Да се придобият опит и практически умения при използване на средства за автоматичен синтез на логически схеми от поведенческо описание на VHDL.

2. Задание

По зададена спецификация да се проектира поведенческо описание на АЛУ. Да се състави съответната логическата схема като се използват средства за автоматичен синтез.

Спецификация на АЛУ

Схемата извършва аритметични операции върху две 4-битови числа като разултатът също е 4-битов. Описанието на сигналите и функциите на АЛУ-то са дадени в следните таблици:

Таблица на сигналите

Сигнал	Тип	Разрядност	Предназначение
A	вход	4	операнд
B	вход	4	операнд
OPCODE	вход	2	код на операция
F	изход	4	резултат

Таблица на функциите

OPCODE	F
00	A
01	not A
10	A + B
11	A - B

3. Общи указания

Необходими програмни средства:

- ◊ Симулатор V-System. Работата с него е описана в Приложение 1;
- ◊ Програма за синтез и оптимизация на логически схеми Synergy. Тази програма е част от системата за проектиране на ИС на фирмата Cadence.

4. Задачи за изпълнение

Проектиране на VHDL модел на АЛУ

① Да се създаде поведенчески VHDL модел на разгледаното в заданието АЛУ. Сигналите трябва да бъдат от тип `std_logic_vector`.

За създаването на модела се използва пакета `std_logic_arith`, който дефинира операциите събиране (+) и изваждане (-) за сигнали от тип `std_logic_vector`. За тази цел в началото на VHDL файла се записват операторите:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

② Да се създаде тестова установка и чрез симулация с програмата V-System да се докаже работоспособността на модела.

Синтез на логическа схема на АЛУ

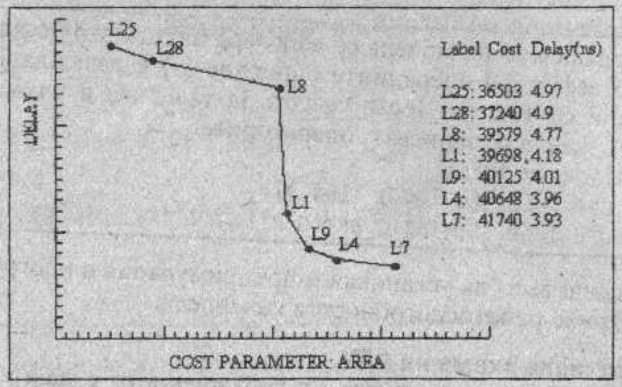
Предназначението на тази точка от упражнението е да се изследва пространството на възможните схемни решения на проектираното АЛУ и да се реализират някои от тях.

③ Да се въведе разработения модел на АЛУ в програмата Synergy:

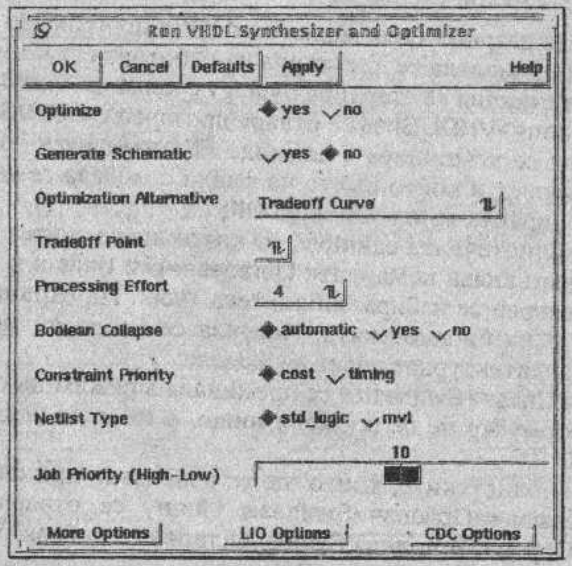
- Файлът с VHDL модела се прехвърля на работната станция;
- От работната станция се стартира Synergy с команда **vhdlSyn&**;
- С команда **Open=>VHDL Shell** се отваря прозорец с име **VSH**;
- VHDL модела се компилира с команда **Files=>Analyze**. На екрана се появява формуляр, в който името на файла с модела се записва в полето **File** и се щраква върху клавиша **OK**;
- Избира се библиотечната единица, от която ще се синтезира схемата. За целта се изпълнява командата **Libraries=>List Units** и в появилия се диалогов прозорец се избира библиотека **Work**. На екрана се появява съдържанието на библиотеката. Посочва се библиотечната единица съдържаща архитектурната част на модела;
- С командата **Units=>Synthesize** се преминава в режим на синтез. В появилия се формуляр не се променя нищо, а само се подтвърждава с клавиша **OK**;
- Указват се библиотеките, които ще се използват при синтеза. С командата **Session=>Options=>Synthesis Library** се отваря формуляра **Synthesizer Options**. В полето **Symbol Libraries** се добавя библиотеката **StdLib**, а в полето **Library Name - Syn_ind**;

④ Да се генерира графиката на възможните проектни решения (trade-off curve). Такава графика е показана на фиг. 1. Всяка от точките L1, L7 и т.н. означава възможна схемна реализация в пространството на решенията. В таблица са дадени цената (измерена в единици площ) и максималното закъснение за всяка от реализациите. Въз основа на тази информация проектантът решава кой от вариантите за реализация на схемата да избере.

- Изпълнява се командата **Run=>Synthesizer**. Появилия се на екрана формуляр се попълва както е показано на фиг. 2 и се натиска **OK**.



Фиг. 1. Графика на проектните решения.

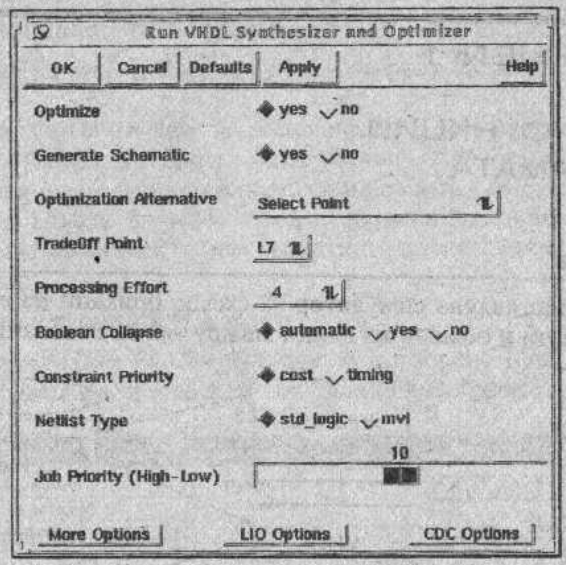


Фиг. 2. Генериране на графика на проектните решения.

- Получената графика се изчертава с командата **Show => Output => Tradeoff Curve**.

⑤ Да се синтезират схемите съответстващи на двете крайни точки от графиката на проектните решения. Да се обясни какви са основните разлики между двете схеми.

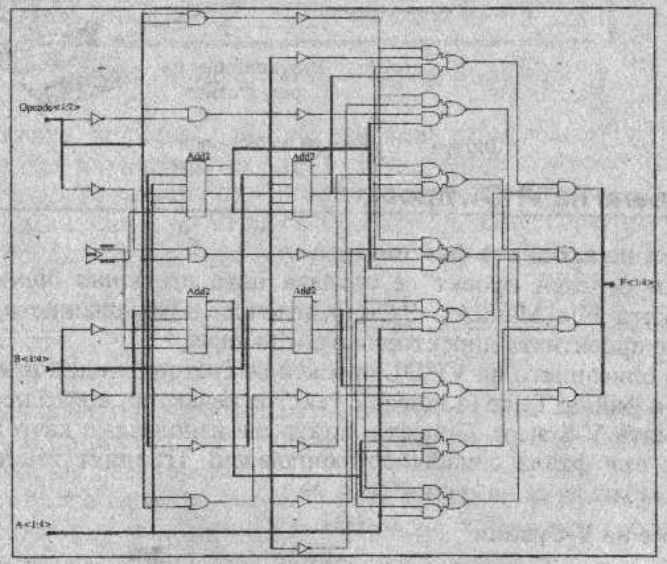
- Изпълнява се командата **Run=>Synthesizer**. Появилия се на екрана формуляр се попълва както е показани на фиг. 3.



Фиг. 3. Синтез на схема.

- Схемата се визуализира с командата **Show => Output => Schematic.**

Схемата на една от реализациите на АЛУ-то е дадена на фиг. 4. (На фигурата не са показани имената на шините и компонентите.)



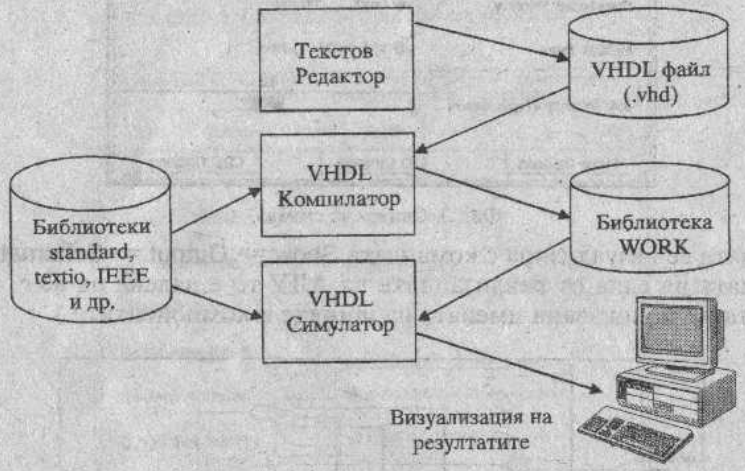
Фиг. 2. Автоматично синтезирана схема на АЛУ.

ПРИЛОЖЕНИЕ № 1

ОСНОВНИ ФУНКЦИИ НА ПРОГРАМАТА V-SYSTEM

1. Въведение

V-System представлява симулатор за схеми, описани на езика VHDL. Етапите на работа и обмена на данни между тях са показани на фиг. 1.



Фигура 1. Структура на V-System

2. Създаване на VHDL проект

Подготовка на входните файлове

- За всеки отделен проект се създава нова *проектна директория* с програмата File Manager. За определеност в изложението ще приемем, че проектната директория е c:\vhdl\mpis.
- Самото описанието на VHDL проекта се състои от един или няколко текстови файла. Те се създават с текстов редактор, който не е част от програмата V-System. Проекта, който ще използваме като пример, е записан във файла c:\vhdl\mpis\example.vhd. Пълният текст на този пример може да се намери в Тема № 3.

Стартиране на V-System

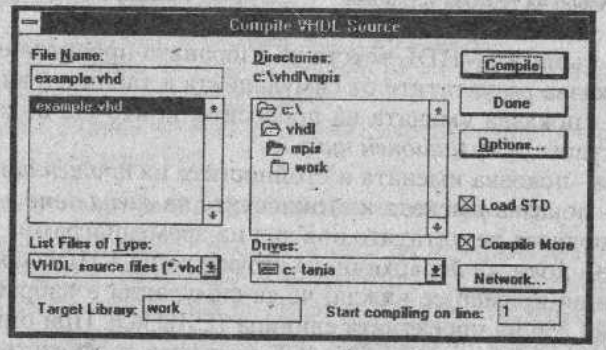
- С мишката се щраква два пъти върху иконата V-System.

Начално установяване на V-System

- С командата **File => Directory** се отива в по-рано създадената проектна директория.
- С командата **Project => New** се създава файл с описанието на проекта. Името на този файл трябва да бъде **vsystem.ini**, а неговото съдържание се попълва автоматично в процеса на работа с V-System.
- С командата **Library => New** се създава работна библиотека с име **work**. В нея ще се записват резултатите от компилацията на *проектните единици*.

Компилация на входния VHDL файл

- Избира се командата **File => Compile**. В прозореца (фиг. 2) се указва името на входния файл и се натиска клавиша **Compile**.



Фиг. 2. Прозорец за избор на файл за компилация

Съобщенията за грешки при компилация се извеждат в прозореца **Transcript**. След компилирането на всички входни файлове прозорецът на компилатора се затваря с натискане на клавиша **Done**. Ако при компилацията има грешки то VHDL файла трябва да се корегира с текстовия редактор и повторно да се компилира.

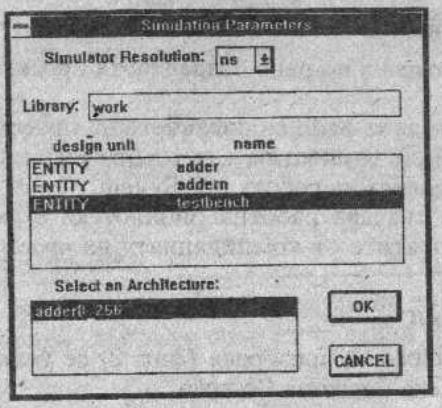
3. Симулация на VHDL проект

Инициализиране на симулатора

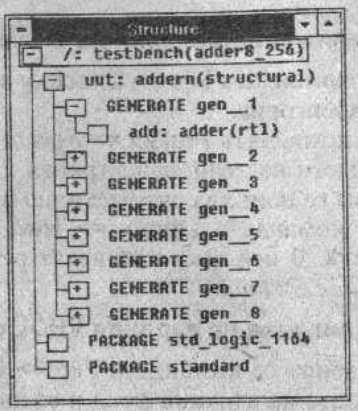
- За преминаване в режим на симулация се използва командата **File => Simulate**. В прозореца (фиг. 3) се избира *проектната единица* (entity), която ще бъде симулирана.

Тази проектна единица трябва да представлява *тестова установка*, т.е. да включва в себе си както VHDL модела, който ще бъде верифициран, така и *процеси* за подаване на входни въздействия.

След инициализирането на екрана се появяват следните прозорци:



Фиг. 3. Избор на тестова установка



Фиг. 4. Йерархия на VHDL проект

- ◊ **Source** - съдържа VHDL текста на избраната проектна единица;
- ◊ **List** - показва резултатите от симулацията в табличен вид;
- ◊ **Process** - показва имената на процесите, които ще бъдат изпълнени в текущия *симулационен цикъл*;
- ◊ **Variables** - показва имената и стойностите на *променливите*.;
- ◊ **Signals** - показва имената и стойностите на *сигналите* в проекта;
- ◊ **Wave** - показва резултатите във вид на времедиаграми;
- ◊ **Structure** - показва йерархичната структура на VHDL проекта.

От показания пример се вижда, че за симулация е избрана *архитектурата* ADDER8_256 на проектната единица TESTBENCH. При описанието на TESTBENCH е използван *компонента* UUT, на който съответства проектната единица ADDERN. Използвани са и *пакетите* STD_LOGIC_1164 и STANDARD. От своя страна ADDERN е съставен от осем компонента ADDER.

Стартиране на симулацията

Симулацията се стартира като в прозореца **Transcript** се напише командата **run** *<време>*. Като параметър се указва времето, до което да се извършва симулацията. Например **run 100 us** означава, че симулацията ще завърши след 100 микросекунди *симулационно време* или по-рано ако се изчерпи списъка на *заплануваните транзакции*. Симулацията може да бъде прекъсната по всяко време с натискането на екранния клавиш **Break**.

Когато е необходимо симулацията да се повтори от начало, се използва командата **File => Restart**. Тя установява симулационното време в нула и заличава от паметта всички предишни резултати. Подобно рестартиране се налага в следните случаи:

- ◊ След промяна и компилация на някой от входните VHDL файлове;
- ◊ След добавяне на нови сигнали в прозореца **Wave**.

4. Анализ на резултатите от симулацията

Резултатите от симулацията с програмата V-System са два типа:

- ◊ Промени на състоянията на сигналите, които се визуализират в прозорците **Wave** и **List**;
- ◊ Съобщения издавани от оператори **assert** включени в VHDL проекта - отпечатват се в прозореца **Transcript**.

За да се наблюдават определени сигнали те трябва да бъдат добавени в прозорците **Wave** и / или **List**. Съществуват няколко възможни подхода в зависимост от това дали за визуализация се избират отделни сигнали или всички сигнали в дадена област от проекта. В случая под "област от проекта" се разбира проектна единица (например TESTBENCH) или неин компонент (например компонента ADDER, генериран от оператора GEN_1).

Избор на единични сигнали

- От прозореца **Signals** с мишката се избира сигнал;
- С командата **Signals => Add to Waveform => Selected signal** избрания сигнал се добавя в прозореца **Wave**;
- С командата **Signals => Add to List => Selected signal** избрания сигнал се добавя в прозореца **List**.

Избор на всички сигнали в дадена област от проекта

- От прозореца **Structure** се избира област от йерархията на проекта;
- С командата **Signals => Add to Waveform => Signals in Region** всички сигнали в избраната област се добавят в прозореца **Wave**;
- С командата **Signals => Add to List => Signals in Region** всички сигнали в избраната област се добавят в прозореца **List**.

Възможно е наведнъж да се добавят всички сигнали от цялата йерархия на проекта. Това става с командите **Signals => Add to Waveform => Signals in Design** или **Signals => Add to List => Signals in Design**.

Представяне на резултатите във вид на времедиаграми - Wave

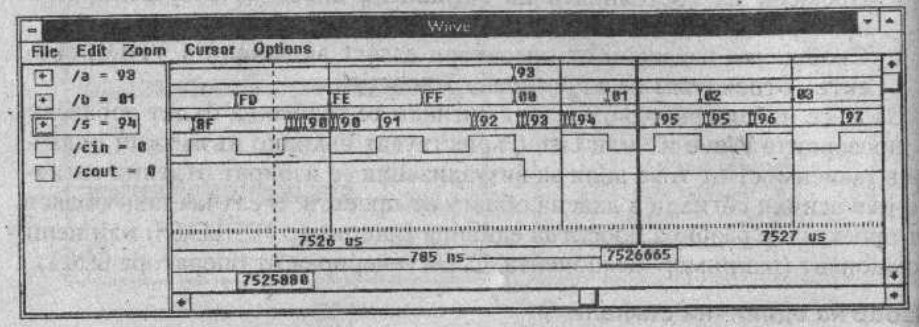
В прозореца **Wave** (фиг. 5) сигналите се представят по два различни начина в зависимост от типа им.

Скаларните сигнали се изчертават като последователност от ниски и високи лигически нива. Когато сигнала е в неопределено или високоимпедансно състояние неговата времедиаграма има междинно ниво. Освен това е възможно различните нива на сигналите да се оцветят в различни цветове с командата **Window => Color => Wave**.

Сигналните вектори се визуализират чрез техните числови стойности. С командата **Options => Signal Options => Radix** може да се избира използваната бройна система: двоична, осмична, десетична или шестнайсетична. При необходимост да се наблюдават отделните сигнали във вектора, с мишката се щраква еднократно върху знака '+' вляво от име-

то на вектора.

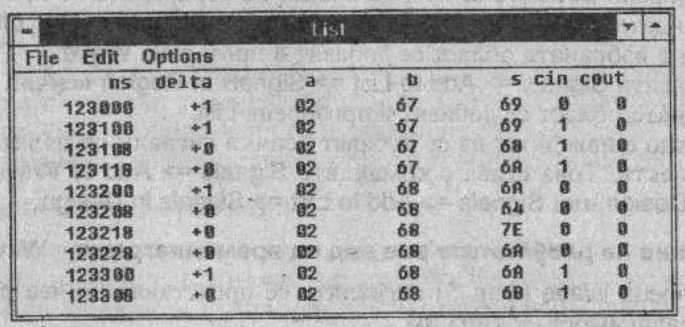
Прецизното отчитане на времеинтервали се извършва с помощта на измервателни курсори. Те се добавят с командата **Cursor => Add**.



Фиг. 5. Представяне на резултатите от симулацията в графичен вид

Представяне на резултатите във вид на таблица - List

Всеки ред в прозореца **List** (фиг. 6) съответства на промяна на стойността на някой от наблюдаваните сигнали. Предимството на този тип представяне е възможността с максимална точност да се определи кога е настъпило дадено събитие.



ns	delta	a	b	s	cin	cout
123000	+1	02	67	69	0	0
123100	+1	02	67	69	1	0
123108	+0	02	67	68	1	0
123110	+0	02	67	6A	1	0
123200	+1	02	68	6A	0	0
123208	+0	02	68	64	0	0
123218	+0	02	68	7E	0	0
123228	+0	02	68	6A	0	0
123300	+1	02	68	6A	1	0
123308	+0	02	68	6B	1	0

Фиг. 5. Представяне на резултатите от симулацията в табличен вид

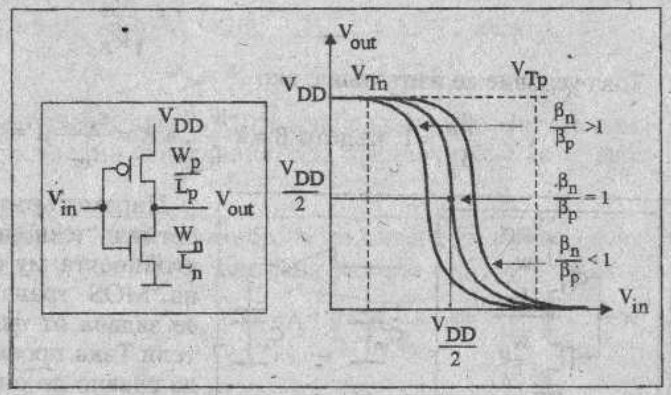
В първите две колони от таблицата е показано в кой момент от симулационното време и на кой цикъл на симулация е настъпила промяната. В останалите колони се отпечатват стойностите на избраните сигнали. За векторните сигнали може да се избира желаната бройна система чрез командата **Options => Signal Options => Radix**. Символите, които се използват за означаване на състоянията, зависят от типа на съответния сигнал.

СХЕМНО ПРОЕКТИРАНЕ НА ЛОГИЧЕСКИ БЛОКОВЕ

3

Ключови думи

- сложни елементи
- бързодействие
- оразмеряване
- оптимизация



Статичните схеми се използват за реализиране на произволни логически функции и служат за основа при проектирането на динамичните схеми. В CMOS изпълнение те се характеризират с ниска консумация, контролируемост на параметрите и висока шумоустойчивост.

При проектирането на CMOS логически елементи трябва да се решат три основни задачи:

- формиране на логиката;
- установяване на подходящи постояннотокови характеристики;
- анализ на бързодействието.

Тъй като структурата на статичните схеми се изгражда на базата на CMOS инвертор и параметрите им се оценяват като разширение на теорията на инвертора ще разгледаме накратко основните изисквания при проектирането му.

3.1. Проектиране на CMOS инвертор

3.1.1 Изисквания на статичния режим

Първата стъпка при проектирането на коректно функционираща схема е да се обезпечи нормален постояннотоков режим. Понятието *логическо прагово напрежение* се дефинира като напрежението, при което входното и изходното ниво се изравняват – $V_{th\,inv} = V_{in} = V_{out}$ (фиг. 3.1).

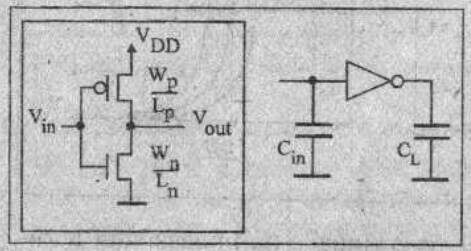
При това ако напрежението на входа е по-високо от стойността на логическото прагово напрежение, то напрежението в изхода ще бъде ви-

наги по-малко от него. За симетрично превключване инверторът трябва да се оразмери така, че логическото прагово напрежение да бъде около половината на захранва щото напрежение, т.е.

$$V_{th\,inv} = V_{in} = V_{out} = \frac{V_{DD} + V_{Tp} + V_{Tn}\sqrt{\frac{\beta_n}{\beta_p}}}{1+\sqrt{\frac{\beta_n}{\beta_p}}} = \frac{V_{DD}}{2}$$

Това условие се изпълнява, ако

$$\frac{\beta_n}{\beta_p} = 1, \text{ където } \beta = k\frac{W}{L}, \text{ а } k = \frac{\varepsilon\varepsilon_o}{t_{ox}}\mu = C_{ox}\mu$$



Фиг.3.1. CMOS инвертор

Параметърът k зависи от конкретния технологичен процес и стойността му в SPICE моделите на MOS транзистор обикновено се задава от фирмите производители.Така проектирането се свежда главно до определяне размерите W и L на транзисторите, където W и L са съответно ширина и дължина на канала на MOS транзистора. С t_{ox} е означена дебелината на тънкия окис, ε е диелектричната му проницаемост, ε_0 – диелектрична проницаемост на вакуума, μ – подвижността на токоносителите, а C_{ox} – специфичният гейтов капацитет. Условието

$$\frac{\beta_n}{\beta_p} = 1 \text{ означава } \beta_n = \beta_p \text{ или } \mu_p\frac{W_p}{L_p} \cong \mu_n\frac{W_n}{L_n}.$$

Като се има предвид, че подвижноста на електроните $\mu_n = (2 \div 3)\mu_p$, то за превключване на CMOS инвертора около средата на захранващото напрежение, отношението в размерите за P-каналните транзистори трябва да се избере от два до три пъти по-голямо от това за NMOS транзисторите

$$\frac{W_p}{L_p} \cong (2 \div 3)\frac{W_n}{L_n}.$$

Нормално за цифровите схеми дължините на каналите на транзисторите се избират равни на минимално възможната стойност за съответната технология. Тогава за превключване на инвертора в средата на захранването $V_{th\,inv} = \frac{V_{DD}}{2}$ трябва да бъде изпълнено

$$W_p = (2 \div 3)\,W_n$$

В повечето случаи целта при проектирането е да се постигне минимална площ, поради което се избира $W_p = W_n$. За транзистор с минимални размери $\beta_n / \beta_p \neq 1$ и логическото прагово напрежение се отклонява от $V_{DD}/2$, но в много малки граници. Например при

$V_{DD} = 5$ V за $W_p = 2W_n$ $V_{th\,inv} = 2.5$ V, докато
за $W_p = W_n$ $V_{th\,inv} = 2.24$ V,

при което изменението е по-малко от 10% и не е критично. Затова винаги, ако няма други специални изисквания, е за предпочитане да се избира $W_p = W_n$.

Да се оразмери PMOS транзистора на инвертор, който превключва при 2.5V за захранващо напрежение $V_{DD} = 5$ V , ако са известни $k_n = 55\ \mu A/V^2$, $k_p = 25\ \mu A/V^2$ и $\frac{W_n}{L_n} = 2$.

Изчислява се $\beta_n = k_n\frac{W_n}{L_n} = 110\ \mu A/V^2$

За прагово напрежение $V_{th\,inv} = \frac{V_{DD}}{2}$ се изисква $\beta_n = \beta_p$

$\beta_n = \beta_p = k\frac{W_p}{L_p} = 110\ \mu A/V^2$, откъдето $\frac{W_p}{L_p} = 4.4$.

Изборът на размерите на транзисторите оказва влияние не само върху стойността на напрежението, при което се извършва превключването, но и върху скоростта, с която се осъществява то.

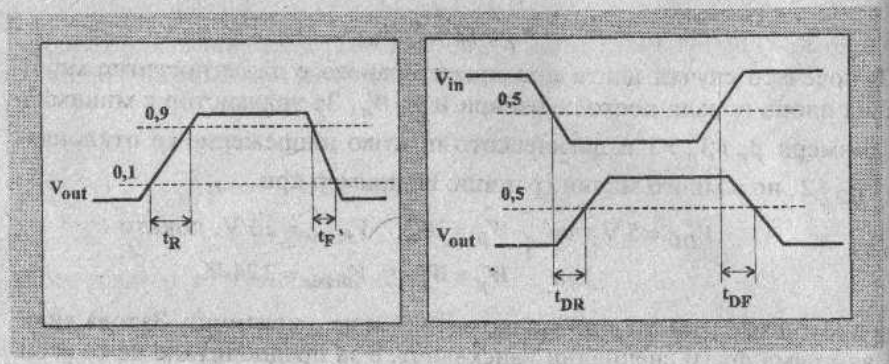
3.1.2. Характеристики при превключване

Бързодействието най-често е основният аспект при проектирането. То се характеризира ***със закъснението на стъпалото t_D и с времената за нарастване и спадане на сигнала t_R и t_F*** (фиг.3.2).

Оценката за тези времена при логическите елементи се дава с използване на *RC* модели за пътищата за зареждане и разреждане на товарния капацитет. Всеки MOS транзистор се замества с резистор със стойност

$$R = \frac{1}{\beta\,(V_{DD} - V_T)}, \text{ където } \beta = k\frac{W}{L}$$

се определя при проектирането със задаване на конкретни стойности на размерите на транзисторите, а V_T е праговото напрежение.



Фиг.3.2. Времена при превключване

Приблизителната стойност на ***времето на нарастване на изходния сигнал*** t_R е

$$t_R = c\frac{C_{out}}{\beta_p V_{DD}}, \text{ където } c = 3 \div 4 \qquad (3.1\text{ а})$$

е константа в указаните граници за стойности на захранващо напрежение V_{DD} между 3 и 5 волта и прагово напрежение $V_T = 0.5 \div 1$ V.

Аналогично ***времето за спадане*** t_F се получава

$$t_F = (3 \div 4)\frac{C_{out}}{\beta_n V_{DD}} \qquad (3.1\text{ б})$$

От тези изрази се вижда, че

- За да се постигне по-голямо бързодействие трябва да се минимизира товарният капацитет на стъпалото.
- Закъснението се изменя обратно пропорционално на β и намалява при увеличаване ширината на канала и намаляване дължината му.
- Времето за превключване се понижава с увеличаване стойността на захранващото напрежение.

Размерите на транзисторите, които ще осигуряват необходимите времена на превключване се изчисляват от

$$\left(\frac{W}{L}\right)_{n,inv} = \frac{(3 \div 4)\, C_{out}}{k_n t_F V_{DD}} \qquad \left(\frac{W}{L}\right)_{p,inv} = \frac{(3 \div 4)\, C_{out}}{k_p t_R V_{DD}} \qquad (3.2)$$

Отношението между времената за нарастване и спадане е:

$$\frac{t_R}{t_F} = \frac{\beta_n}{\beta_p}$$

За транзистор с минимални размери ($W_p = W_n$) времето за нарастване е приблизително два пъти по-голямо от това при спадане $t_R = 2t_F$, поради разликата в подвижностите на двата типа транзистори ($\mu_n = 2\mu_p$). При изискване за пълна симетрия при превключване е необходимо да се

избере $W_p = (2 \div 3)\, W_n$, за да се обезпечи $\beta_n = \beta_p$ и оттук $t_R = t_F$.

Времето за закъснение t_D се дефинира като времето между 50% изменение на установените стойности на входния и изходен сигнал и показва колко време е необходимо промяната на входа на логическия елемент да се предаде в изхода му.

То зависи от скоростта на изменение на сигналите и приблизително може да се определи като $t_{DR} = \frac{t_R}{2}$ и $t_{DF} = \frac{t_F}{2}$, където t_{DR} е закъснението при нарастване на сигнала, а t_{DF} – закъснението при спадане.

Закъснението при превключване се определя съответно от времеконстантата на зареждане τ_p и разреждане τ_n на изходния капацитет, както следва

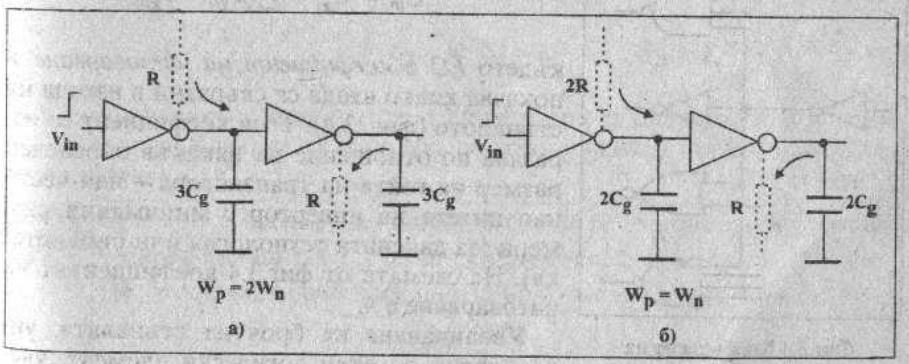
$$\tau_p = R_p C_{out} = \frac{1}{\beta_p (V_{DD} - |V_{Tp}|)} C_{out} = t_{DR}$$

$$\tau_n = R_n C_{out} = \frac{1}{\beta_n (V_{DD} - V_{Tn})} C_{out} = t_{DF},$$

където R_p и R_n са еквивалентните съпротивления на канала на отпушените P и N транзисторите, през които изходният капацитет C_{out} се зарежда до V_{DD} или разрежда до маса. С V_{Tn} и V_{Tp} са означени праговите напрежения за N и P MOS транзисторите.

Управление на еднотипно стъпало

Най-често дадено стъпало се свързва с друг логически елемент, чийто входен капацитет играе роля на товарен за предходното стъпало. Така например, ако инверторът е натоварен с аналогичен инвертор, то *входният капацитет* на следващото стъпало $C_{in} = C_{ox}(L_n W_n + L_p W_p)$ се определя от гейтовия капацитет C_g на транзисторите на товарния инвертор.



Фиг.3.3. Определяне закъснението на двойка CMOS инвертори

Да се определи закъснението на последователно свързани, еднотипни инвертори (в изрази на времеконстантата τ), ако размерите им са както указаните на фиг. 3.3 *а* и *б*.

Когато инвертор с $W_p = 2W_n$ е натоварен с аналогичен инвертор (фиг.3.3 *а*), то $R_p = R_n$, а еквивалентният входен капацитет на следващото стъпало $C_{in} = 3C_g$. Тогава закъснението на двойката инвертори е

$$t_D = t_{DR} + t_{DF} = R3C_g + R3C_g = 6RC_g = 6\tau$$

Ако инверторите са с минимални размери (фиг.3.3 *б*), еквивалентният входен капацитет $C_{in} = 2C_g$, $R_p = 2R_n$ и закъснението на двойката инвертори е

$$t_D = t_{DR} + t_{DF} = R2C_g + 2R2C_g = 6RC_g = 6\tau$$

Разгледаният *RC* модел е удобен за начални изчисления, но дава по-оптимистични резултати от практиката, тъй като не отчита нелинейността на елементите, наличието на паразитни капацитети и влиянието им от размерите и напреженията. В общ случай изходният капацитет е сума от входния капацитет на следващото стъпало C_{in}, паразитните капацитети C_{GD} и C_{DB} за N и P каналните транзистори, и капацитетите от междусъединенията.

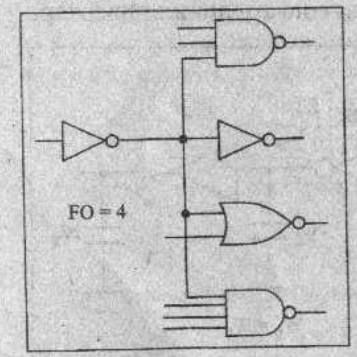
$$C_{out} = C_{in} + (C_{GD})_{n,p} + (C_{DB})_{n,p} + C_{wire}$$

Тези капацитети се изчисляват от програмата SPICE, ако се зададат размерите на областите за съответните параметри на модела.

Управление на няколко стъпала

Ако в изхода са включени повече стъпала, капацитетър е сума от входните им капацитети

$$C_{in} = C_{in} = \sum_{1}^{FO}(C_{gn} + C_{gp}),$$



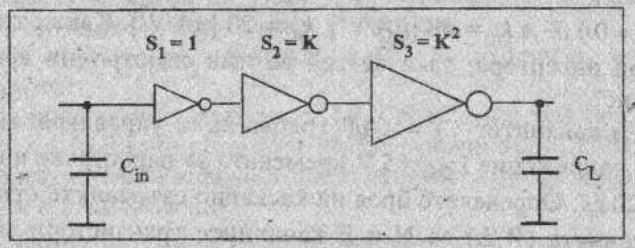
Фиг.3.4 Коефициент на натоварване

където *FO* е *коефициент на натоварване* и показва колко входа са свързани в изхода на стъпалото (фиг. 3.4). Този коефициент се изразява по отношение на някакъв определен размер на гейта на транзистора – най-често капацитета на инвертор с минимални размери (за дадената технология или библиотека). На схемата от фиг.3.4 коефициентът на натоварване е 4.

Увеличаване на броя на стъпалата, управлявани от един логически елемент, увеличава капацитивния товар и изисква промяна в размерите на управляващото стъпало, за да се постигне същото бързодействие.

Управление на голям капацитивен товар

Проблемите с управление на голям капацитивен товар възникват при необходимост от разпространяване на сигналите по дълги магистрали, при извеждането им извън чипа и др., понеже времената при превключване нарастват линейно с увеличаване на товарния капацитет. Този проблем се разрешава чрез използване на няколко последователно свързани инвертора (или други логически елементи) всеки от които е с по-големи размери от предшестващия с коефициент *S* (фиг.3.5), така че последният инвертор да може е да управлява големия капацитивен товар C_L за времето, което се изисква. Целта е да се минимизира общото закъснение на цялата група инвертори като същевременно се намали площта и разсейваната мощност.



Фиг. 3.5 Каскадно свързани инвертори за управление на голям капацитивен товар

Броят на необходимите стъпала *N* се изчислява от:

$$N = \ln\frac{C_L}{C_{in}}$$

след което се определя мащабният коефициент *K*, от който се формират коефициентите за всяко едно от стъпалата:

$$K = \left(\frac{C_L}{C_{in}}\right)^{\frac{1}{N}} \quad S_1 = 1 \quad S_2 = K \quad S_3 = K^2 \ldots\ldots S_N = K^{N-1}$$

Оптимално закъснение на групата инвертори се получава при $K = e = 2.73$, но в зависимост от използвания технологичен процес обикновено се използва $K = 2 \div 5$.

Стъпало с входен капацитет $C_{in} = 150$ fF трябва да управлява товарен капацитет $C_L = 2$ pF. Определете N и K.

Необходимият брой каскадно свързани стъпала се дава с:

$$N = \ln\left(\frac{2000}{150}\right) \approx 2.6$$

Избира се N=3. Изчислява се мащабният коефициент K

$$K = \left(\frac{2000}{150}\right)^{\frac{1}{3}} \cong 2{,}37\text{, откъдето } S_1 = 1 \quad S_2 = 2.37 \quad S_3 = 5.62$$

Третото стъпало се проектира, така че да увдовлетворява изискванията за конкретни времена на превключване при управление на C_L, след което размерите на останалите стъпала се получават чрез скалиране със съответните мащабни коефициенти.

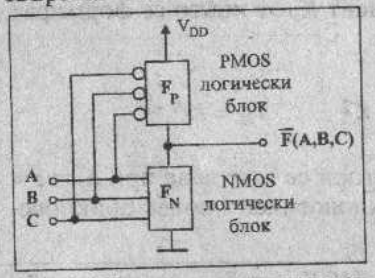
Контролни въпроси и задачи

① При захранващо напрежение $V_{DD} = 5$ V да се изчислят t_R и t_F за инвертор с размери $W_p = W_n = 6$ μm, $L_p = L_n = 1$ μm, ако товарният капацитет е $C_{out} = 0.6$ pF, а $k_n = 50$ [μA/V^2], $k_p = 20$ [μA/V^2]. Какви трябва да са размерите на инвертора, така че той да има симетрични времена при превключване?

② Външен капацитет $C_L = 50$ pF трябва да се управлява, така че при захранващо напрежение $V_{DD} = 5$ V времената за нарастване и спадане да са $t_R = t_F = 20$ ns. Определете броя на каскадно свързаните стъпала, както и отношенията (W/L) за N и P каналните транзистори за всяко от стъпалата, ако са известни $C_{in} = 150$ fF, $k_n = 55$ [μA/V^2], $k_p = 25$ [μA/V^2].

3.2. Проектиране на сложни CMOS логически елементи

Сложни логически функции могат да се реализират чрез подходящо свързване на N и P MOS транзистори в съответните вериги на инвертора. Блок-схема на произволен логически елемент е представена на фиг. 3.6, като с F_N и F_P са отбелязани съответно N- и P-логическите блокове.



Фиг.3.6 Блок-схема на произволен сложен логически елемент

Логическите функции се осъществяват чрез последователно или паралелно свързване на транзистори в логическите блокове, докато бързодействието и електрическите характеристики се определят от отношението в размерите W/L на транзисторите.

3.2.1. Правила за синтезиране на сложни логически елементи

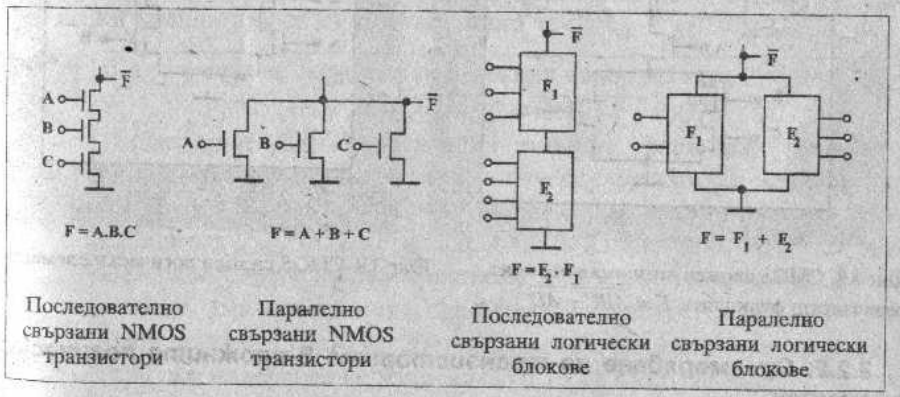
При проектиране на сложни CMOS логически елементи се използват следните основни правила:

- Всеки вход на логическия елемент се свързва едновременно към NMOS и PMOS транзистор;
- NMOS и PMOS веригите са комплементарни;

Спазва се последователността:

- Първоначално се структурира NMOS блока, като се съблюдават правилата:
  1. Последователно свързани NMOS транзистори изпълняват логическата функция И на входните си сигнали;
  2. Паралелно свързани NMOS транзистори реализират функция ИЛИ;
  3. Паралелни NMOS рамена реализират операцията ИЛИ на индивидуалните логически функции във всяко от рамената;
  4. Последователно свързани логически блокове реализират операцита И между логическите функции на всеки от блоковете;
  5. Изходът на схемата е инверсна функция на логическата функция на така формираният NMOS блок.
- Накрая се формира веригата на PMOS транзисторите, съгласно правилото:
  6. Последователно свързаните NMOS транзистори се трансформират в паралелно свързани PMOS транзистори, докато на паралелните NMOS транзистори съответстват последователно свързани в PMOS блока.

Правилата от 1 до 5 са онагледени на фиг.3.7.



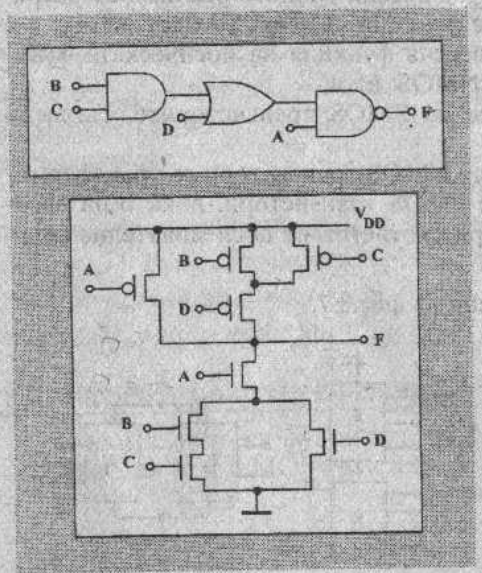
Фиг.3.7. Правила за формиране на NMOS логически блок

Да се състави сложен логически елемент, който изпълнява функцията: $F = \overline{ABC + AD}$.

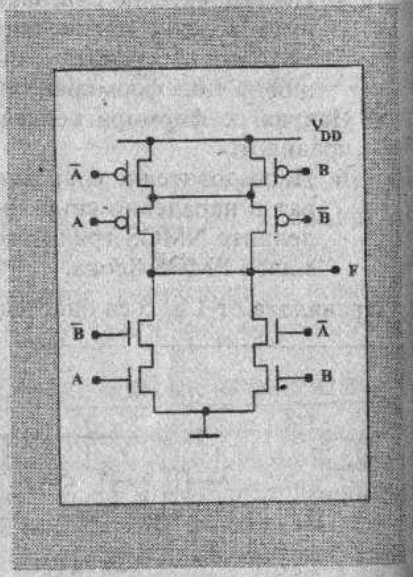
Уравнението може да се упрости, като се препише във вида:

$$F = \overline{A(BC + D)}.$$

Схемата, реализираща тази логическа функция е показана на фиг. 3.8. Изразът BC се получава чрез два последователно свързани NMOS транзистори, на входовете на които се подават съответно B и C сигналите. Съответните PMOS транзистори са в паралел. Функцията ИЛИ между BC и D изисква рамото с последователно свързаните BC транзистори да бъде паралелно включено към единичен MOS транзистор, на чийто вход е подаден сигналът D. В PMOS секцията сигналът D трябва да е последователен на паралелните B и C. Накрая операцията И между A и $(BC+D)$ се постига чрез последователно свързване на A с NMOS блока, изпълняващ функцията $(BC+D)$, и подходящото му паралелно включване в PMOS частта на схемата.



Фиг.3.8. CMOS сложен логически елемент, реализиращ функцията $F = \overline{ABC + AD}$



Фиг.3.9. CMOS сложен логически елемент

3.2.2. Оразмеряване на транзисторите в сложните логически елементи

Електрическите характеристики на сложните логически елементи зависят от структурата на схемата, от размерите на транзисторите и от реализацията на междусъединенията между тях.

Постояннотоковите характеристики, наклонът на предавателната функция и стойността на логическото прагово напрежение силно зависят от размерите на транзисторите и могат да бъдат изследвани с помощта на компютърни симулации.

Трябва да се има в предвид, че една и съща логическа функция може да се реализира по различен начин, но бързодействието е чувствително към броя на участващите транзистори в сигналния път, редът по който превключват и т.н. Общото съпротивление във веригата се получава с отчитане на паралелно и последователно включените резистори. Най-критичен за времето за нарастване е пътят с най-голямо съпротивление между изходния възел и захранване, докато за времето за спадане – този с максимално съпротивление към земя. Оценката за капацитетите се базира на размера на транзисторите и параметрите на процеса.

Алгоритъм за оразмеряване на сложни логически елементи

При проектиране на сложни логически елементи с цел постигане на определено бързодействие се използва следната последователност:

1. Оценява се изходният капацитет на схемата (C_{out}). Оразмерява се инвертор, който натоварен с този капацитет, ще осигури желаните времена при превключване. В резултат се определят размерите на NMOS и PMOS транзисторите в инвертора – $(W/L)_{n,inv}$ и $(W/L)_{p,inv}$.
2. Конструира се NMOS логическият блок и се преброява най-големият възможен брой m от последователно свързани транзистори. Размерите на всеки един от тези транзистори се избира, така че $(W/L)_n = m(W/L)_{n,\,inv}$.
3. Конструира се PMOS блокът и също се отчита най-големият брой q от последователно свързани транзистори в него. Размерите на всеки един от тях се изчисляват, така че $(W/L)_p = q(W/L)_{p,\,inv}$.

Ще разгледаме схемната реализация на функцията

$$F = \overline{AB + AC + BC} = \overline{AB + (A+B)C}$$

Сложният CMOS логически елемент, формиран съгласно правилата да изпълнява тази логическа функция, е показан на фиг. 3.10 *а*. Схемата трябва да се оразмери, така че при изходен капацитет C_{out} = 0,2 pF и захранващо напрежение V_{DD} = 5V да има равни времена за нарастване и спадане на сигнала $t_R = t_F$ =1.5 ns. Електрическите параметри са както следва: k_n = 55 [μA/V^2], k_p = 25 [μA/V^2]. Отношението в размерите на транзисторите в инвертора се изчислява от (3.2), както следва:

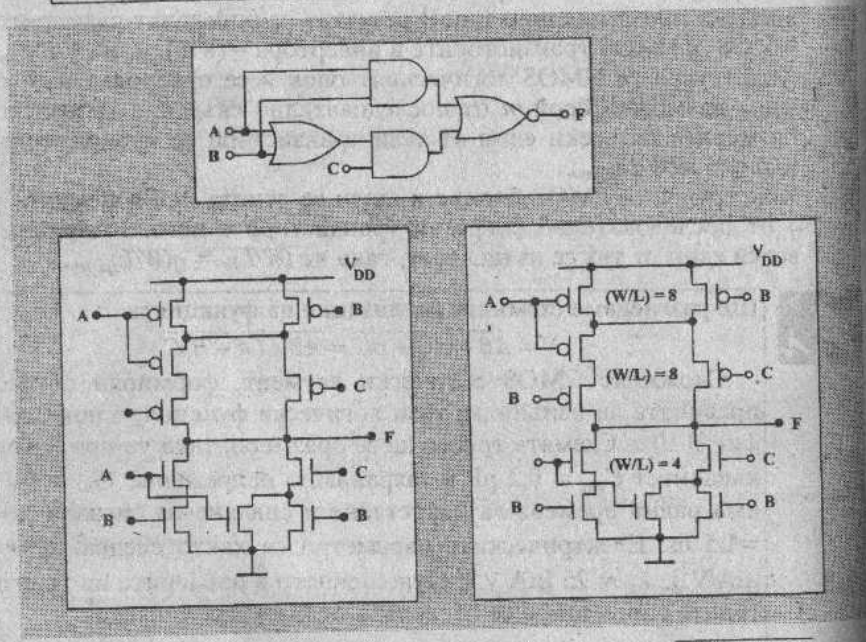
$$\left(\frac{W}{L}\right)_{n,inv} = \frac{(3 \div 4)\, C_{out}}{k_n\, t_F\, V_{DD}} = \frac{3{,}5.\ 0.2.10^{-12}}{(55.10^{-6})(1.5.10^{-9}).5} \cong 1.7$$

$$\left(\frac{W}{L}\right)_{p,inv} = \frac{(3 \div 4)\, C_{out}}{k_p t_R\, V_{DD}} = \frac{3{,}5.\ 0.2.10^{-12}}{(25.10^{-6})(1.5.10^{-9}).5} \cong 3.7$$

И за двата логически блока в токовия път в най-лошия случай участват по два последователно свързани MOS транзистори, т.е. $m=2$ и $q=2$. Тогава отношенията в размерите на транзисторите за двата блока са

$$\left(\frac{W}{L}\right)_n \cong 3.4 \quad \text{и} \quad \left(\frac{W}{L}\right)_p \cong 7.4 .$$

Те могат да се закръглят съответно на 4 и 8, за да се компенсира наличието на паразитни капацитети. Като се има в предвид, че при проектирането на MOS схеми обикновено дължината на канала на транзисторите се приема равна на минималната възможна за дадената технология, то от горните отношения могат да се изчислят ширините на MOS транзисторите за N и P блока на сложния логически елемент. Схемата на логическия блок, оразмерен за закъснения от 1,5ns при изходен капацитет $C_{out} = 0{,}2\,pF$ е показана на фиг. 3.10 *б*.



Фиг. 3.10 CMOS логически елемент, реализиращ функцията $F = \overline{AB + AC + BC}$

Този подход е полезен за първоначално оразмеряване и дава приблизителна оценка за закъсненията, тъй като при изчисленията не се отчитат паразитните капацитети на транзисторите и междусъединенията. Той илюстрира и проблемите с бързодействието на сложните логически елементи. В CMOS схемите не може да се избегне наличието на последователно свързани транзистори, тъй като NMOS и PMOS логическите блокове са комплементарни. При проектирането обаче трябва да се има в предвид, че последователно свързани NMOS транзистори са по-бързи от същия брой последователно включени PMOS транзистори.

Когато бързодействието е от първостепенно значение могат да се формулират следните ***основни препоръки*** за проектиране на сложни CMOS логически елементи:

- където е възможно се използват И-НЕ структури ;
- избягват се ИЛИ структури в бързи схеми, особено при повече от 4 входа или при голямо натоварване в изходния възел ;
- коефициентът на натоварване не трябва да превишава 5 ÷ 10 ;
- елементите, свързани във възли с голям коефициент на натоватване, се проектират с транзистори с минимални размери, за да се намали входният им капацитет, а оттам и общия товар във възела;
- стъпалата се оразмеряват така, че сигналите да имат стръмни фронтове.

Контролни въпроси и задачи

① По аналогия с фиг. 3.7 формулирайте правилата за изграждане на PMOS логическия блок, осигуряващ И и ИЛИ операции между входните сигнали в CMOS схемите.

② Съставете схемата за сложен логически елемент, изпълняващ функцията:

$$F = \overline{AB + CD}$$

③ Декомпозирайте схемата на мултиплексор 4:1, така че да съдържа три еднакви базисни блока. Каква логическа функция ще изпълнява всеки един от тях?

④ Съставете сложния логически елемент, реализиращ функцията изключващо ИЛИ между променливите A и B ($F = A \oplus B$).

⑤ Коя логическа функция изпълнява схемата от фиг.3.9?

⑥ Съставете схемата на двувходов И-НЕ с минимални размери на всички транзистори. Изразете времената на превключване в най-лош случай чрез тези на инвертор с минимални размери.

⑦ Изчислете времената за нарастване и спадане в най-лошия случай за 3-входов И-НЕ ако $\beta_n = 40$ [μA/V^2] при $W_n = 9$ μm, $L_n = 3$ μm и $\beta_p = 20$ [μA/V^2] при $W_p = 9$ μm, $L_p = 3$ μm) и $C_{out} = 0{,}2$ pF. Оразмерете схемата, така че двете времена да са равни.

ТЕМА № 5

ИЗСЛЕДВАНЕ И ОРАЗМЕРЯВАНЕ НА БАЗИСНИ ЛОГИЧЕСКИ ЕЛЕМЕНТИ

1. Цел на упражнението

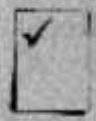
Да се анализира влиянието на натоварването върху динамичните параметри на базисни CMOS логически схеми и да се проектират елементи, способни да управляват конкретен капацитивен товар при определено закъснение.

2. Задание

Да се симулират схеми на CMOS инвертор, двувходов И-НЕ и двувходов ИЛИ-НЕ логически елементи за различни по стойност капацитивни товари като се анализира асиметрията на изходния сигнал при нарастване и спадане на сигнала. Да се оразмерят стъпалата за постигане на конкретно бързодействие при определен товар.

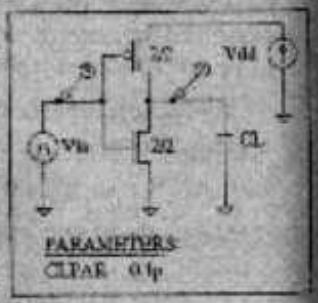
3. Общи указания

Да се изследват преходните процеси в схемите до 50 ns със стъпка 1 ns. За всички схеми захранващото напрежение е 5V и се подава от източник на напрежение от тип VSRC със стойност DC=5V, а входния сигнал – от източник тип VPULSE с параметри V1=5, V2=0, TR=TF=TD=5n, PW=25n. При изчертаване на CMOS схемите се използват символите CMOS-N и CMOS-P съответно за N и P каналните транзистори.



Размерите на транзисторите са в микрони и са означени на фигурите с отношението между дължината и ширината на канала L/W.

4. Задачи за изпълнение



Фиг. 1. CMOS инвертор

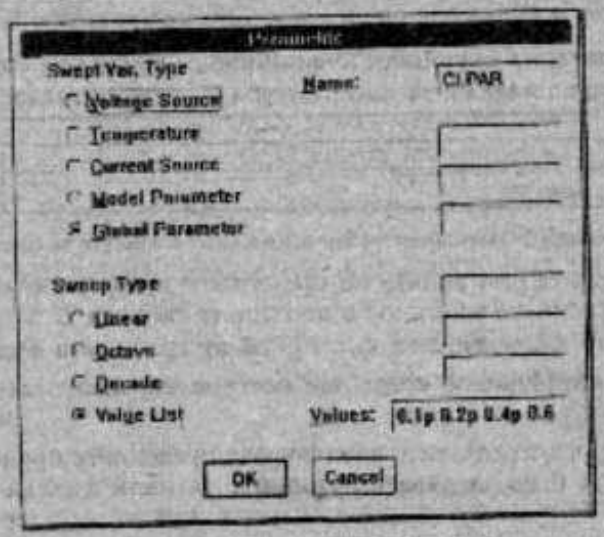
① Да се анализират преходните процеси в инвертор, показан на фиг. 1. Да се визуализират резултатите на входа и изхода. За изходния сигнал да се измерят стойностите на закъсненията при нарастване и спадане на сигнала.

Да се обясни на какво се дължи асиметрията на изходния сигнал при преминаване от 0→1 и от 1→0.

② Да се сравнят резултатите от анализа на преходните процеси в инвертор за стойности на товарния капацитет C_L = 0.1, 0.2, 0.4, 0.6 и 0.8 pF.

- В обща координатна система да се визуализират изходните напрежения за различни C_L.

За извършване на параметричния анализ в схемата се добавя символ PARAM (от библиотеката Special) и му се задават атрибути NAME1=CLPAR и VALUE1=0.01p. Така дефинираният *глобален* параметър CLPAR, се поставя на мястото на стойността на CL, т.е. VALUE=(CLPAR). Чрез командата Setup от меню Analisys се избира екранният клавиш Parametric и се попълва формата, в която се указва при какви стойности на CLPAR да се извършат симулациите (фиг. 2).



Фигура 2. Параметрична симулация

- Да се извърши анализ на преходните процеси и се отчете закъснението при нарастване и спадане на изходния сигнал за всяка стойност на C_L.
- Да се попълни таблица 1 и в обща координатна система се начертаят зависимостите $t_{DR} = f(C_L)$ и $t_{DF} = f(C_L)$ за инвертора.

C_L	pF	0.1	0.2	0.4	0.6	0.8
t_{DR}	ns					
t_{DF}	ns					

Таблица 1. Зависимост на закъснението от товарния капацитет

③ Оразмерете CMOS инвертор с минимални размери с оглед симетрия на времената за нарастване и спадане на изходния сигнал при зададен минимална дължина на канала $L = 2\mu m$. Моделирайте преходните процеси в схемата за капацитивен товар $C_L = 0.2$ pF и отчетете времето за закъснение на изходния сигнал.

④ Проектирайте CMOS инвертор, който ще обезпечи същото закъснение, както в точка 3, но при $C_L = 0.6$ pF.

Анализирайте преходните процеси в схемата, проектирана в т.3, но с товар $C_L = 0.6$ pF, като последователно изменяте размерите, както следва: $W_n' = f \cdot W_n$ и $W_p' = f \cdot W_p$ за $f = 1, 2, 3...$

С W_n и W_p са означени ширините на канала за N и P транзисторите, а f е мащабен коефициент, с който се увеличава ширината на канала, за да се осигури по-голяма товароспособност.

f	1	2	3	4	.....
t_{DR}, ns					

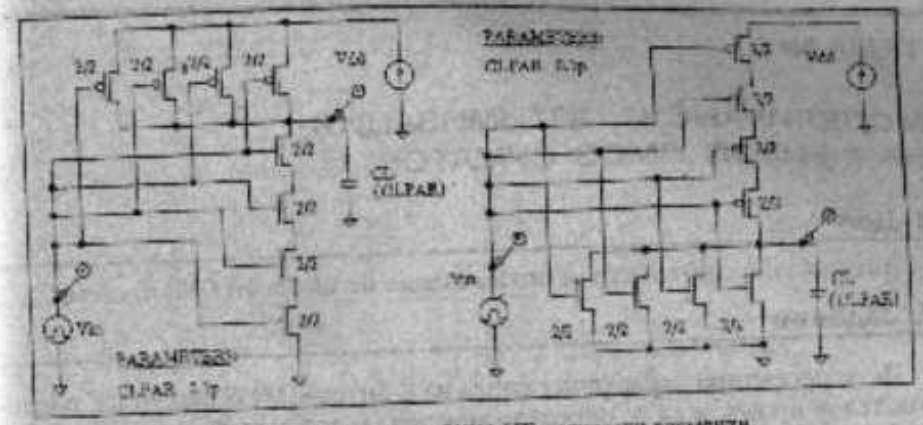
Таблица 2. Зависимост на закъснението от ширината на канала

- От получените при анализ на преходните процеси резултати определете закъснението за всяко f и попълнете таблица 2.
- Начертайте зависимостта $t_D = F(f)$ и от графиката определете този мащабен коефициент, който ще осигури търсеното закъснение при по-голямо C_L.
- Проверете резултата, като анализирате преходните процеси в схемите от т.3 и т.4 и наблюдавате изходните сигнали в обща координатна система.

⑤ Да се изследва зависимостта на наклона на предавателната характеристика на инвертора от размерите на транзисторите.

Извършва се *постояннотоков анализ* като входното напрежение се променя от 0 до 5V със стъпка 0.1V. Ширината на канала на PMOS транзистора се задава като глобален параметър WP със стойности WP=2, 4 и 8 μm. Резултатите се изчертават в обща координатна система.

⑥ Анализирайте преходните процеси в четиривходови И-НЕ и ИЛИ-НЕ логически елементи, показани на фиг. 3, за стойности на товарния капацитет $C_L = 0.1$, 0.2, 0.4 и 0.6 pF.



Фиг. 3 CMOS И-НЕ и ИЛИ-НЕ логически елементи

Да се изследва *динамичното* поведение на схемата, а не функционалността ѝ. За целта се използва само един източник на сигнал с атрибути V1=5, V2=0, TD=TR=TF=5n, PW=25n. Анализът на преходните процеси се извършва до 60 ns.

- В обща координатна система начертайте зависимостта на закъсненията при нарастване като функция на товарния капацитет $t_{DR} = F(C_L)$ за И-НЕ и ИЛИ-НЕ елементите. Аналогично и за закъсненията при спадане $t_{DF} = F(C_L)$.

⑦ Проектирайте двувходови И-НЕ и ИЛИ-НЕ елементи за симетричен изходен сигнал и закъснение като това на инвертора от точка 3 при товар $C_L = 0{,}2$ pF.

Въпроси и задачи

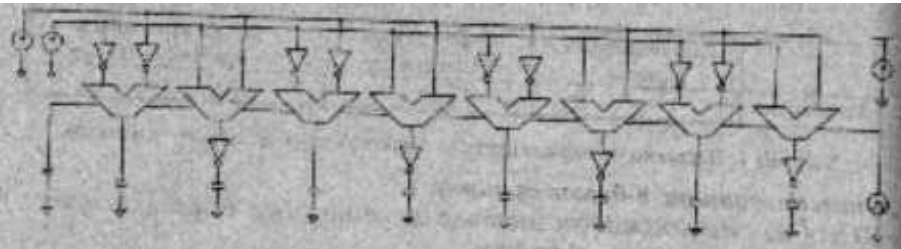
① Определете сумарното закъснение на двойки CMOS инвертори с минимална дължина на канала в следните случаи:

а) $W_p = \mu_n W_n$, б) $W_p = W_n$

② Защо времето за спадане на сигнала в изхода на И-НЕ логически елемент е по-голямо от времето за нарастване?

③ Коя е причината за асиметрията на изходния сигнал при многовходов ИЛИ-НЕ елемент с минимални размери на транзисторите?

④ Докажете, че в CMOS инвертор трябва да е изпълнено отношението $W_p / W_n = \mu_n / \mu_p$ за да се изравнят времената за нарастване и спадане на изходния сигнал.



Фигура 7. Оптимизирана архитектура на 8-битов суматор

⑥ Оразмеряване на транзисторите.

За да се повиши бързодействието на една MOS схема е необходимо да се увеличат ширините на гейтовете на транзисторите в критичния път. Всички други транзистори остават с минимални размери, за да се избегне нежелателно нарастване на площта и консумацията на схемата.

В схемата на еднобитовия суматор критичният път минава от cin към cout. На фиг. 6 са показани новите размери на транзисторите.

- Да се променят размерите на транзисторите в схемата на йерархичния символ на еднобитов суматор съгласно фиг. 6.
- Да се извърши анализ на преходните процеси и да се измери закъснението в критичния път. Данните да се запишат в таблица 1, в колоната "архитектурна оптимизация и оразмеряване".

Въпроси и задачи

① Да се докаже, че принципната схема на суматора фиг. 1 отговаря на таблицата на истинност от фиг. 2.

② Да се докаже еквивалентността на схемите на суматора от фиг.5 и фиг.7.

За целта е достатъчно да се сравнят логическите уравнения на изходите Sum1 на двете схеми и да се докаже, че са еквивалентни.

③ Защо след премахването на инверторите от критичния път на суматора не се наблюдава увеличаване на бързодействието?

④ Защо при оптимизацията чрез оразмеряване на транзисторите, размерите на NMOS и PMOS транзисторите не се променят еднакво?

⑤ Обяснете защо точно указаните на фиг.6 транзистори са с променени размери за подобряване на бързодействието.

ПРИЛОЖЕНИЕ № 2

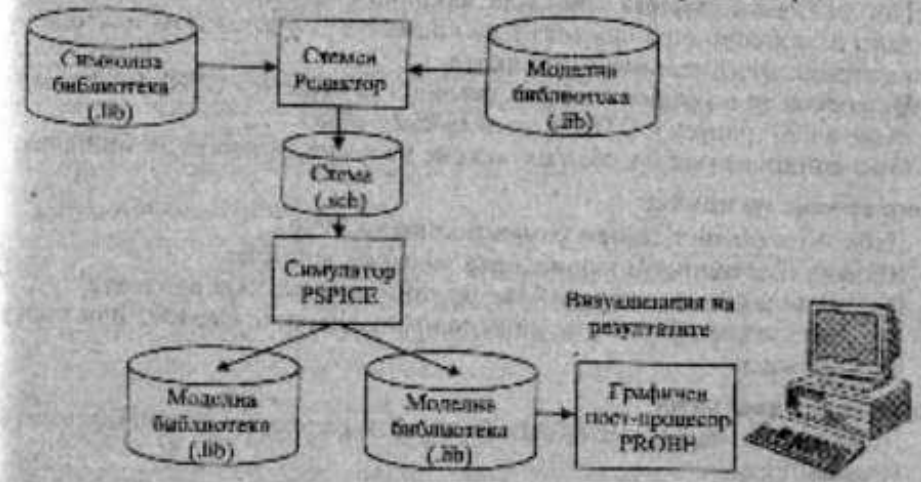
ОСНОВНИ ФУНКЦИИ НА DESIGN CENTER

1. Въведение

Системата за проектиране Design Center служи за изчертаване и моделиране на аналогови и смесени аналогово-цифрови електронни схеми. Нейните по-важни компоненти са следните:

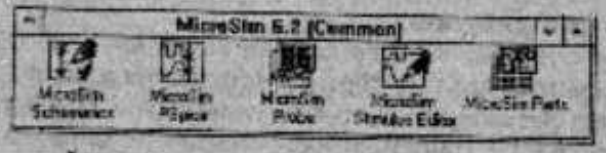
- ◊ Schematics – схемен редактор;
- ◊ PSpice – симулатор;
- ◊ Probe – графичен пост-процесор на резултатите от PSpice.

Връзките между отделните програми, последователността на тяхното използване и данните, които си обменят са илюстрирани на фиг. 1.



Фиг. 1. Структура на Design Center

Стартирането на всяка програма става по стандартния за Windows начин – като съответната й икона (фиг. 2) се посочи с курсора и двукратно се натисне левият клавиш на мишката.



Фиг. 2. Икони на програмите от Design Center

Схемата, която ще се изследва, се изчертава с графичния схемен ре-

дактор Schematics. За да може да се симулира, схемата трябва да съдържа следните обекти:

- Електронни елементи с атрибути, указващи техните параметри – например размери на транзисторите, стойности на кондензаторите и т.н.;
- Връзки между елементите;
- Връзки към земя и захранване;
- Източници на входни въздействия.

В настоящото приложение са разгледани само тези възможности на Design Center, които са необходими за моделиране на цифрови MOS интегрални схеми.

2. Основни операции за изчертаване на схема

Добавяне на елемент

- Избира се клавиш  или команда Draw => Get New Part .
- В прозореца се написва името на елемента и се натиска клавиша OK.
- Поставят се в схемата един или няколко елемента от избрания тип като се позиционира символът на елемента на желаното място и се натиска левият клавиш на мишката.
- В процеса на позициониране на символа се използва CTRL-R за *завъртане* на 90 градуса и CTRL-F за *огледален образ*.
- Операцията завършва след натискане на десния клавиш на мишката.

Прекарване на връзка

- Избира се клавиш  или команда Draw => Wire.
- Щраква се с мишката в началната точка на връзката.
- Всяко следващо щракане добавя по един сегмент към връзката.
- Връзката завършва като се щракне върху извод на елемент или върху друга връзка.

Свързване към земя

- Добавя се символ с име AGND и към него се прекарват необходимите връзки.

Свързване към захранващи източници

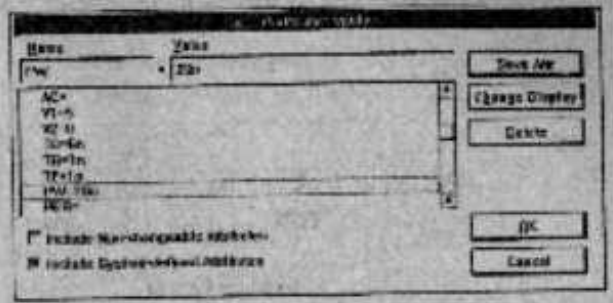
- За всеки захранващ източник се добавя по един символ VSRC.
- Напрежението на източника се задава посредством стойността на атрибута DC (например DC=5V).

Редактиране на атрибутите

Атрибутите описват свойствата (или качествата) на обекти от схемата като елементи, връзки, портове и др. Всеки атрибут се състои от име и стойност. Задаването на стойности на някои атрибути е задължително, а на други – не. Например, задължително е да се укаже стойността на използвания резистор или кондензатор (чрез атрибута VALUE), но не е задължително да се наименоват връзките (чрез атрибута LABEL).

Атрибутите на обект от схемата се редактират по следния начин:

- Обектът се избира с *двукратно* щракване с мишката върху него.
- Появява се прозорец със списъка на атрибутите на избрания обект.



- След промяна на стойността на даден атрибут, се натиска клавиша Save Attr.

Преместване на обекти

- С мишката се избира единичен обект или група обекти (заградени в прозорец).
- Избраните обекти се преместват като курсорът се постави върху един от тях, натиска се левият клавиш на мишката и тя се движи без да се отпуска клавиша.
- След отпускане клавиша обектите се разполагат на новото им място.

Копиране на обекти

- С мишката се избира обект или група обекти.
- Избраните обекти се копират в клипборда с команда Edit => Copy.
- С команда Edit => Paste избраните обекти се вземат от клипборда и с местене на мишката и щракане на левия й клавиш се разполагат на желаното място в схемата.

Запис на схемата във файл

- Избира се клавиш  или команда File => Save.
- Ако схемата се запомня за първи път, то в появилия се прозорец трябва да се укаже пътят до желаната директория и името на файла.

Зареждане на схема от файл в схемния редактор

- Избира се клавиш  или команда File => Open.
- В прозореца се задава пътят до желаната директория и името на файла, от който да се прочете схемата.

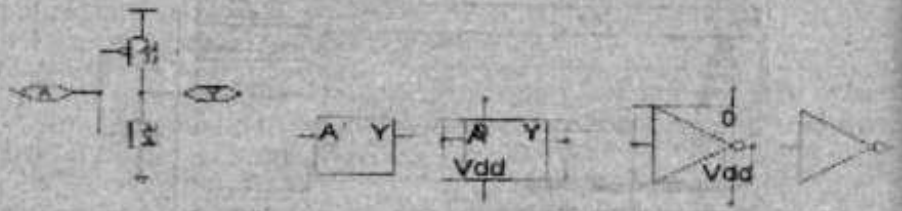
3. Създаване на йерархични схеми

Идеята на йерархичното описание е да се заменят части от оригиналната схема със символи, за да се облекчи изчертаването и интерпретирането на схемата.

Създаване на йерархичен символ

Като пример ще разгледаме създаването на йерархичен символ на CMOS инвертор (фиг. 2-а).

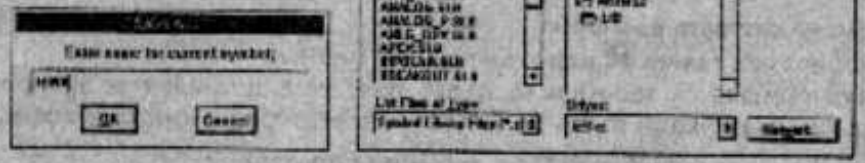
- Първо се изчертава схемата, която ще бъде заменена с йерархичния символ.



а) схема б) редактиране

Фигура 2. Създаване на йерархичен символ

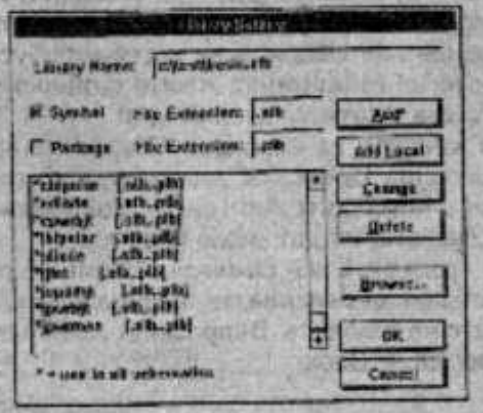
- На входовете и изходите на схемата се поставят *портове* от тип **Interface** и им се присвояват етикети съответстващи на имената на входно-изходните сигнали.
- Свързването към сигнали като земя и захранване, които са общи за всички нива на йерархия, се извършва посредством *глобални портове*. В разглежданя пример, това са портовете **Vdd** и **Agnd**.
- От схемата автоматично се генерира символ с командата **File => Symbolize**. Първо се дава име на новия символ (в примера то е **HINV**), а след това се избира библиотеката, в която ще бъде записан. Ако библиотека с указаното име не съществува, то тя ще бъде създадена:



Използване на йерархичен символ

За да може новосъздаденият символ да бъде използван в чертеж е необходимо неговата библиотека да се добави в конфигурацията на схемния редактор:

- Изпълнява се командата **Options=>EditorConfiguration=>Library Settings**
- Указва се името на библиотеката и пътят до нея;
- Ако се натисне клавиша **Add Local** съдържанието на библиотеката ще бъде достъпно само в текущия чертеж. С клавиша **Add*** – библиотеката се прави достъпна във всички чертежи.



При работа мрежова инсталация на Windows обикновенно не е възможно да се използва командата **Add***. Тя се опитва да промени файла **MSIM.INI** в системната директория на Windows и тъй като тази директория е недостъпна за запис, се получава съобщение за грешка.

- Добавянето на йерархичен символ в чертежа става с **Draw => Get Part**.

Редактиране на символ

При първото добавяне на всеки автоматично генериран символ се преценява дали графичното му представяне се нуждае от коригиране. Обикновенно се налага ръчно редактиране тъй като Design Center създава само правоъгълни контури, а и разположението на изводите не винаги е удачно. Преминаването в режим на редактиране на символ става с команда **Edit => Symbol**. На фигура 2-б, от ляво на дясно са илюстрирани последователни етапи от редактирането на символа **HINV**.

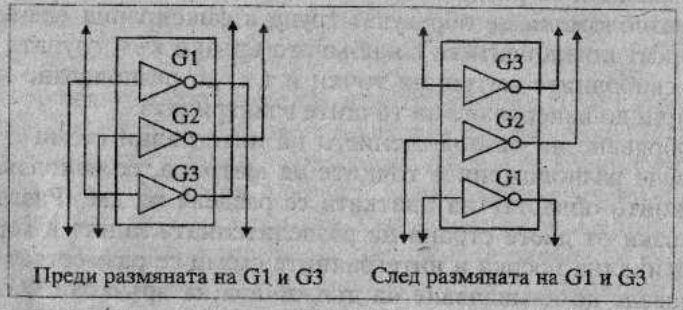
В началото символа е добавен в чертеж и е избран с мишката.

След това е изпълнена команда **Edit => Symbol** и схемния редактор е отворил прозорец, в който може да се редактира избраният символ. Типични операции на този етап са:

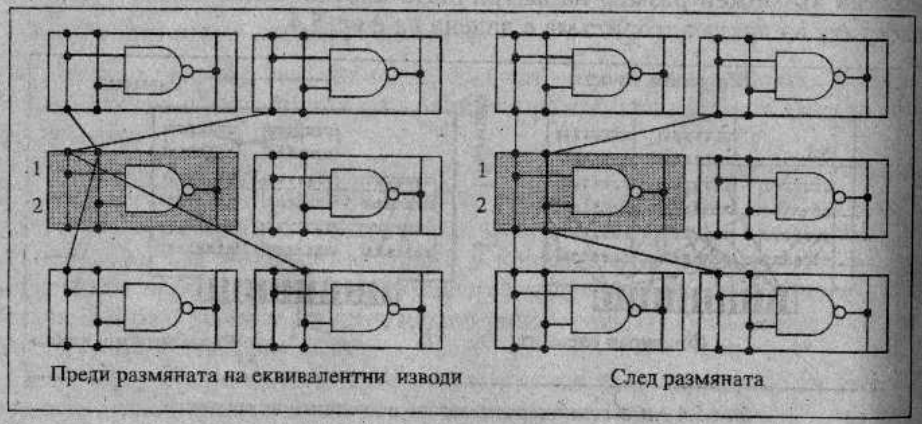
- Преместване на извод на елемента. Използват се процедурите за преместване и ротация на обекти, описани по-горе. Например всички входове да се разположат от лявата страна, а всички изходи - от дясната. Единия край на всеки извод е маркиран с "x" и той трябва да е ориентиран в посока "навън" от символа. В мястото на този маркер се осъществява контактът между извод и връзка.
- Промяна на типа на извод. С командата **Part => PinList** се отваря прозорец със списък на изводите и техните атрибути. Посочва се името на желания извод и в полето **Type** се избира типът му. Най-често се

да се намали общата им дължина. При тази операция се отстраняват кръгови връзки и даден извод се свързва към най-близката възможна точка от дървото на връзките. (Дървото на връзките обхваща всички електрически свързани елементи в дадена верига). Възможно е и такова пренареждане, при което конкретни връзки (най-често за земя и захранване) се насочват по протежение на надлъжната ос на интегралните схеми (biased reconnect), за да се облекчи впоследствие трасирането им.

В редица случаи, разполаганите обекти съдържат в общ корпус повече от един логически елемента. Например SN7404 има четири функционално еквивалентни ИНЕ елемента, а SN7474 – два D-тригера. Ако се разменят местата на еквивалентните елементи (gate swap) може да се постигне значително намаление на дължината на връзките. По този начин се облекчава трасирането, без да се нарушава функционирането на схемата (фиг.5.5).



Фиг. 5.5 Размяна на еквивалентни логически елементи



Фиг. 5.6 Размяна на еквивалентни логически изводи

Допълнително подобрение може да се постигне, ако се разменят и местата на еквивалентни логически изводи (pin swap). Разменянето на

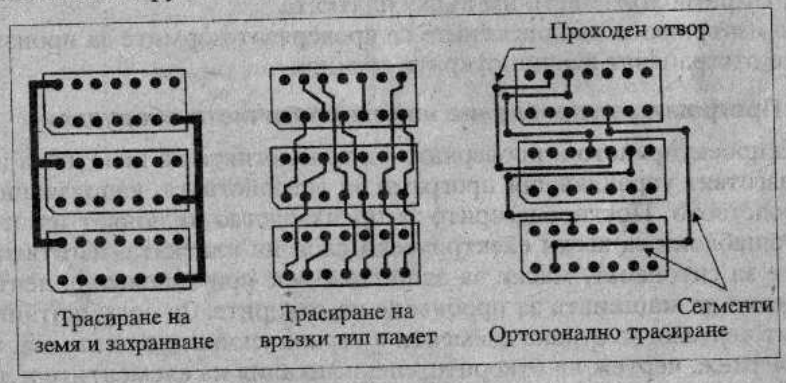
всеки един от входовете на двувходов ИНЕ логически елемент няма да промени функционирането му, но може значително да облекчи опроводяването на връзките, както е показано на фиг. 5.6.

5.4. Трасиране на връзките

След окончателното разполагане на компонентите по площта на платката се пристъпва към опроводяване на връзките между тях. При това се използват различни подходи и алгоритми за трасиране на отделните типове връзки.

Най-голямо внимание се отделя на опроводяването на сигналите за земя и захранване. Тези връзки обикновено са с по-голяма ширина на пътечката, разполагат се предимно върху един слой и имат не повече от три сегмента между два извода на компонента (фиг.5.7*а*). Тяхното трасиране трябва да се завърши *окончателно* преди да се започне опроводяването на сигналните връзки.

Връзките *тип памет* свързват изводи на интегрални схеми, разположени в един или между два съседни реда или стълба (фиг.5.7*б*). Те се прекарват на един и същ слой и имат до пет сегмента на 45 градуса един спрямо друг. Този алгоритъм за трасиране позволява по-голяма гъстота на пътечките по платката, тъй като при опроводяването не се използва преминаване на друг слой, свързано с появата на проходни отвори.



Фиг.5.7 Типове връзки и подходи за опроводяването им

При *ортогоналното трасиране* се опроводяват всички останали връзки като хоризонталните сегменти се разполагат на един слой, а вертикалните – на друг с автоматично въвеждане на проходни отвори между тях (фиг. 5.7*в*).

За да се увеличи броят на автоматично опроведените връзки могат да се използват различни подходи – да се задава различен брой на използваните сегменти, да се извършва трасиране на отделен сигнал, на конкретно зададен участък от платката, опроводяване на част от връзка и др. Съществуват и съвременни алгоритми, които премахват вече прека-

рани писти, за да се освободи място за трасирането на друг сигнал, след което търсят нов път за отстранената писта, както и мощни алгоритми за намиране на път само в един слой между вече прекарани писти.

5.5. Проверка на проекта

Целта на тази проверка е да се гарантира работоспособно изделие след производството му. На този етап се проверява съответствието на връзките в платката с тези на електрическата схема и се сравнява разработената топология с изискванията на нормите на използвания технологичен процес.

5.6. Подобряване на технологичността на изделието

За да се подобри рандемана при производството, както и да се снижат електромагнитните смущения, особено при работа при високи честоти, се извършват довършителни работи за подобряване технологичността на изделието. Към тях спадат скосяване под 45 градуса или закръгляване на ъглите на печатните пътечки, добавяне на медна област или максимизиране на тази област в определен участък на платката, използване на конусовидни преходи на пистите към контактните площадки и др. За по-голямо удобство при монтажа на елементите може да се извърши и автоматичното им преномериране във възходящ ред в зависимост от разположението им върху платката.

След извършване на корекциите се проверяват нормите за производство и се отстраняват всички открити грешки.

5.7. Програми за управление на технологичното оборудване

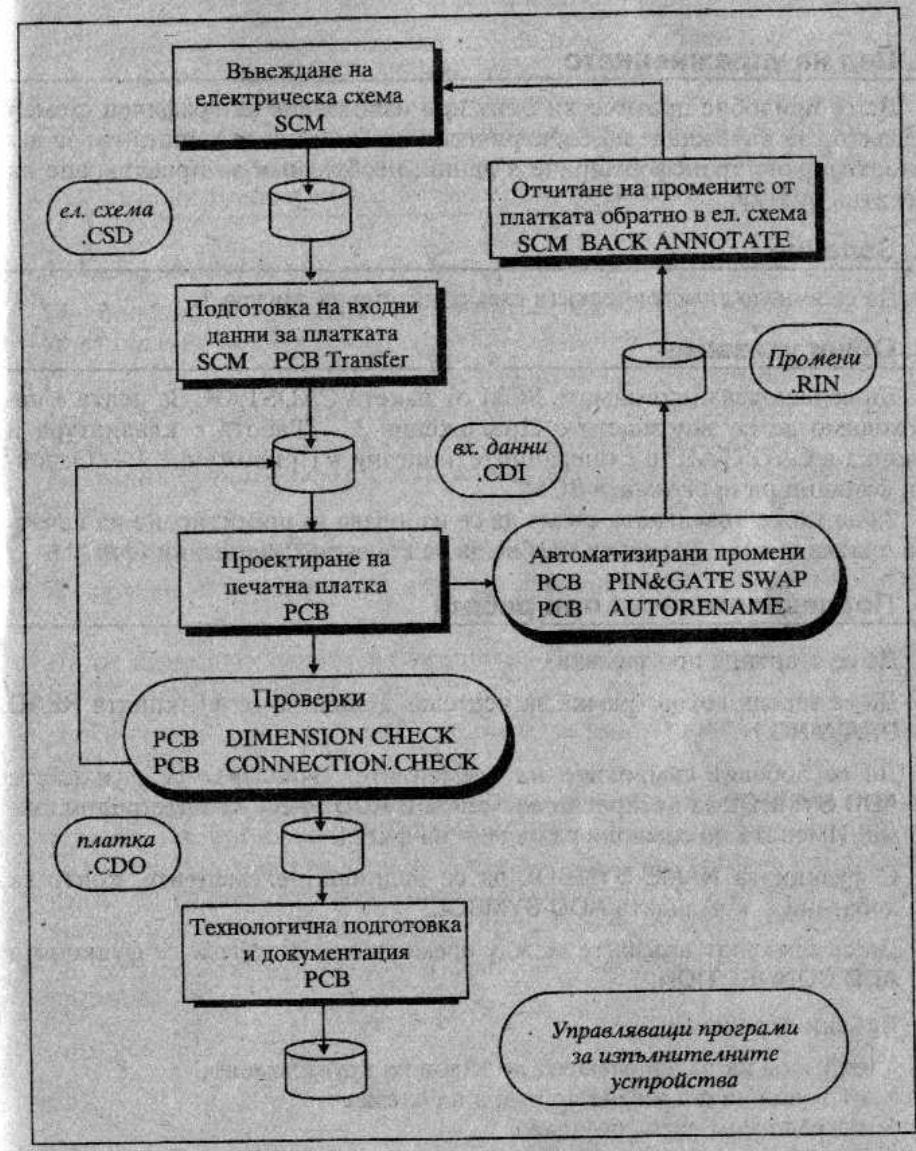
След проектирането и проверката на топологията на печатната платка се изготвят управляващи програми за устройствата, използвани при производството. Постпроцесорите за производство включват изготвяне на фотошаблони за всеки електрически слой на платката, изготвяне на маските за ситопечат, маски за защитния лак при запояване, лента за управление на машината за пробиване на отворите. За документация се изготвят чертежи за страна елементи и страна спойки на платката, монтажен чертеж, чертеж на отворите, спецификация на елементите и др.

5.8. Последователност при проектирането на печатни платки

Етапите, през които се преминава при проектирането на печатни платки, са илюстрирани на фиг. 5.8. Ако проектът съдържа специфични компоненти, за които в библиотеките няма въведени символи и корпуси с разположение на изводите, то те трябва да се създадат предварително преди започване на проектирането на платката.

Ако при проектирането на печатната платка се извършва автоматично разместване на еквивалентни логически елементи и изводи или автоматично преномериране на компонентите, тези промени трябва да се

отразят в електрическата схема. В противен случай няма да има съответствие между документацията за схемата и платката, което може да доведе до грешки и трудности при монтажа на елементите, оживяването на схемата и ремонта на изделието.



Фиг.5.8 Етапи при проектирането на печатни платки с програмата CADSTAR

ТЕМА № 9

ВЪВЕЖДАНЕ НА ЕЛЕКТРИЧЕСКА СХЕМА

1. Цел на упражнението

Да се придобие пратически опит при използване на графичен схемен редактор за въвеждане на електрически схеми и подготовката им за автоматизирано трансформиране в данни, необходими за проектиране на печатна платка.

2. Задание

Да се въведе електрическата схема, дадена на фигура 1.

3. Общи указания

Да се използва програмата SCM от пакета CADSTAR. За целта е необходимо да се запознаете с Приложение 3 – "Работа с клавиатура и мишка в CADSTAR" и с операциите, описани в Приложение 4 – "Основни функции на програмата SCM".

За да може въведената схема да се използва за проектиране на печатна платка всички елементи трябва да са коректно надписани (фиг. 1).

4. Последователност при работа

- Да се стартира програмата.
- Да се зареди готова рамка за чертежа. Използва се функцията **READ DRAWING**.
- Да се добавят символите на елементите. Използват се функциите **ADD SYMBOL** за дискретни елементи и **ADD PART** за интегрални схеми. Имената на символи са дадени на фиг. 1.
- С функцията **NAME SYMBOL** да се надпишат елементите, които са добавени с командата **ADD SYMBOL**.
- Да се прекарат връзките между елементите. Използва се функцията **ADD CONNECTION**.

Връзки се задават:

◊ от извод на даден елемент до извод на друг елемент;
◊ от точка на свързване до извод на елемент;
◊ посредством сигнално име.
◊ от извод на елемент до вече прекарана връзка. В този случай пакетът автоматично поставя точка на свързване (**JUNCTION POINT**).

Когато върху кръстосани връзки се постави допълнително точка на свързване с функцията **ADD JUNCTION POINT** това не довежда до създаване на електрически контакт между тях.

Два или повече единични извода с едно и също име, зададено с функцията **NAME SIGNAL**, се разглеждат като електрически свързани помежду си, независимо че между тях няма прекарана физическа връзка. Например всички единични изводи със име **VCC** се свързват автоматично помежду си и към захранващия токоизточник посредством "сигналното име **VCC**".

Код на връзките се задава с въвеждане от клавиатурата на:

W <шпация> **6** <**RETURN**> – за връзките към захранване ;
W <шпация> **7** <**RETURN**> – за връзките към земя;

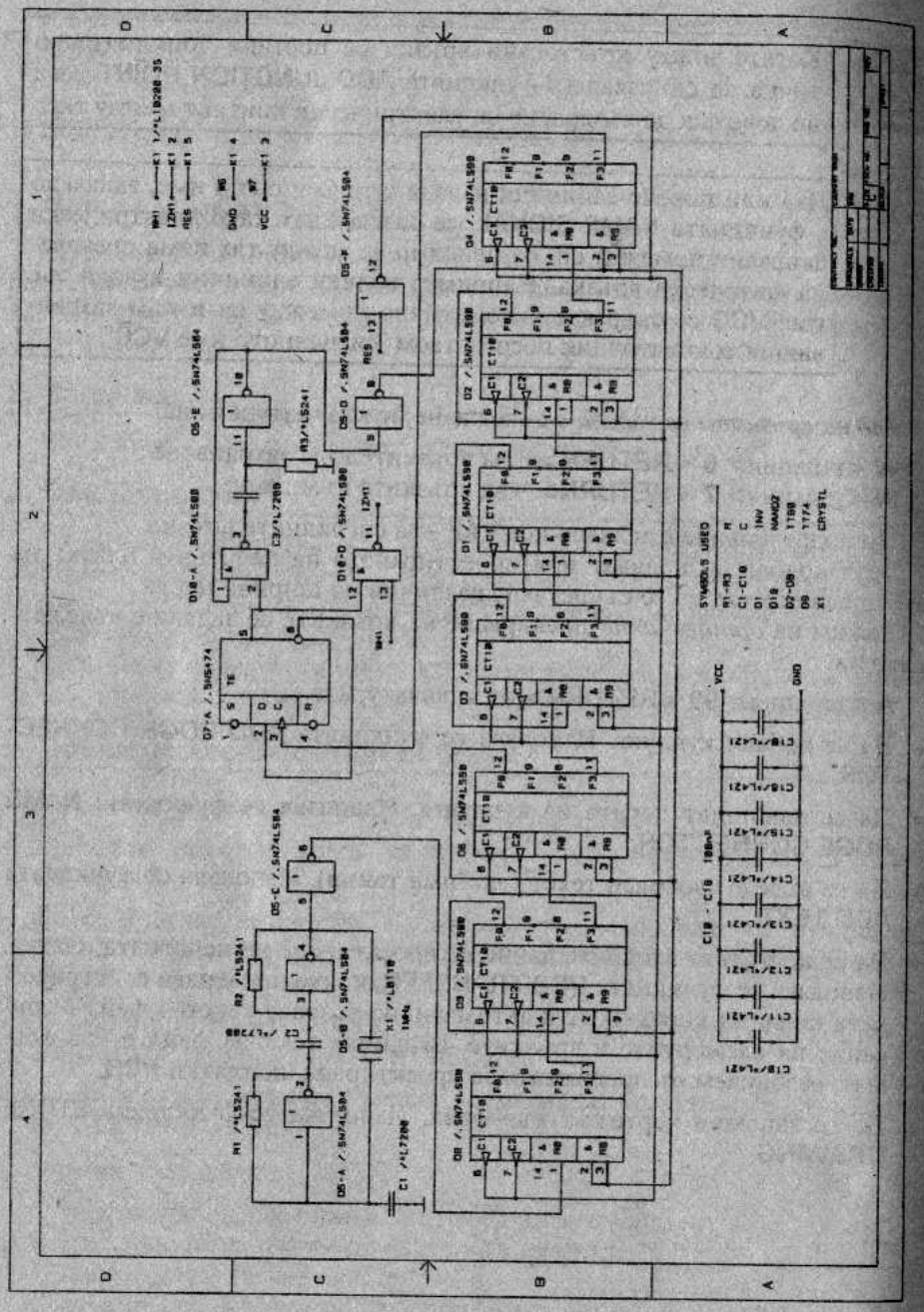
По подразбиране е зададен код **W 2** – за сигналните връзки.

Тези кодове позволяват при проектирането на печатната платка на всеки един от тях да се съпоставят различни по ширина писти.

Режим на *ортогонално* прекарване на връзките се задава с въвеждането на:

A <шпация> **90** <**RETURN**> от клавиатурата.

- Да се добави куплунг. Използва се функцията **ADD EDGE CONNECTOR**.
- Да се надпишат пера на куплунга. Използва се функцията **NAME EDGE CONNECTOR**.
- Да се въведе свободен текст (ако има такъв). Използва се функцията **ADD TEXT**.
- Да се подготвят входните данни за проектиране на печатната платка. Използва се функцията **PCB TRANSFER**, която проверява електрическата схема за коректност и автоматично създава текстов файл с описание на елементите и връзките. Входният текстов файл е във формат, разбираем от програмата за проектиране на платки **PCB**.
- Да се запомни чертежът във файл. Използва се функцията **STORE DRAWING**.



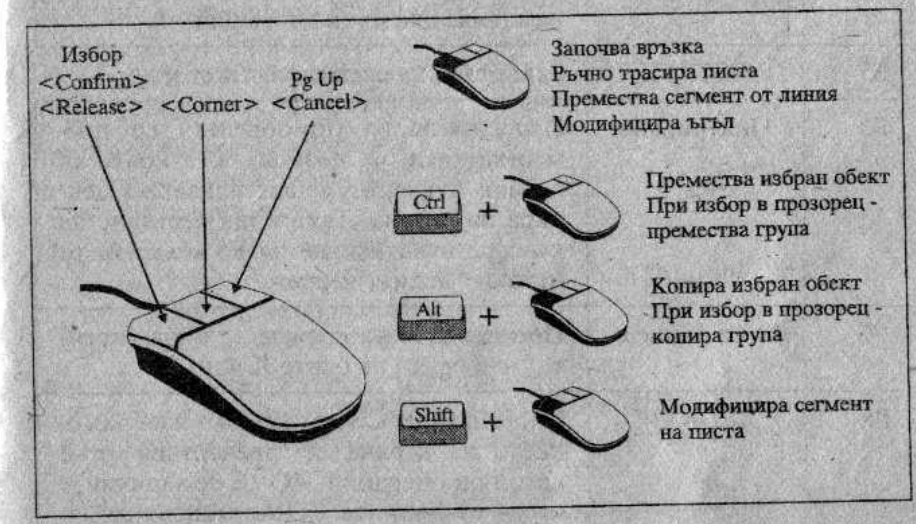
Фиг. 1. Принципна схема.

ПРИЛОЖЕНИЕ № 3

РАБОТА С КЛАВИАТУРА И МИШКА В CADSTAR

1. Мишка

Заменя функциите на клавиатурата за управление движението на курсора, както и действието на част от функционалните клавиши. Левият бутон дублира клавишите **F10**, **Home** и **Enter**, а десният - **PgUp и F1**, както е показано на фигурата по-долу.



На същата фигура са посочени и възможностите за автоматично задаване на команди с помощта на мишката.

2. Клавиатура

Клавиши за управление на курсора – Намират се вдясно на клавиатурата. Стрелките върху тях показват посоката на преместване на курсора.

PgUp – с този клавиш на екрана се изобразяват менютата. Ако в програмата има повече менюта, те се сменят с последователно натискане на същия клавиш.

Home – клавиш за избор, маркиране, указване. С него се избира операция, указва елемент, задава начало и край.

End – клавиш за смяна на стъпката при преместване на курсора.

Клавиш	Функция	Действие
F1	*CANCEL*	Прекратяване на операция, връщане на стъпка назад в програмата, отрицателен отговор на въпрос и др.
F2	Увеличаване	Всяко натискане на F2 увеличава изображението два пъти до мащаб 32:1.
F3	*MOVE*	Преместване на елементи, надписи, изводи на куплунзи, точки за връзка и др.
F4	Намаляване	При натискане на F4 изображението се намалява два пъти до мащаб 1:8.
F5	*ROTATE*	Завъртане на елементи, текст и др.
F6	Центриране (PAN)	Изображението (прозорецът) се премества така, че позицията, в която се намира курсорът, се визуализира в центъра на екрана. Така с преместване на курсора и натискане на F6 може да се "обходи" целият чертеж.
F7	*Move Segment*	Преместване на сегмент, т.е. на отсечка, успоредна на осите X или У.
F8	*REDRAW*	Опресняване. С този клавиш изображението на екрана се "пречертава", тъй като при операции MOVE се заличават части от линиите. Пречертаване се извършва автоматично и при използване на клавиши F2, F4, F6, но с F8 нищо друго от изображението не се променя.
F9	*CORNER*	Ъгъл. При тази операция се създават или видоизменят ъгли.
F10	*CONFIRM / RELEASE*	Клавиш за потвърждение, продължаване на работа, изпълнение на операция. Използва се за завършване на операция, потвърждение, когато има опасност от унищожаване на изображението и др.

ПРИЛОЖЕНИЕ № 4

ОСНОВНИ ФУНКЦИИ НА ПРОГРАМАТА SCM В CADSTAR

1. Общи функции

Стартиране на програмата
- Написва се името на програмата SCM.

Прочитане на схема от файл
- Избира се функцията READ DRAWING от меню CONTROL;
- Файлът се избира от списъка, показан на екрана;
- Програмата изписва коментара към прочетения чертеж и очаква подтвърждение с <CONFIRM>.

Задаване рамка на чертежа
- Преди да започне изчертаването на схемата се зарежда файл със стандартна рамка.

Записване на схемата във файл
- Избира се функцията STORE DRAWING от меню CONTROL;
- Въвежда се името на файла;
- Въвежда се коментар към дадения чертеж.

Приключване на работа с програмата SCM
- Избира се функцията FINISH от меню CONTROL;
- С <CONFIRM> се подтвърждава завършването на работата.

2. Функции за добавяне на обекти

Добавяне на символ за дискретен компонент

Тази команда се използва за компоненти, които не са дефинирани в библиотеката PARTS. След добавянето на символа е необходимо той да бъде наименован с NAME SYMBOL и евентуално да се номерират изводите му с NUMBER PIN.
- Избира се функцията ADD SYMBOL от меню ADD;
- Въвежда се името на символа или се избира символ от екрана;
- Символът се поставя на избраното място, като се премества с мишката или се завърта с клавиша F5;
- Окончателното разполагане на символа става с <CONFIRM>.

Добавяне на символ за интегрална схема

Тази команда се използва за компоненти, които са дефинирани в библиотеката PARTS. Заедно с добавянето на символа автоматично се из-

вършва неговото наименоване и номерирането на изводите му. С тази команда е удобно да се добавят интегрални схеми.

- Избира се функцията ADD PART от меню ADD;
- Въвежда се името на компонента или се избира от екрана;
- Символът се поставя на избраното място, като се премества с мишката или се завърта с клавиша F5;
- Окончателното разполагане на символа става с <CONFIRM>.

Добавяне на перо от куплунг

- Избира се функцията ADD EDGE CONNECTOR от меню ADD;
- Символът за куплунг (+) се поставя на желаното място;
- Фиксирането става с клавиша <CONFIRM>.

Прекарване на връзка

- Избира се функцията ADD CONNECTION от меню ADD;
- За по-лесно прекарване на връзките може да се зададе режим на ортогонално трасиране като се въведе A 90 <RETURN>;
- Ако е необходимо се сменя кодът на връзката: например командата W 7 <RETURN> установява код 7;
- С мишката се избира началната точка на връзката. (Тя може да бъде само извод на елемент или точка от вече прекарана връзка. В последния случай автоматично се поставя точка на свързване;
- Връзката се изчертава като последователност от праволинейни сегменти, а ъглите между тях се създават с клавиша F9;
- Завършването на връзката става като се позиционира курсорът върху нейната крайна точка и се натисне левият клавиш на мишката .

Добавяне на свободен текст

- Избира се функцията ADD TEXT от меню ADD.
- Текстът се въвежда от клавиатурата, позиционира се на конкретно място, ако е необходимо се завърта с F5, след което се потвърждава с <CONFIRM>.

Добавяне на точка на свързване

- Избира се функцията ADD JUNCTION POINT от меню ADD;
- Позиционира се точката на свързване и се натиска Home.

3. Функции за наименоване на обекти

Наименоване на дискретен компонент

Тази функция се използва за компоненти добавени с ADD SYMBOL.

- Избира се функцията NAME SYMBOL от меню INTERACTION;
- Въвежда се името на компонента в следния формат :
  <име> /*L<библиотечен номер>
  например R1 /*L5241
- Символът се избира с мишката и надписът се фиксира с <CONFIRM>.

Надписване на перата на куплунг

- Избира се NAME EDGE CONNECTOR от меню INTERACTION;
- Въвежда се името на първото перо от куплунга в следния формат :
  <име> <номер на перо>/*L<библиотечен номер> <брой пера>
  например K1 1/*L10200 35
- Избира се съответното перо и надписът се фиксира с <CONFIRM>;
- Останалите пера от куплунга се надписват по аналогичен начин, като форматът за тях е следният :
  <име> <номер на перо>, например K1 2.

Наименоване на сигнал

- Избира се функцията NAME SIGNAL от меню INTERACTION;
- Въвежда се името на сигнала (напр. VCC);
- Избира се единичен извод (точка за връзка JUNCTION POINT) или извод на елемент;
- Надписът се фиксира с <CONFIRM>.

4. Функции за редактиране

Премахване на обект (символ, връзка, текст и др.) :

- Избира се функцията REMOVE от меню INTERACTION;
- Избира се обектът;
- Избраният обект изчезва от екрана. Ако сега се натисне <CONFIRM>, то обектът ще бъде премахнат от схемата;
- Ако избрания обект е свързан символ, то преди той да изчезне от екрана се появява съобщението "Connected Item", на което трябва да се отговори с <CONFIRM> или <CANCEL>.

Преместване на обект (не се отнася за връзките!):

- Избира се MOVE от меню INTERACTION или се натиска F3;
- Избира се обектът и той започва да се движи заедно с курсора;
- След позициониране на обекта операцията се завършва с натискане на левият клавиш на мишката.

Копиране на обект (не се отнася за връзките!)

- Избира се функцията COPY от меню ADD;
- Избира се обектът и той започва да се движи заедно с курсора;
- След позиционирането на обекта операцията се завършва с натискане на левият клавиш на мишката.

Ако елементът е надписан, той се копира на новото място под същото име, което води до дублиране на означенията за два различни обекта.

Завъртане на обект

- Избира се функцията ROTATE от меню INTERACTION;
- Избира се обектът;

- При всяко натискане на клавиш **F5** избраният обект се завърта на 90 градуса срещу часовниковата стрелка;
- След избиране на новата ориентация на обекта, операцията се завършва с **<CONFIRM>**;

Преместване на сегмент от връзка

- Избира се функцията **MOVE SEGMENT** от меню **INTERACTION** или се натиска клавишът **F7**;
- Избира се сегмент от връзка. Той започва да се движи заедно с курсора, като остава успореден на първоначалното си положение;
- С натискане на левият клавиш на мишката сегмента се фиксира.

Преместване, копиране или премахване на група обекти

- Избира се съответо функцията **MOVE GROUP**, **COPY GROUP** или **REMOVE GROUP** от меню **UTILITY**;
- Задава се областта, в която е разположена групата с клавишите за управление на курсора и **F9**. Избраните обекти сменят цвета си;
- Потвърждават се избраните вече елементи (с **<CONFIRM>**) или се добавят или отстраняват елементи чрез щракване с мишката;
- След позиционирането на групата операцията се завършва с **<CONFIRM>**;

Промяна на текст

- Избира се функцията **EDIT TEXT** от меню **INTERACTION**;
- Избира се текстът и се редактира с клавишите за управление на курсора и клавиша **Backspace**;
- Операцията завършва с **<CONFIRM>**.

5. Помощни функции

Намиране на прозорец

- Избира се функцията **WINDOW FIND** от меню **CONTROL**;
- Въвеждат се X и Y координати или името на елемента. На указаното място върху екрана се появява знак @;

Тази функция е удобна за бързо придвижване до определен обект или координата от чертежа.

Част от прозорец

- Избира се функцията **FRAME WINDOW** от меню **CONTROL**;
- Ако е необходимо да се увеличи даден участък от екрана, на въпроса **FRAME INWARDS?** се отговаря с **<CONFIRM>**, след което се очертават контурите на прозореца. При отговор с **<CANCEL>** цялата схема се визуализира намалена в левия долен ъгъл на екрана и с помощта на изчертан прозорец се уголемява определен участък от нея;

Параметри на екрана

- Избира се функцията **DISPLAY OPTIONS** от меню **CONTROL**;

- От появилото се меню се избират цветове за изобразяване на определени категории обекти, включват се или се изключват обектите за визуализиране, задава се помощна мрежа при изчертаването и др.

6. Функции за връзка с програмата за печатни платки PCB

Създаване на входни данни за платка

- Избира се функцията **PCB TRANSFER** от меню **UTILITY**;
- По-нататъчния диалог е илюстриран на следната фигура;

```
PCB TRANSFER MK 7.3

Component file name                  [.CDI] > TEST          1, 2
Connections file name                [.CDI] > TEST.CDI
Part names file name                 [.CDI] > TEST.CDI
Errors file name                     [.REP] > TEST          3
Errors listing device                  [N] >                4
Unnumbered terminals listing device [N] >                   5
Unnamed edge connector name                > ERROR         6
Unnamed edge connector library number      > 9999          7
```

- Въвежда се име на файл, в който ще се запишат входните данни (1);
- С **<CONFIRM>** се потвърждава, че данните ще се генерират само в един файл (2);
- Въвежда се име на файл, в който ще се запишат откритите при трансформацията грешки (3). Файлът е с разширение **.REP**;
- Указва се устройството, където да се изведе списъкът с грешки и списъкът на неизползваните изводи (4,5). Ако се въведе **S**, данните се визуализират на екрана, а при **N** - не се извеждат. Тази информация ще се запишат в файла за грешки и затова не е необходимо да я извеждаме и на друго устройство. Натиска се **<RETURN>**;
- Въвежда се име и библиотечен номер за ненадписан куплунг, ако случайно е останал такъв в схемата (6,7). В чертежа не трябва да има такива куплонзи! Задава се произволно име и номер;

```
INPUT DRAWING NAMES

> TEST
>
```

- Задава се името на файла със схемата. След появяването на втория промпт ">" се натиска **<RETURN>**;

```
ACCEPT DRAWING LIST                          CONFIRM OR CANCEL

  Drawings                                 Hier  drawings
  --------                                 --------------
  TEST.CSD
```

- Натиска се **<CONFIRM>** ;

```
PRINT DRAWING LIST                          CONFIRM OR CANCEL

  Drawings                                  Hier  drawings
  --------                                  --------------
  TEST.CSD
```

Натиска се **<CANCEL>**.

Ако по погрешка се натисне **<CONFIRM>** програмата се опитва да отпечата списъка на чертежите на принтер. В този случай трябва да се изчака известно време преди програмата да продължи работа.

```
PRINT END OF RUN REPORT                                  CONFIRM OR CANCEL

  PCB TRANSFER MK 7.3
  End of run report
  -----------------
  TEST.CDI contains 26 components
                    100 connections in 29 trees
                    10 components with part names

  TEST.REP contains 0 warnings and 0 errors
```

Това е отчетът за трансформирането на схемата. На последния ред е показан броят на предупредителните съобщения (warnings) и грешките (errors). Ако те са различни от нула трябва да се прочете съдържанието на файла с грешки и да се отстранят в схемата. След това се повтаря операцията **PCB TRANSFER**.

Програмата предлага да отпечата този отчет. Трябва да се натисне **<CANCEL>**.

```
PROCESSING COMPLETED                    ANOTHER RUN?  CONFIRM OR CANCEL
```

Натиска се **<CANCEL>**.

Отразяване в схемата на извършените промени в платката :

- Избира се функцията **BACK ANNOTATION** от меню **UTILITY**;

Въвежда се името на файла, където е записана информацията за извършваните в платката промени (файлът е с разширение **.RIN**) ;

ТЕМА № 10

ПРОЕКТИРАНЕ НА ПЕЧАТНА ПЛАТКА. АВТОМАТИЗИРАНО РАЗПОЛАГАНЕ НА ЕЛЕМЕНТИТЕ

1. Цел на упражнението

Да се придобие пратически опит за проектиране на печатна платка с автоматизирани средства и да се усвоят умения за автоматизирано разполагане на дискретни елементи и интегрални схеми, както и за оптимизиране на разположението по отношение дължината на връзките.

2. Задание

Да се разположат елементите върху печатната платка като разположението им се оптимизира по отношение дължината на връзките.

3. Указания

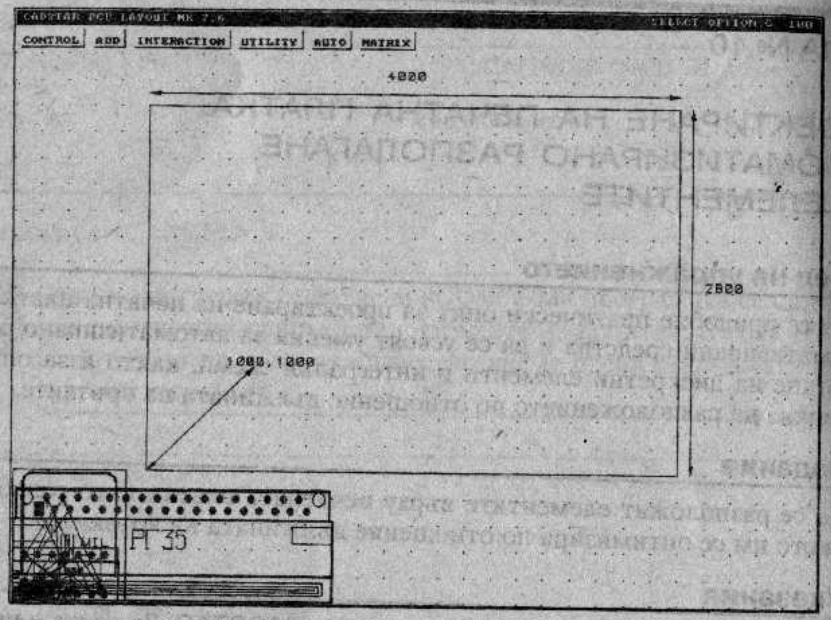
- Да се използва програмата **PCB** от пакета **CADSTAR**. За целта е необходимо да се запознаете с операциите, описани в Приложение 5 "Основни функции на програмата **PCB** в **CADSTAR** ".

4. Последователност при работа

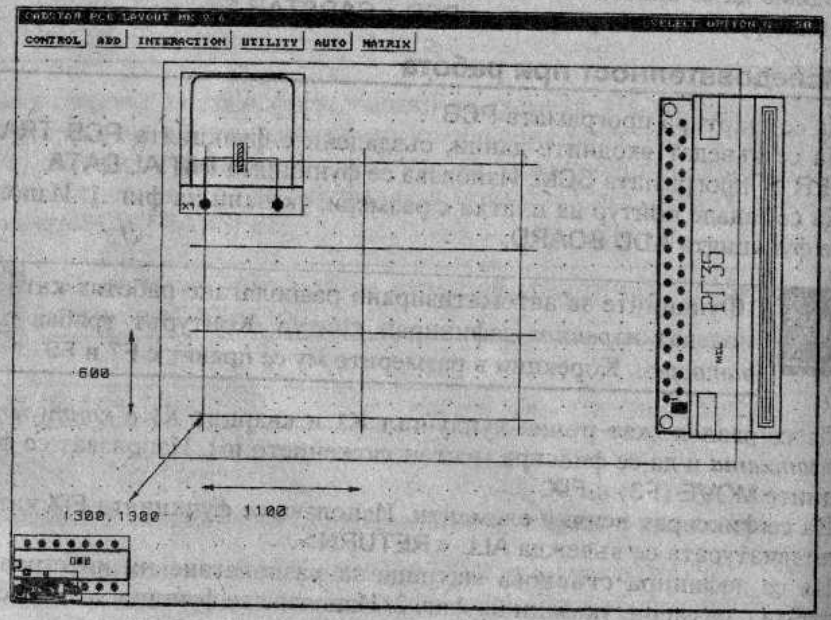
- Да се стартира програмата **PCB**.
- Да се въведат входните данни, създадени с функцията **PCB TRANSFER** от програмата **SCM**. Използва се функцията **INITIAL DATA**.
- Да се зададе контур на платка с размери, указани на фиг. 1. Използва се функцията **ADD BOARD**.

Функциите за автоматизирано разполагане работят като използват *първият* дефиниран контур. Контурът трябва да е *затворен*. Корекции в размерите му се правят с **F7** и **F9**.

- Да се разположат ръчно куплунгът **K1** и кварцът **X1** *в контурът на платката* и да се фиксира местоположението им. Използват се функциите **MOVE (F3)** и **FIX**.
- Да се фиксират всички елементи. Използва се функцията **FIX** като от клавиатурата се въвежда **ALL <RETURN>**.
- Да се дефинира стъпкова матрица за разполагане на интегралните схеми с размери, указани на фиг. 2. Използва се функцията **ADD STEP MATRIX**.



Фиг. 1. Размери на контура на платката

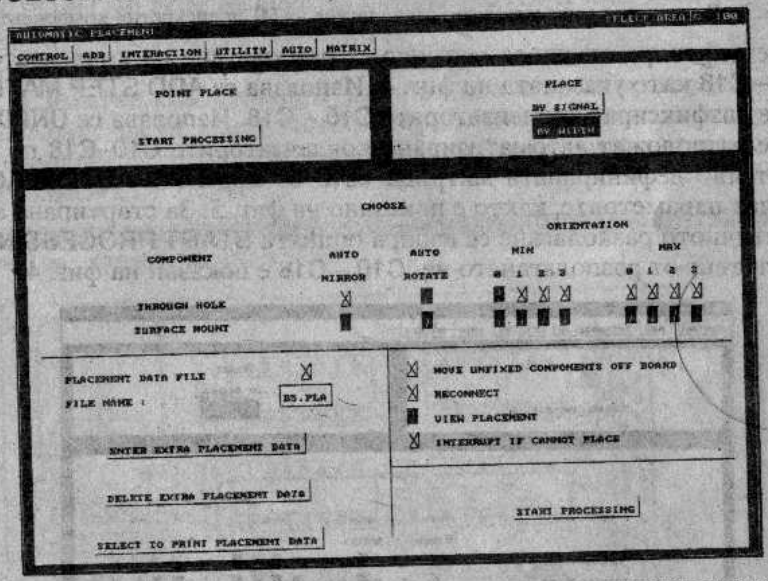


Фиг. 2. Размери на стъпковата матрица

Стъпковата матрица има равни разстояния между линиите по *x* от една страна и линиите по *y* от друга и елементите се разполагат във възлите ѝ с базовата си точка (долен ляв ъгъл на корпуса).

За корекции на матрицата се използват функциите за редактиране на матрици **ADD MATRIX LINE, REMOVE MATRIX LINE, CHANGE STEP MATRIX, STRECH STEP MATRIX.**

- Да се разфиксират интегралните схеми **D1 – D10.** Използва се функцията **UNFIX** като от клавиатурата се въвежда **D(1 10)<RETURN>.**
- Да се разположат автоматизирано интегралните схеми по възлите на предварително дефинираната стъпкова матрица като от подменю **AUTOPLACE** се зададат параметрите, както е показано на фиг. 3. За стартиране на автоматичното разполагане се избира опцията **START PROCESSING.**

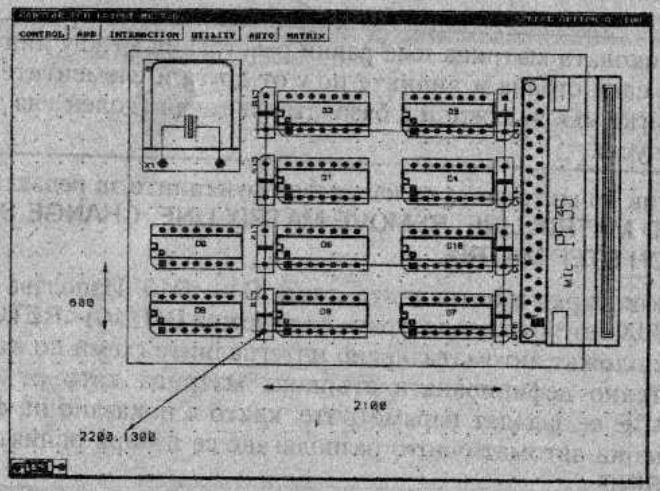


Фиг. 3. Задаване параметрите за автоматично разполагане на интегралните схеми

Резултатът от операцията е показан на фиг. 4.

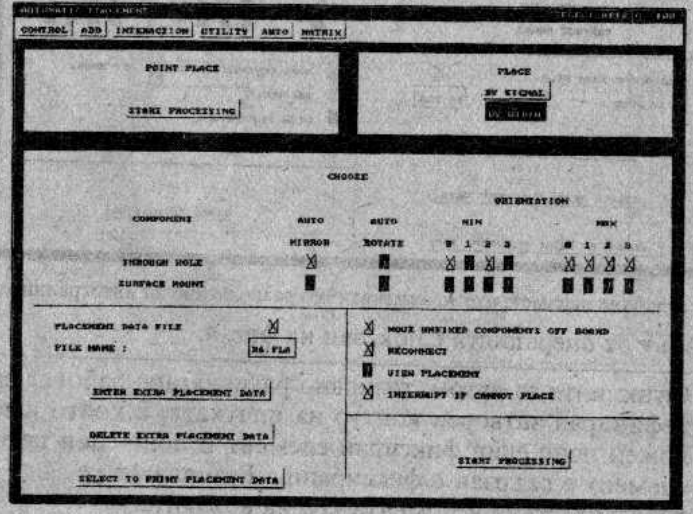
Функциите за автоматизирано разполагане работят само при дефиниран затворен контур на платката, в който има разположен поне един фиксиран елемент и поне един нефиксиран елемент е свързан с фиксирания. Разполагат се *само* елементите с указания в момента код на връзките.

- Да се фиксират разположените интегралните схеми. Използва се **FIX.**



Фиг. 4. Резултат от автоматичното разполагане на ИС и дискретни компоненти

- Да се дефинира стъпкова матрица за разполагане на кондензаторите C10 – C18 като указаната на фиг. 4. Използва се ADD STEP MATRIX.
- Да се разфиксират кондензаторите C10 – C18. Използва се UNFIX.
- Да се разположат автоматизирано кондензаторите C10–C18 по предварително дефинираната матрица като от подменю AUTOPLACE се зададат параметрите, както е показано на фиг. 5. За стартиране на автоматичното разполагане се избира опцията START PROCESSING. Резултатът от разполагането на C10 – C18 е показан на фиг. 4.



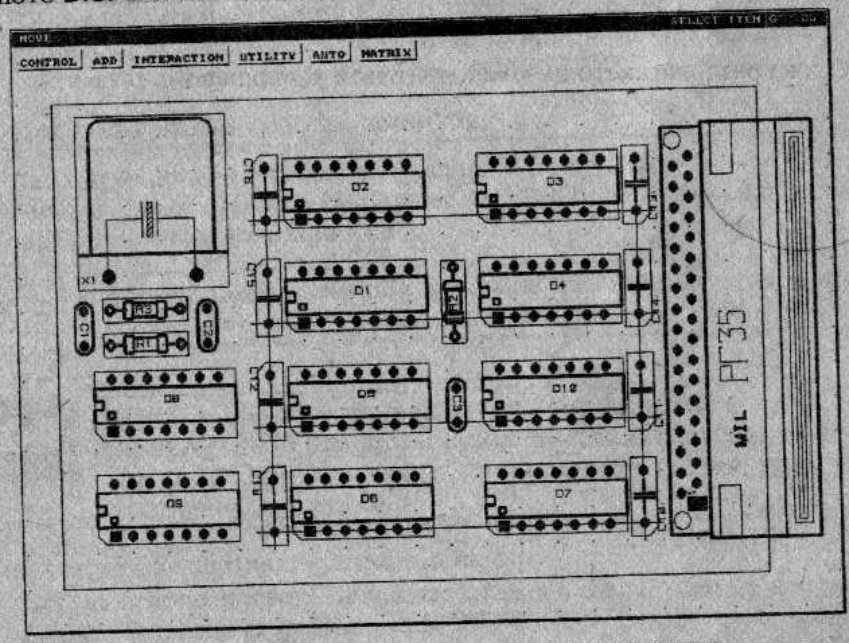
Фиг. 5. Задаване параметрите за автоматичното разполагане на кондензаторите

- Ръчно да се разположат останалите дискретни компоненти, както е показано на фиг. 6 като предварително се разфиксират.
- Да се минимизира дължината на връзките. Използва се функцията RECONNECT. Конкретната стойност на общата дължина на връзките може да се провери с функцията CONNECTION LENGHT.

За да се наблюдава резултатът от операцията, предварително връзките между елементите се възстановяват като видими с определен цвят от подменю DISPLAY OPTIONS.

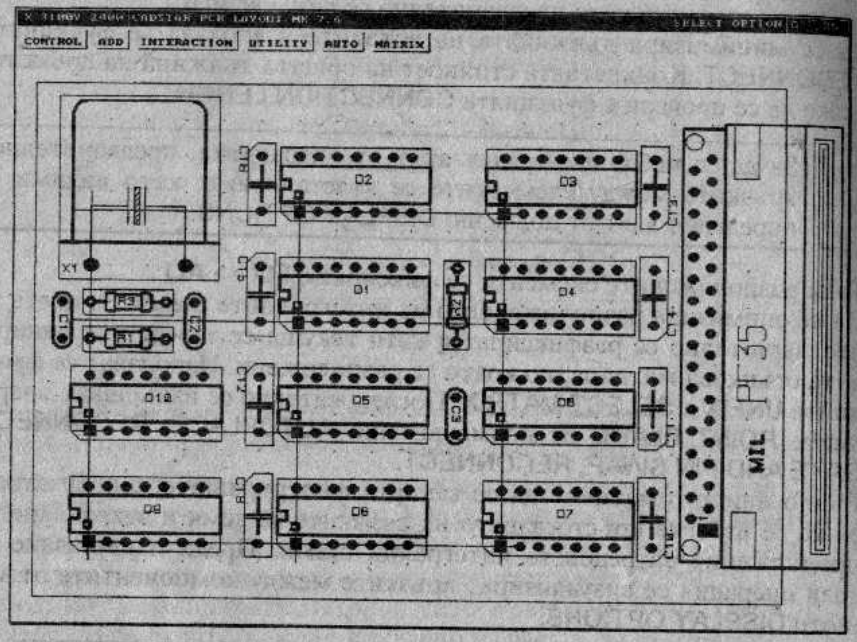
Така разположените елементи се фиксират с FIX => ALL.

- Да се оптимизира разположението на интегралните схеми. За целта те предвари-телно се разфиксират и като текуща се зарежда дефинираната стъпкова матрица, по която са разположени. Използват се функциите UNFIX и SELECT MATRIX. Последователно се извършват операциите POINT PLACE от подменюменю AUTOPLACE, RECONNECT, GATE AND PIN SWAP, RECONNECT.
- Ръчно или автоматично да се завъртя филтриращите кондензатори, за да се избегне кръстосването на връзките за земя и захранване по продължение на редовете интегрални схеми. Преди извършване на тази операция се визуализират връзките между компонентите от менюто DISPLAY OPTIONS.



Фиг. 6. Ръчно разполагане на дискретните компоненти

Резултатът след оптимизиране на разположението е даден на фиг. 7.



Фиг. 7. Резултат след оптимизиране на разположението

С това разполагането на компонентите е завършено.

ТЕМА № 11

ПРОЕКТИРАНЕ НА ПЕЧАТНА ПЛАТКА. АВТОМАТИЗИРАНО ТРАСИРАНЕ НА ВРЪЗКИТЕ

1. Цел на упражнението

Да се придобие пратически опит за проектиране на печатна платка с автоматизирани средства и да се усвоят умения за трасиране на междусъединенията и за извършване на необходимите проверки на проекта.

2. Задание

Да се трасират междусъединенията на печатната платка като топологията на получените писти се сравни с връзките от изходната електрическа схема и се провери за възможност да бъде произведена по зададена технология.

3. Указания

- Да се използва програмата PCB от пакета CADSTAR. За целта е необходимо да се запознаете с операциите, описани в Приложение 5 "Основни функции на програмата PCB в CADSTAR".

4. Последователност при работа

- Да се извърши автоматично трасиране на връзките към земя и захранване. Да се използва функцията AUTOROUTE като параметрите се зададат, както е показано на фиг. 1.

Алгоритъмът позволява връзките към земя и захранване се прекарват с три сегмента, разположени върху един слой.

За да се облекчи намирането на тези връзки, в подменюто DISPLAY OPTIONS да се зададе различен цвят за визуализация на връзките с код 6 и 7, а сигналните връзки (с код 2) да се направят невидими.

Резултатът от операцията е показан на фиг. 2.

- Останалите неопроводени връзки за земя и захранване да се прекарат ръчно. Използва се функцията MANUAL ROUTE като хоризонталните писти се прекарват на слой 1, а вертикалните на слой 16.

Смяна на слоя, на който се прекарват връзките, става като от клавиатурата се въведе L<RETURN>.

Примерно опроводяване на шините за земя и захранване е показано на фиг.3.

- Да се трасират останалите връзки. Опциите от подменю **AUTOROUTE** се задават, както е показано на фиг.4.

Връзките от тип "памет" се прекарат с пет сегмента, разположени върху един слой, при използване на ъгъл от 45 между сегментите.

Ортогоналните връзки се разполагат върху два слоя – в хоризонтално направление – върху единия, а във вертикално – върху другия като пакетът автоматично поставя преходни отвори.

Тук броят на сегментите може да се зададе 5, 7 или 9.

Резултатът от автоматичното трасиране на сигналните връзки е показан на фиг. 5

- Ръчно да се трасират останалите връзки като хоризонталните сегменти се задават в слой 1, а вертикалните – в слой 16.

Всички връзки да се възстановят като видими от менюто **DISPLAY OPTIONS**.

Смяна на слой на вече прекаран сегмент от писта може да се извърши с функцията **SWAP SEGMENT**.

- Да се извърши автоматизирано минимизиране на преходните отвори. Използва се функцията **AUTOROUTE** като в подменюто се разрешава за изпълнение опцията **VIA MINIMISATION**, а всички други опции се забраняват.
- Да се извърши проверка за съответствие на прекараните писти с връзките от изходната принципна електрическа схема. Използва се функцията **CONNECTION CHECK**, която изисква името на файла с входните данни за платката (с разширение .CDI). Ако при проверката се срещнат несъответствия, програмата дава подходящи съобщения.
- Да се оформи окончателно платката като се нанесат необходимите корекции и довършителни работи. Използват се функциите **MITRE CORNER, FILLET CORNER, ADD COPPER, SWAP SEGMENT, ADD TEXT**.

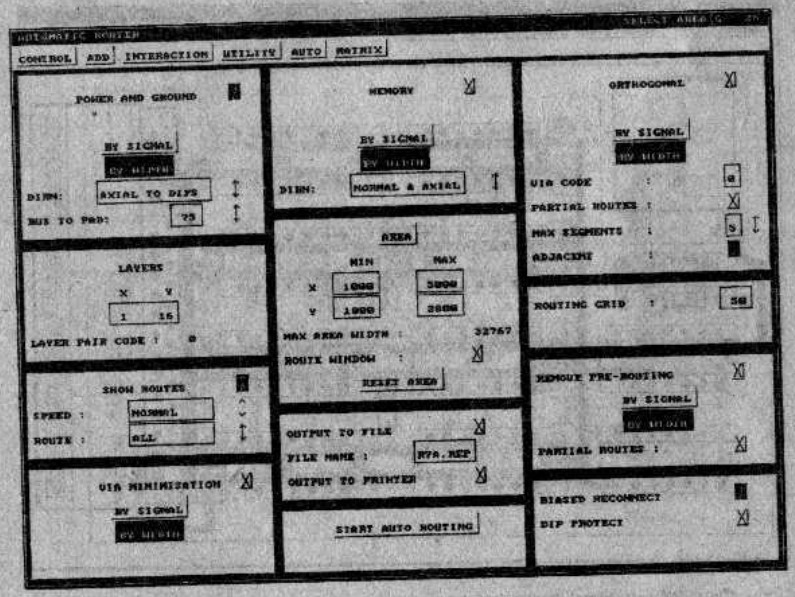
Примерен вариант на окончателно оформената платка е даден на фиг.6.

- Да се извърши проверка на технологичните норми за производство на платката. Използва се функцията **DIMENSION CHECK**. От появилото се подменю се задават:
  - ◊ Слоевете, които ще се проверяват – **LAYERS** – 1 и 16;
  - ◊ Обектите, които ще се контролират – избира се **CHECK ALL**, което задава проверка на всички обекти;
  - ◊ Областта, в която ще се извършва проверката – избира се **CHECKING BY LAYER** (т. е. проверката ще се извършва по целия слой);

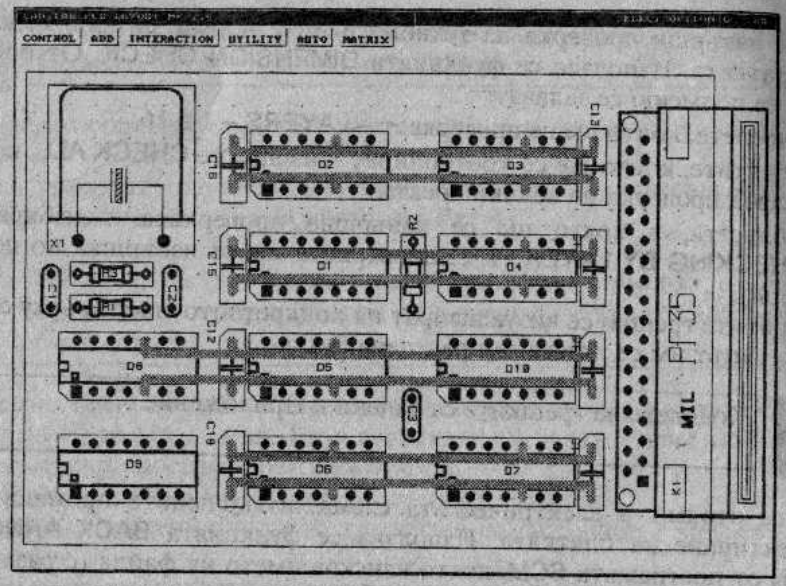
Откритите грешки се визуализират на конкретното място върху екрана като типът им се указва с определена буква.

Кодовете на грешките са дадени в Приложение 5.

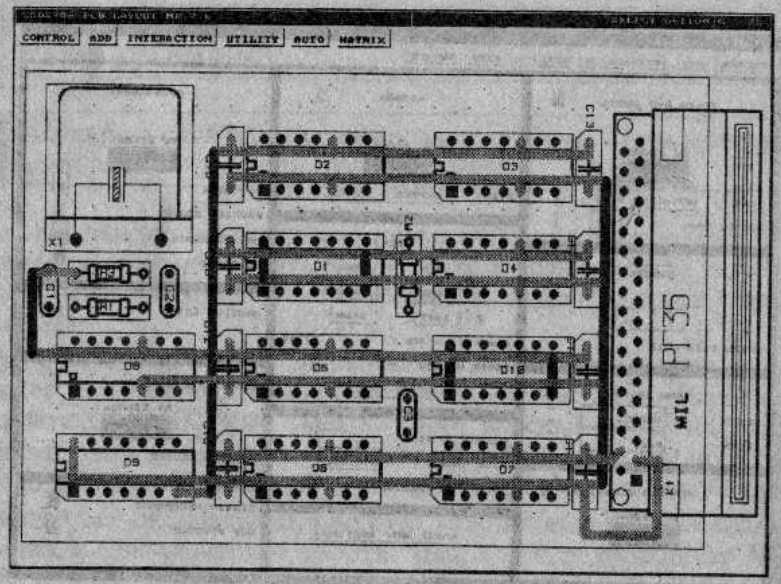
- Да се отразят в електрическата схема, извършените промени при проектиране на платката. Използва се функцията **BACK ANNOTATION** от програмата **SCM**, която изисква името на файла (с разширение .RIN), в който са запазени направените промени при операцията **GATE AND PIN SWAP**.



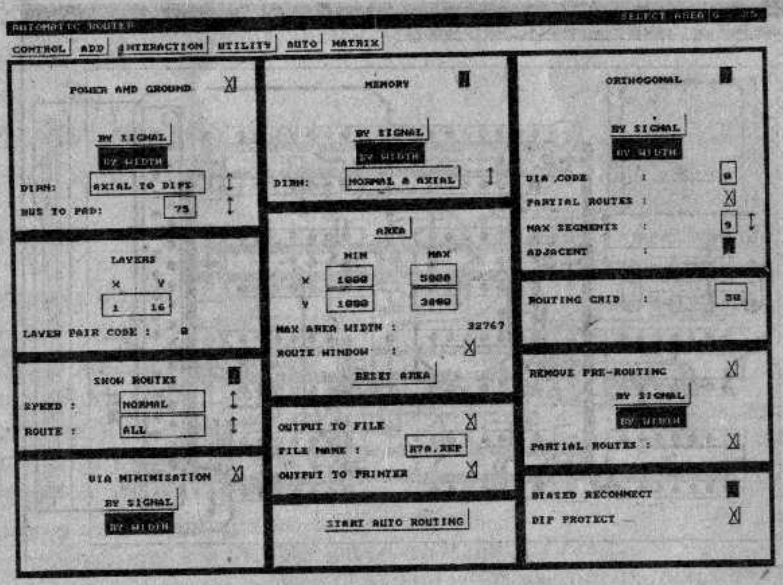
Фиг. 1 Параметри за автоматично трасиране на връзките за земя и захранване



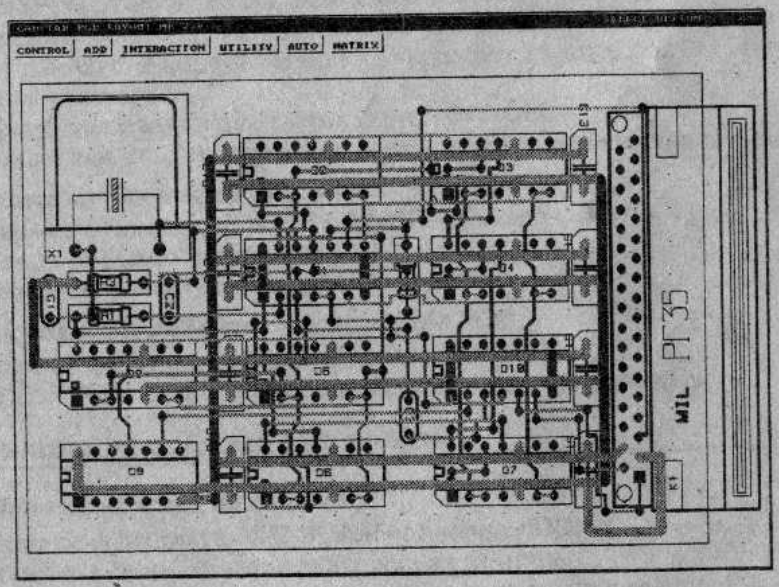
Фиг. 2 Резултат от автоматичното трасиране на земя и захранване



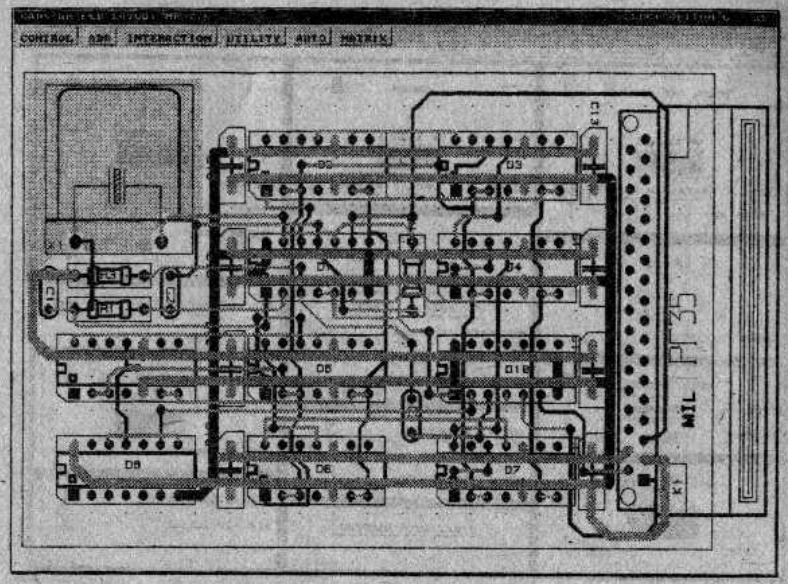
Фиг. 3 Примерно опроводяване на връзките за земя и захранване



Фиг. 4 Задаване параметрите за опроводяване на сигналните връзки



Фиг. 5 Резултат от автоматичното трасиране на сигналните връзки



Фиг. 6 Примерен вариант на окончателно оформена платка

ПРИЛОЖЕНИЕ № 5

ОСНОВНИ ФУНКЦИИ НА ПРОГРАМАТА PCB В CADSTAR

1. Общи функции

Стартиране на програмата

- Написва се името на програмата PCB

Прочитане на схема от файл

- Избира се функцията READ LAYOUT от меню CONTROL;
- Въвежда се името на файла или се избира от списъка на екрана;
- Програмата изписва коментара към прочетения чертеж и очаква подтвърждение с <CONFIRM>;

Прочитане на входни данни за платката

- Избира се функцията INITIAL DATA от меню UTILITY;
- Въвежда се името на файла (файловете) с входни данни. (Входните данни са предварително създадени автоматично с функцията PCB TRANSFER от програмата SCM или с текстов редактор).
- С мишката се избира полето ADD TO CURRENT LAYOUT за да се прочетат данните. Ако при четенето се срещнат грешки се дава подходящо съобщение;
- Операцията се завършва с функцията <CANCEL>.
  В долния ляв ъгъл на екрана се появяват компонентите на схемата;

Записване на платката във файл

- Избира се функцията STORE LAYOUT от меню CONTROL;
- Въвежда се името на файла;
- Въвежда се коментар към дадения чертеж.

Приключване на работа с програмата PCB

- Избира се функцията FINISH от меню CONTROL;
- С <CONFIRM> се подтвърждава завършването на работата.

2. Функции за добавяне на обекти

Добавяне на контур на платка

- Избира се функцията ADD BOARD от меню ADD;
- С мишката се избира начална точка на контура;
- Контура се описава като на всеки ъгъл се натиска клавиша F9;
- Последната точка на контура трябва да съвпада с първата.

Така дефинираният контур се използва при операциите AUTOPLACE и AUTOROUTE.

Функциите AUTOPLACE и AUTOROUTE работят само в *първия* дефиниран контур на платката.

Добавяне на компонент от библиотеката

Използва се, за да се добави към платката допълнителен компонент, невключен в първоначалните входни данни (например допълнителен развързващ кондензатор или компонент, подобряващ характеристиките на схемата).

- Избира се функцията ADD COMPONENT от меню ADD;
- Въвежда се номер на корпус (напр. 1024) или име на компонент (напр. SN74LS00);
- Въвежда се име на компонента (напр. IC201);
- Появява се контур на корпус, който се позиционира на избраното място като се движи с мишката или се завърта с клавиша F5;

Ако името на компонента започва с цифра преди него се поставя точка - например .74LS00.

Добавяне на връзка

Използва се, за да се свърже допълнително въведен компонент, невключен в първоначалните входни данни, към останалите компоненти върху платката.

- Избира се функцията ADD CONNECTION от меню ADD;
- Ако е необходимо се сменя кода на връзката. Например ако от клавиатурата се въведе W 7 <RETURN> се установява код 7;
- С мишката се указват началото и края на връзката.
- Прекараната връзка се потвърждава с <CONFIRM> или се отхвърля с <CANCEL>.

Добавяне на свободен текст

- Избира се функцията ADD TEXT от меню ADD.
- Въвежда се текстът от клавиатурата и с мишката се разполага на подходящо място.

Добавяне на медна област

- Избира се функцията ADD COPPER от меню ADD;
- Задава се контурът, в който ще има фолиран участък. Процедира се аналогично, както при операцията ADD BOARD.

3. Функции за редактиране

Премахване на обект

- Избира се функцията REMOVE от меню INTERACTION;
- Обектът се избира с мишката;
- Избраният обект изчезва от екрана. Ако сега се натисне <CONFIRM>, то обектът ще бъде премахнат от схемата;
- Ако избрания обект е свързан компонент, то преди той да изчезне от екрана се появява съобщението "Connected Component", на което трябва да се отговори с <CONFIRM> или <CANCEL>;
- Потвърждаването на операцията се извършва с <CONFIRM> или се отхвърля с <CANCEL>;

Преместване на обект (не се отнася за връзките!)

- Избира се функцията MOVE от меню INTERACTION или се натиска клавиша F3;
- Обектът се избира с мишката;
- Избрания обект се позиционира на желаното място, като се движи с мишката или се завърта с клавиша F5;

Копиране на обект (не се отнася за връзките!)

- Аналогично на *Преместване на обект*, като вместо MOVE се избира функцията COPY.

Завъртане на обект

- Избира се функцията ROTATE от меню INTERACTION или се натиска клавиша F5;
- Обектът се избира с мишката;
- При всяко натискане на клавиш F5 избраният обект се завърта на 90 градуса срещу часовниковата стрелка;
- След избиране новата ориентация на обекта, операцията се завършва с натискане на левия клавиш на мишката.

Преместване, копиране или премахване на група обекти

- Избира се съответно функцията MOVE GROUP, COPY GROUP или REMOVE GROUP от меню UTILITY;
- С мишката и клавиша F9 се описва контур около групата обекти. Избраните обекти сменят цвета си;
- Чрез посочване с мишката, към групата могат да се добавят или премахнат обекти. Окончателното подтвърждаване на избора се извършва с клавиша F10;
- Операцията се завършва с <CONFIRM>.

Промяна на текст

- Избира се функцията EDIT TEXT от меню INTERACTION;
- Текстът се избира с мишката;
- Така избраният текст се редактира с клавишите за управление на курсора и клавиша Backspace;
- Операцията завършва с <CONFIRM> (запазване на промените) или <CANCEL> (възстановяване на изходния текст);

Промяна на ъгъл

Тази функция се използва за добавяне и редактиране на ъгли от писти и контури.

- Избира се функцията **CORNER** от меню **INTERACTION** или се натиска клавиша **F9**;
- С мишката се посочва участък от писта или контур, който трябва да се променя;
- Корекциите се извършват с преместване на мишката и се потвърждават с натискане на левият ѝ клавиш.

Преименоване на компонент
- Избира се функцията **RENAME COMPONENT** от меню **UTILITY**;
- Компонентът се избира с мишката;
- Въвежда се новото име на компонента (напр. **IC202**);
- Новото име се подтвърждава с **<CONFIRM>**.

Смяна на компонент
- Избира се функцията **REPLACE RART** от меню **UTILITY**;
- Следва въпрос дали да се заменят всички компоненти от даден тип или само един от тях. Отговаря се с **<CONFIRM>** (за всички компоненти) и с **<CANCEL>** (само за конкретно указан компонент);
- Въвежда се името на новия компонент (напр. **74ALS00**);
- Ако се заменя само конкретен компонент той се избира с мишката.

Смяна на корпус на компонент
- Избира се функцията **REPLACE COMPONENT** от меню **UTILITY**;
- Следва въпрос дали да се заменят всички корпуси с даден библиотечен номер или само един от тях. Отговаря се с **<CONFIRM>** (за всички корпуси) и с **<CANCEL>** (само за конкретно указан корпус);
- Въвежда се библиотечния номер на новия корпус (напр. **1316**);
- Ако се заменя само корпуса на конкретен компонент той се избира с мишката.

Фиксиране / разфиксиране на компонент

Използва се, за да забрани / разреши на конкретни компоненти да променят местоположението си върху платката.
- Избира се функцията **FIX / UNFIX** от меню **INTERACTION**;
- Избира се компонентът, който ще се фиксира / разфиксира;
- *Група компоненти* се фиксират / разфиксират чрез въвеждане от клавиатурата на:
  ALL <RETURN> – за всички компоненти
  C (1 20) <RETURN> – за кондензатори с имена от **C1** до **C20**
- Операцията се завършва с **<CONFIRM>**.

Преместване на сегмент

Тази финкция се използва за редактиране на линейни сегменти от писти, контури на медни области, контур на платка и т.н.
- Избира се функцията **MOVE SEGMENT** от меню **INTERACTION** или се натиска клавиша **F7**;
- Сегментът се избира с мишката;

- Избраният сегмент започва да се движи заедно с курсора, като остава успореден на първоначалното си положение;
- След позициониране на сегмента операцията се завършва с натискане на левия клавиш на мишката.

Прехвърляне на сегмент

Използва се, за да се прехвърли сегмент от писта от един слой на друг. При това автоматично се добавят или премахват преходни отвори.
- Избира се функцията **SWAP SEGMENT** от меню **INTERACTION**;
- Сегментът се избира с мишката;
- Задава се номера на новия слой и се натиска **<RETURN>**.

Вмъкване на сегмент

Използва се, за да се добави сегмент във вече съществуваща писта или контур.
- Избира се функцията **INSERT SEGMENT** от меню **INTERACTION**;
- С мишката се указват началото и края на новия сегмент;
- Сегмента се поставя на желаното място и се фиксира с натискане на левия клавиш на мишката.

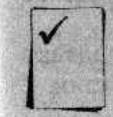
Скосяване на ъгъл
- Избира се функцията **MITRE CORNER** от меню **INTERACTION**;
- Избира се ъгъл от писта или контур и с мишката се оформя желаната дълбочина на скосяване.

Закръгляване на ъгъл
- Избира се функцията **FILLET CORNER** от меню **INTERACTION**;
- Избира се ъгъл от писта или контур и с мишката се избира подходящ радиус на закръгление.

Ръчно трасиране
- Избира се функцията **MANUAL ROUTE** от меню **INTERACTION**;
- Избира се връзката, която ще се трасира;
- Трасирането се извършва с преместване на мишката и с натискане на клавиша **F9** в местата където трябва да има ъгли.

За смяна на слоя при прекарване на пистата от клавиатурата се въвежда **L <RETURN>**. При това автоматично се поставя проходен отвор.

- Операцията се завършва с **<CONFIRM>**.



Частично трасирани писти се завършват с натискане на **P** (от клавиатурата) и **<CONFIRM>**. Така част от връзката се трасира до определено място, а останалата част се запазва като връзка.

4. Помощни функции

Намиране на прозорец

- Избира се функцията WINDOW FIND от меню CONTROL;
- Въвеждат се *x* и *y* координати или името на компонента. На указаното място върху екрана се появява знак @;

Част от прозорец

- Избира се функцията FRAME WINDOW от меню CONTROL;
- Ако е необходимо да се увеличи даден участък от екрана, на въпроса FRAME INWARDS? се отговаря с <CONFIRM>, след което се очертават контурите на прозореца. При отговор с <CANCEL> цялата платка се визуализира намалена в левия долен ъгъл на екрана и с помощта на изчертан прозорец се уголемява определен участък от нея.

Параметри на екрана

- Избира се функцията DISPLAY OPTIONS от меню CONTROL;
- От появилото се меню се избира подходящ цвят за изобразяване на определени категории обекти, включват се или се изключват обектите за визуализиране, задава се помощна мрежа и др. ;
- След промяната на определени параметри, с <CANCEL> се излиза от подменюто и обектите върху екрана сменят цветовете си според направения избор.

Запитване за обект

Тази функция дава пълна информация за даден обект: тип, име, координати, слой и т.н.

- Избира се функцията QUERY от меню CONTROL;
- С мишката се посочва обект;
- На екрана се визуализира информацията за него;
- Операцията се преустановява с <CANCEL>.

5. Функции за работа с матрици

Дефиниране на матрици за автоматично разполагане

От меню MATRIX се избира съответно функцията:

- ADD DEFAULT MATRIX – матрица по подразбиране. Задава се броят на линиите в *x* и *y* направление.
- ADD STEP MATRIX – стъпкова матрица. Задава се местоположението на първата вертикална линия, стъпката в това направление, броят на линиите или местоположението на крайната линия. Аналогично се процедира и с хоризонталните линии.
- ADD POINT MATRIX – точкова матрица. Последователно се задават всички вертикални линии и след <CANCEL> – всички хоризонтални.

Редактиране на матриците

От меню MATRIX се избира функцията:

- ADD MATRIX LINE – добавяне на линия към матрицата. Типът на линията (вертикална или хоризонтална) се определя като се избере вече съществуваща линия. Мястото на новата линия се посочва с мишката;
- REMOVE MATRIX LINE – изтриване на линия от матрицата. Избира се линия от матрицата и операцията се потвърждава с <CONFIRM>.
- BLOCK / UNBLOCK POINT – блокиране / деблокиране на точка от матрицата. Избира се точка от матрицата, която ще бъде забранена за разполагане на компонент. С повторен избор същата точка се разрешава.
- CHANGE STEP MATRIX – промяна на стъпкова матрица. Избира се линия от матрицата и се задава новия брой на линиите от този тип.
- STRECH STEP MATRIX – свиване / разтягане на стъпкова матрица. Избира се матрицата и се променят размерите ѝ;

Избор на текуща матрица

Тази функция служи за избор на матрица, която ще бъде използвана при следващата операция за автоматично разполагане.

- От меню MATRIX се избира функцията SELECT MATRIX;
- С мишката се посочва матрицата;
- Операцията се завършва с <CONFIRM>.

6. Функции за автоматизирани операции

Автоматизирано разполагане

Операцията позволява да се разположат автоматично компонентите върху платката за да се осигури оптимално трасиране на връзките.

- Избира се функцията AUTOPLACE от меню AUTO;
- Избира се методът за автоматично разполагане:
  - ◊ CHOOSE – компонентите се разполагат по възлите на предварително дефинирана матрица. За да се осъществи разполагане по този метод се изисква задаване на параметри като ориентация, възможност за завъртане, огледално разполагане и др.
  - ◊ POINT PLACE – подобрява се разположението на компоненти с двуредни копуси по възлите на матрицата.
- Операцията се стартира с щракване върху екранния клавиш START PROCESSING и с <CONFIRM> се потвърждава изпълнението ѝ.

Оптимизация на връзките

Тази операция се използва, за да се минимизира общата дължина на връзките.

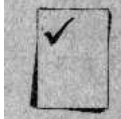
- Избира се функцията RECONNECT от меню AUTO;
- Изпълнението се потвърждава с <CONFIRM>;

Ориентирана оптимизация на връзките

Тази операция се използва, за оптимално свързване и разполагане на

връзките с определен код (най-често земя и захранване) по отношение на корпусите на интегралните схеми.

- Избира се функцията **BIASED RECONNECT** от меню **AUTO**;
- Появява се съобщение **AXIAL WIDTH>**. От клавиатурата се въвеждат кодовете на връзките (напр. **6 7<RETURN>**), които трябва да се ориентират аксиално на корпусите на интегралните схеми.



За да се наблюдава резултатът от операцията е необходимо за връзките с конкретните кодове да се избере различен цвят за визуализация от подменюто **DISPLAY OPTIONS**.

Размяна на еквивалентни логически елементи и изводи

- Избира се функцията **GATE AND PIN SWAP** от меню **AUTO**;
- От подменюто се избира **GATE SWAP** и / или **PIN SWAP**;
- Операцията се стартира с избор на опцията **START PROCESSING** и се извършва на няколко паса като всеки от тях се потвърждава с **<CONFIRM>**.
- За преустановяване на операцията се използва **<CANCEL>**.

Програмата автоматично създава файл с разширение **.RIN**, в който са записани промените. Този файл се използва за отразяване на измененията, направени върху платката обратно в принципната електрическа схема за да има съответствие между платката и схемата.

Автоматично трасиране

- Избира се функцията **AUTOROUTE** от меню **AUTO**;
- В подменюто се задават:
  1. Алгоритъм по който ще се извърши трасирането
     - ◊ земя и захранване – **POWER AND GROUND**;
     - ◊ връзки тип памет – **MEMORY**;
     - ◊ ортогонално трасиране – **ORTHOGONAL**;
  2. Два слоя за трасиране – един за направление *x* и втори – за *y*;
  3. Област, в която ще се извърши трасирането – задава се с координатите на долния ляв и дорния десен ъгъл на правоъгълника (**AREA**);

Програмата автоматично избира областта на първия контур на платката. Ако той е дефиниран некоректно функцията **AUTO ROUTE** не работи.

  4. Мрежа, по която ще се трасира – **ROUTING GRID**;
  5. Минимизация на отворите – **VIA MINIMIZATION**;
  6. Възможност за ориентирана оптимизация на връзките по отношение надлъжната ос на корпусите на интегралните схеми – **BIASED RECONNECT**;
  7. Разрешение / забрана за трасиране на писти под интегралните схеми – **DIP PROTECT**;

  8. Възможност за премахване на вече прекарани писти – **REMOVE PRE-ROUTING**;

Указват се и способи за визуализация на операциите и др.

За конкретните алгоритми за трасиране се задава посока на трасиране, брой сегменти, възможност за опроводяване на частично трасирани връзки, начин за свързване на две съседни площадки на интегрална схема и др.

- Операцията се стартира с избор на опцията **START AUTO ROUTING**; Върху екрана се визуализира информация за трасираните връзки;
- С **<CONFIRM>** се продължава.

Проверка за съответствие между схема и платка

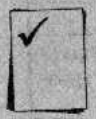
- Избира се функцията **CONNECTION CHECK** от меню **AUTO**;
- В подменюто се задава името на файла с входни данни за платката;
- Операцията се стартира с избор на опцията **START PROCESSING** ;

Върху екрана се визуализира информация за резултатите от проверката и съобщения за открити несъответствия между връзките в платката и тези в принципната електрическа схема (ако има такива);

- С **<CONFIRM>** се продължава.

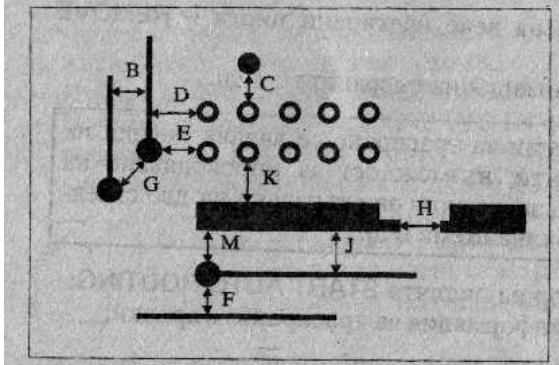
Проверка на технологичните норми

- Избира се функцията **DIMENSION CHECK** от меню **AUTO**;
- От подменюто се задават:
  1. Номерата на слоевете, върху които ще се извършва проверката;
  2. Обектите, които ще се контролират:
     - ◊ Разстояние между контактни площадки (**CHECK PAD TO PAD**);
     - ◊ Разстояние между контактни площадки, писти, преходни отвори, текст (**CHECK PADS, TRACKS, VIAS, TEXT AND TEARDROPS**);
     - ◊ Всички обекти (**CHECK ALL**)
  3. Област, в която ще се извършва проверката:
     - ◊ в прозорец (**CHECKING IN WINDOW**)
     - ◊ върху целия слой (**CHECKING BY LAYER**).
- Операцията се стартира с избор на опцията **START PROCESSING** ;



Върху екрана се визуализират резултатите от проверката, като отделните типове грешки се кодират с буква (в конкретното място, където са установени).

По-често срещаните грешки са илюстрирани на следната фигура:



Нарушени минимални разстояния:

- B Писта – писта
- C Контактна площадка – контактна площадка
- D Писта – контактна площадка
- E Контактна площадка – проходен отвор
- F Писта – проходен отвор
- J Писта – медна област
- H Медна област – медна област
- K Контактна площадка – медна област
- M Проходен отвор – медна област

7. Команди задавани от клавиатурата

В този раздел са изброени командите, които не са изведени на менюта. Те се задават от клавиатурата и могат да бъдат изпълнявани след като е стартирана друга команда. Например докато един компонент се мести с **MOVE** може да се зададе оптимизация на връзките с **C** и **D**.

Команда	Описание
A	Разрешава прекарването на линия (сегмент) под определен ъгъл:
	A0 – под произволен ъгъл; **A45** – под 45 градуса; **A90** –под 90 градуса
C	Включва визуализирането на връзките към преместван компонент.
D	Динамично минимизира дължината на сегментите, свързани към преместван компонент (**C** трябва да е вече активирано).
G <стойност>	Задава стъпката на мрежата
L <слой>	Сменя слоя за трасиране на сегмента или слоя върху които е разположен избрания обект.
L	Превключва между двойката избрани слоеве за трасиране.
M	Огледален образ на компонент или текст.
N	Изтъняване на избрания сегмент от писта.
P <код>	Сменя кода на контактната площадка при създаване на компонент
P	Включва визуализиране на контактните площадки върху движещ се компонент.
R	Включва визуализация на пистите, завършващи върху движещ се компонент (**C** трябва да е активирано предварително).
S <код>	Променя кода на текста за избран текстов стринг
U	Създава частично трасирана писта. Завършва пистата в текущата позиция на курсора, оставяйки нетрасирана останалата част на връзката.
W <код>	Променя кода за ширина на линията.

ЛИТЕРАТУРА

1. Ayres, R. F., *VLSI Silicon Commpilation and the Art of Automatic Microchip Design*, Prentice-Hall, 1983
2. Chirlian, P., *Analysis and Design of Integrated Electronic Circuits*, Harper&Row, 1981
3. Gajski, D. D., *Silicon Compilation*, Addison-Wesley, 1988
4. Gajski, D. D., N. D. Dutt, A. C-H Wu и S. Y-L Lin, *High-Level Synthesis. Introduction to Chip and System Design*, Kluwer Academic, 1992
5. Glasser, L. A., и D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley, 1985
6. Heinbuch, D.V., *CMOS3 Cell Library*, Addison-Wesley, 1988
7. Mead, C. A. и L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980
8. Mazor, S. и P.Langstraat, *A Guide to VHDL*, Kluwer Academic, 1992
9. Sapio, S. и R. J. Smith, *Handbook of Design Automation*, Prentice-Hall, 1984
10. Weste, N. H. E. и K.Eshraghian, *Principles of CMOS VLSI Design. A System Perspective*, Addison-Wesley, 1993
11. Uyemura, J. P., *Circuit Design for CMOS VLSI*, Kluwer Academic, 1993
12. Василева Т. К., и Н. Т. Тюлиев, *Проектиране на печатни платки с персонални компютри*, С., Техника, 1992
13. Ватанабе М., К.Асада, К.Кани и Т.Оцуки, *Проектирование СБИС*, М.,Мир, 1988
14. Киносита К., К.Асада и О.Карацу, *Логическое проектирование СБИС*, М., Мир, 1988
15. Лунд П., *Прецизионные печатные платы*, М., Енергоатомиздат, 1983
16. Мурога С., *Системное проектирование СБИС*, 1 и 2 том, М., Мир, 1985
17. Ульман Д. Д., *Вычислительные аспекты СБИС*, М., Радио и связь, 1990