Презентационные материалы к курсу лекций «ПРИКЛАДНАЯ ИНФОРМАТИКА»

Рындин Е.А.

Ивченко В.Г. Применение языка VHDL при проектировании специализированных СБИС: Учебное пособие. Таганрог: Изд-во ТРТУ, 1999. –80 с.

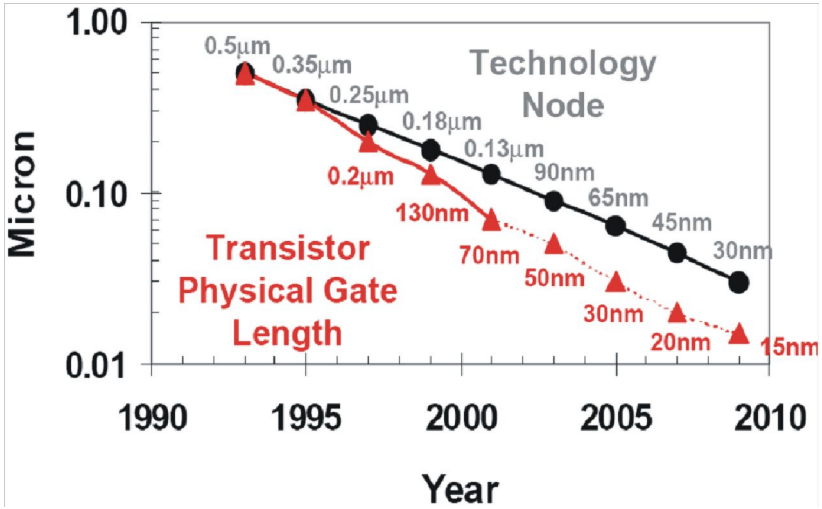
Интегральная схема (микросхема) – законченное микроэлектронное изделие, выполняющее определенные функции обработки информации (сигналов), все интегральные элементы которого (транзисторы, диоды, резисторы и др.) и соединения между ними выполняются в едином технологическом цикле.

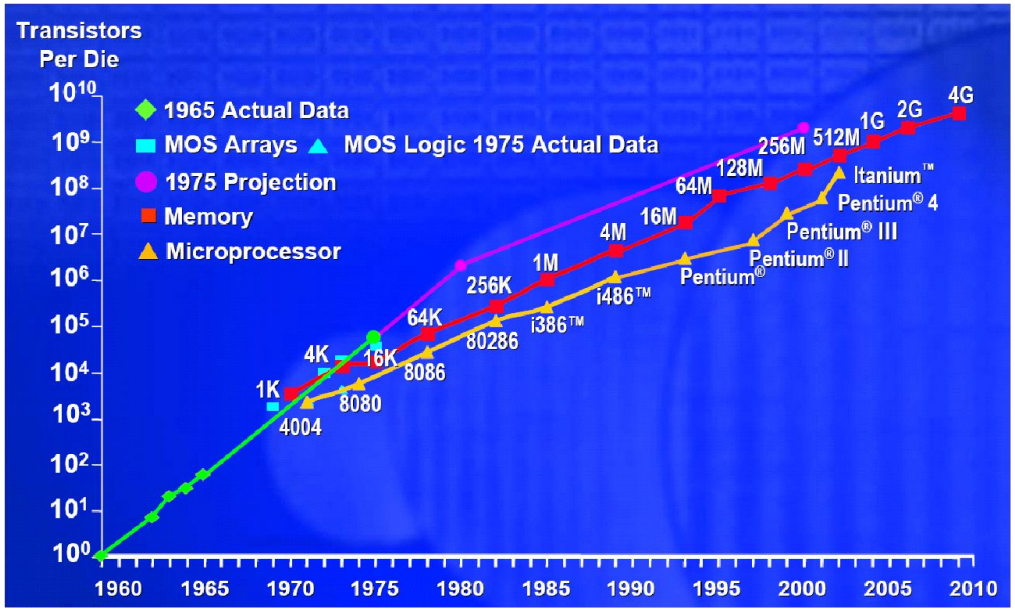
Степень интеграции: $I = \lg(N)$,
где N – число транзисторов на кристалле

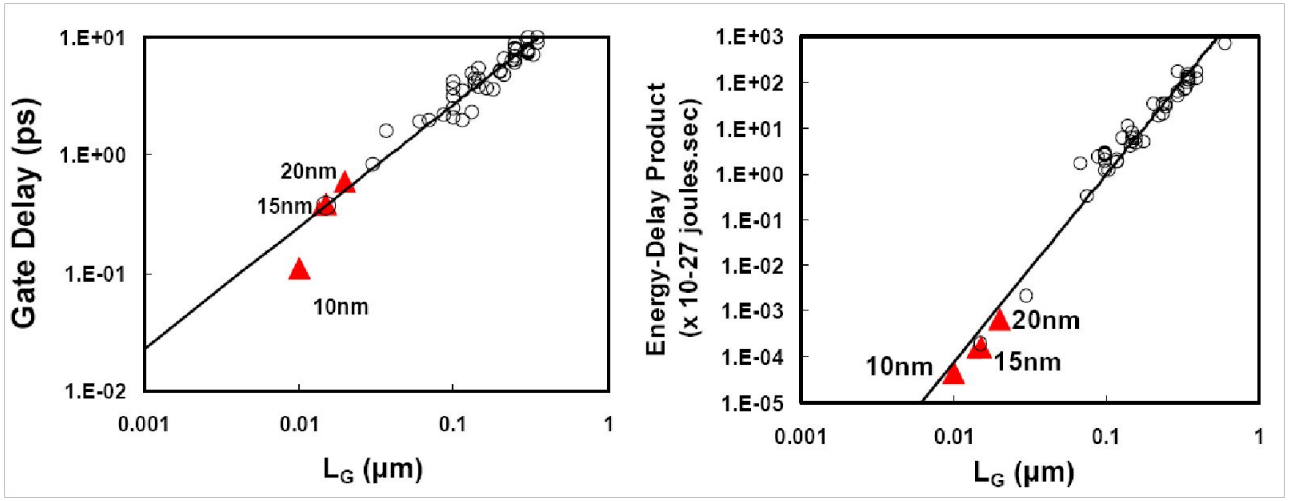
Способы реализации специализированных сверхбольших интегральных схем (СБИС):

- ***полностью заказные***, предполагающие полный цикл проектирования на уровне транзисторных структур;
- ***заказные на основе библиотечных элементов***, предполагающие полный цикл проектирования на уровне функциональных элементов (логические вентили, дешифраторы, счетчики, регистры, АЛУ, процессоры и др.);
- ***полузаказные на основе базовых матричных кристаллов (БМК)***, предполагающие использование кристаллов-полуфабрикатов, функциональная специализация которых определяется на этапе проектирования металлизированных соединений;
- ***на основе программируемых логических интегральных схем (ПЛИС)***, предполагающие использование готовых кристаллов в корпусах, функциональная специализация которых определяется посредством программирования.

Темпы повышения степени интеграции и быстродействия СБИС







В «Концепции развития в Российской Федерации работ в области нанотехнологий на период до 2010 года», одобренной Правительством Российской Федерации (18 ноября 2004 г.), используются следующие ***термины***:

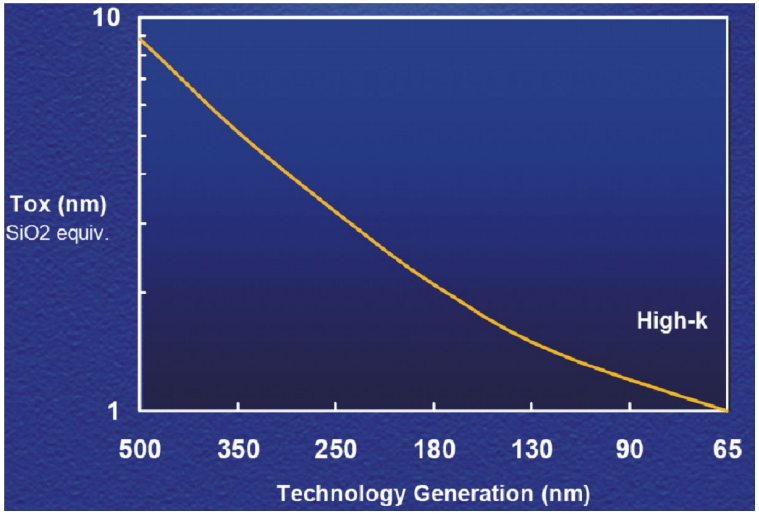
нанотехнология – совокупность методов и приемов, обеспечивающих возможность контролируемым образом создавать и модифицировать объекты, включающие компоненты с размерами менее 100 нм, хотя бы в одном измерении, и в результате этого получившие принципиально новые качества, позволяющие осуществлять их интеграцию в полноценно функционирующие системы большего масштаба. В более широком смысле этот термин охватывает также методы диагностики, характерологии и исследований таких объектов;

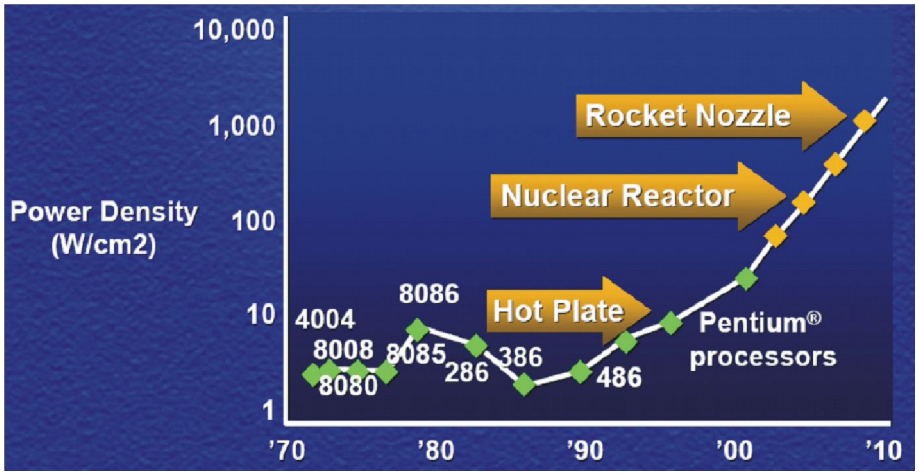
наноматериал – материал, содержащий структурные элементы, геометрические размеры которых, хотя бы в одном измерении, не превышают 100 нм, и, благодаря этому, обладающий качественно новыми свойствами, в том числе заданными функциональными и эксплуатационными характеристиками;

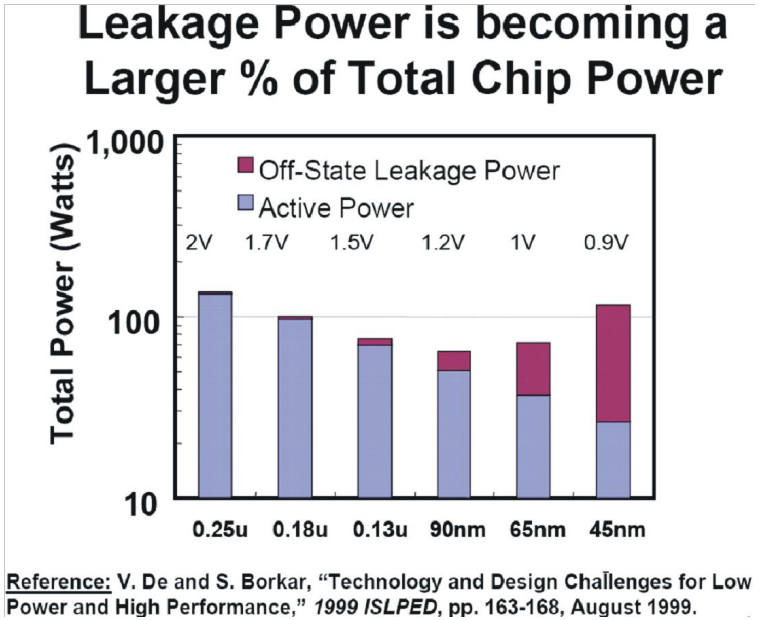
наносистемная техника – созданные полностью или частично на основе наноматериалов и нанотехнологий функционально законченные системы и устройства, характеристики которых кардинальным образом отличаются от показателей систем и устройств аналогичного назначения, созданных по традиционным технологиям;

наноиндустрия – вид деятельности по созданию продукции на основе нанотехнологий, наноматериалов и наносистемной техники.

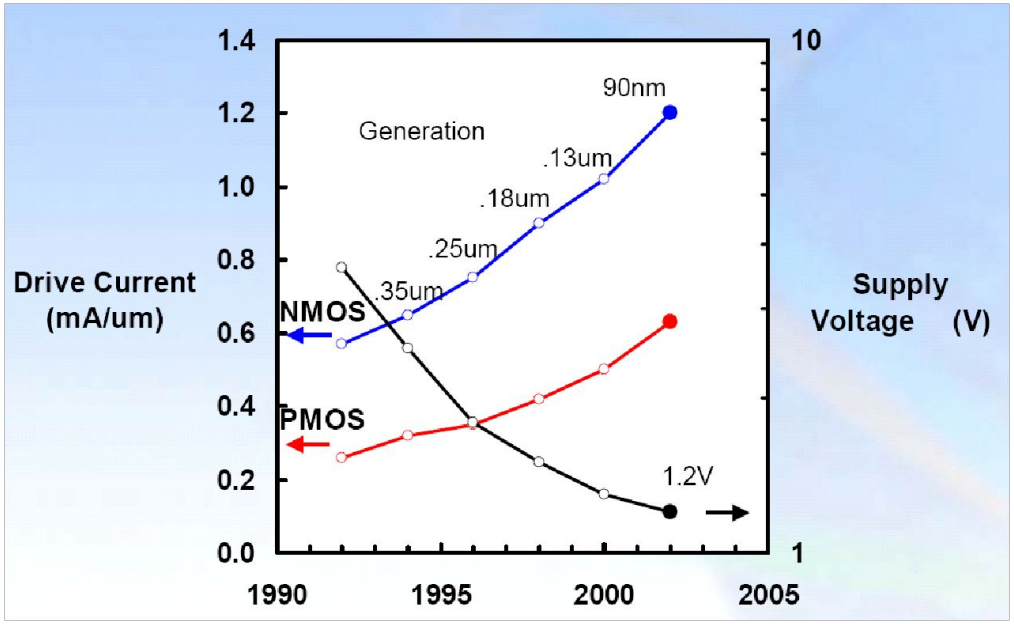
Проблемы и перспективы повышения степени интеграции и быстродействия СБИС



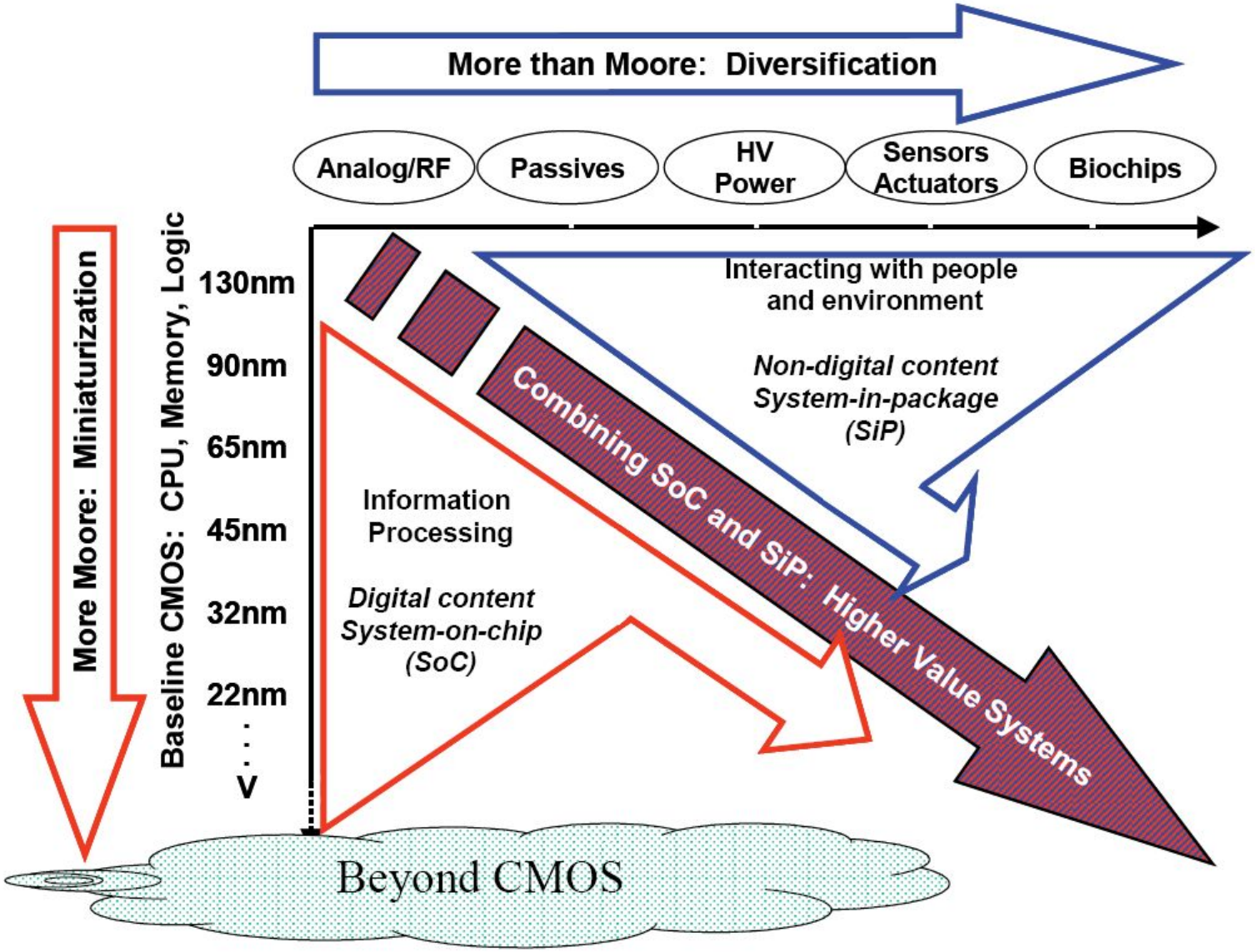




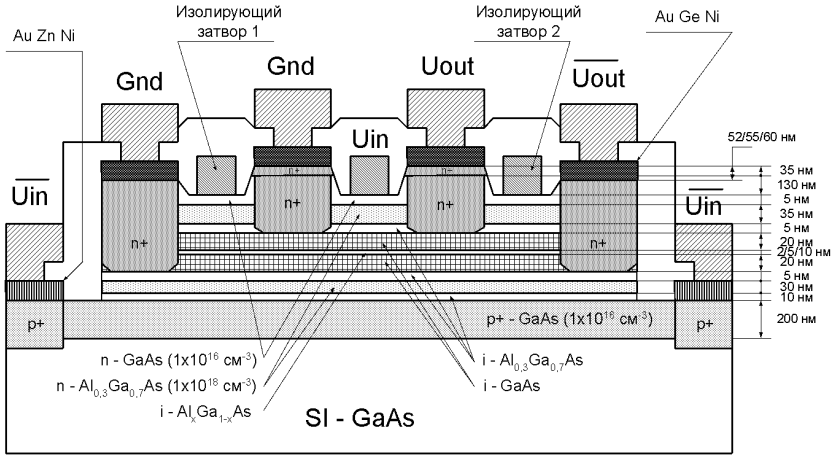
Reference: V. De and S. Borkar, "Technology and Design Challenges for Low Power and High Performance," *1999 ISLPED*, pp. 163-168, August 1999.

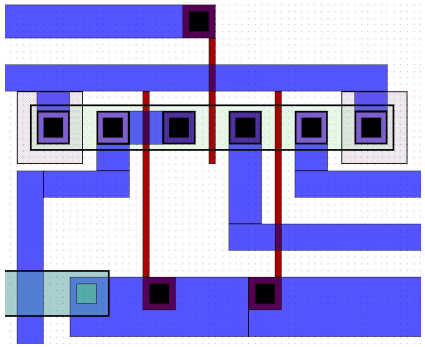


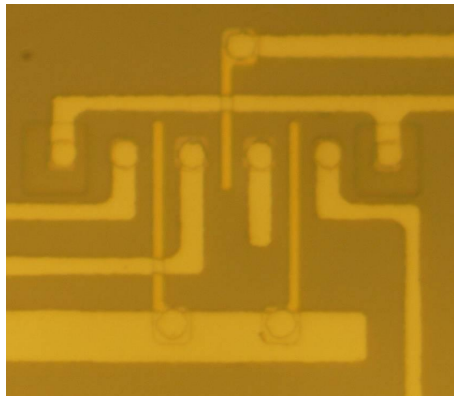
Перспективы повышения степени интеграции и быстродействия СБИС и систем-на-кристалле



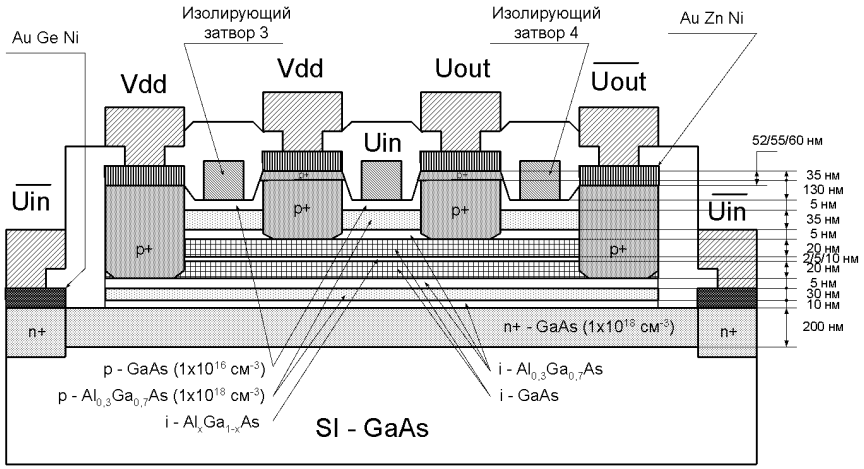
Экспериментальные образцы интегральных элементов на основе туннельно-связанных наноструктур с взаимодополняющими типами проводимости







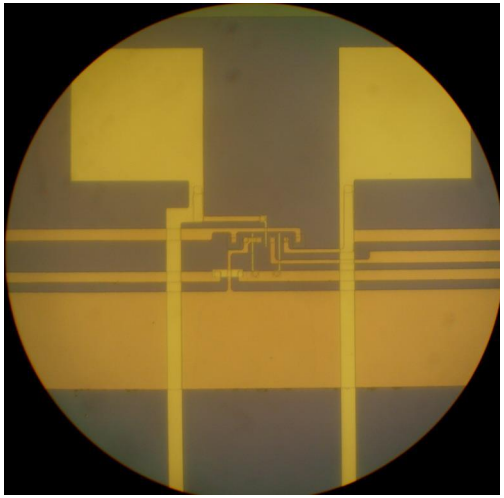
Топология ТСН с электронной проводимостью



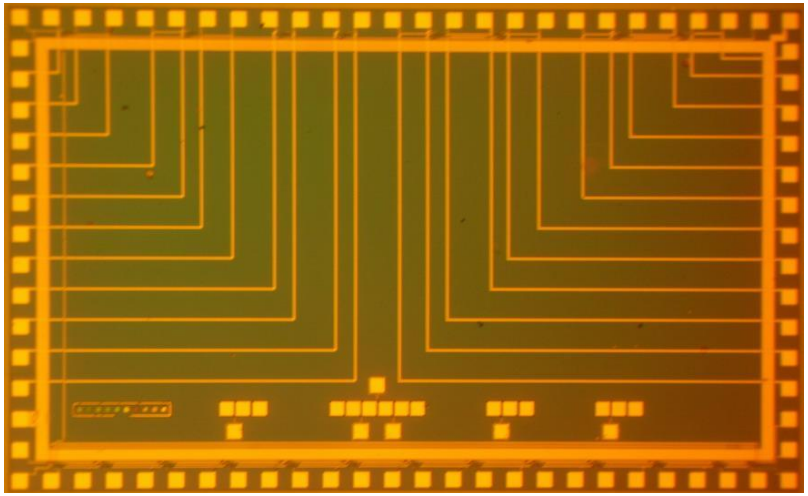
Структуры экспериментальных образцов

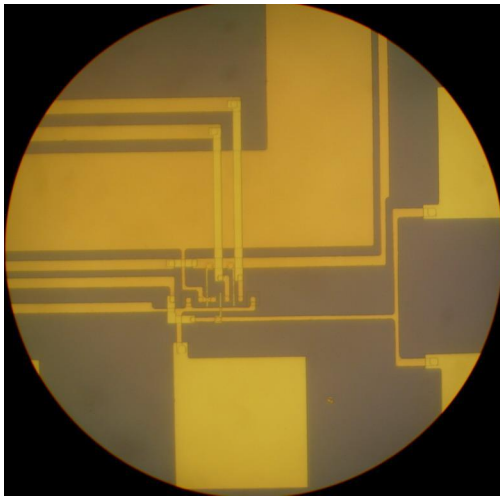
Топология тестовой ТСН

Экспериментальные образцы интегральных элементов на основе туннельно-связанных наноструктур



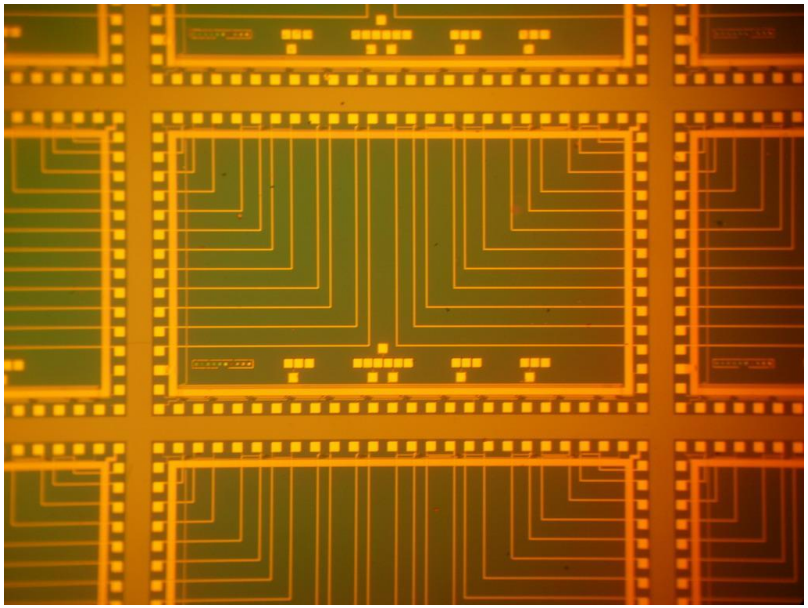
Топология кристалла



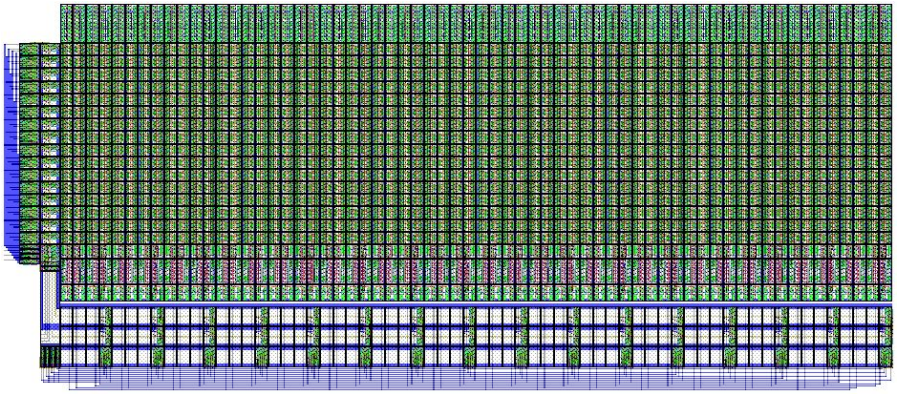


Топология ТСН с электронной проводимостью

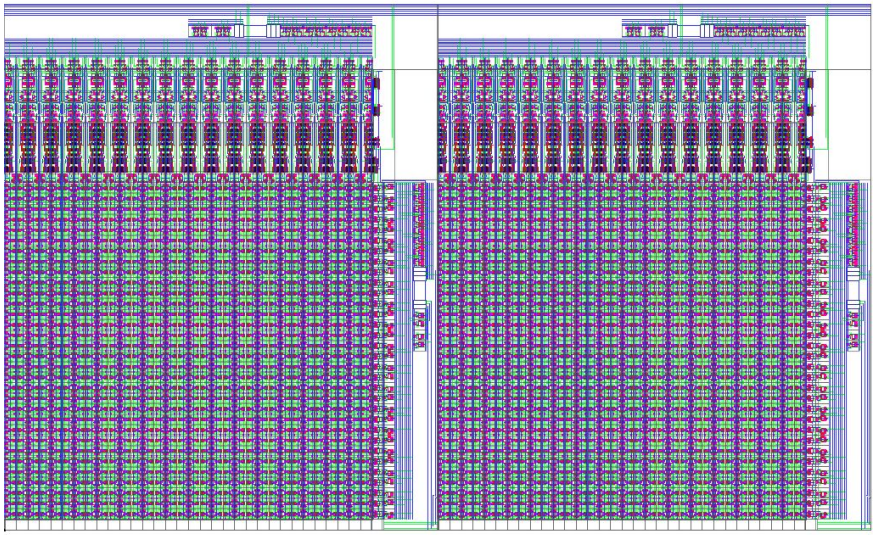
Фрагмент пластины



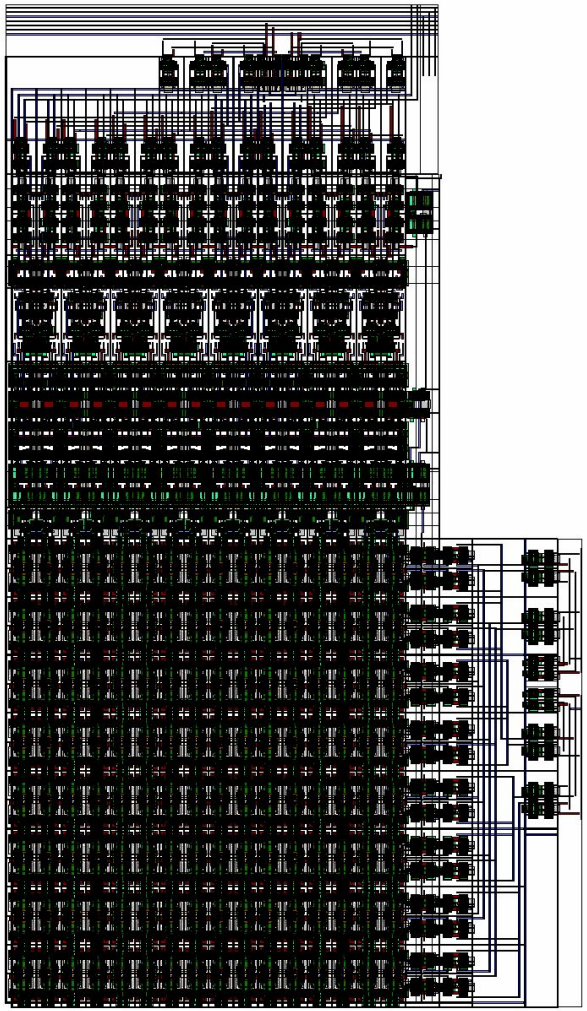
Методика автоматического проектирования блоков СОЗУ на основе библиотек параметризуемых интегральных элементов



Топология секции асинхронного СОЗУ:
организация – 2D, $R_A = 8$, $R_D = 4$, 256x4 бит



Топология секции асинхронного СОЗУ:
организация – 2D, $R_A = 10$, $R_D = 2$, 1024x2 бит



Топология секции асинхронного СОЗУ на основе радиационно-стойкой библиотеки элементов для КНД КМОП-технологии серии БМК «SOI ТЦ»:
организация – 2D, $R_A = 8$, $R_D = 1$, 256x1 бит

Виды обеспечений систем автоматизированного проектирования (САПР):

- ***информационное*** – совокупность информационных ресурсов, необходимых для автоматизированного проектирования;
- ***математическое*** – математические модели, методы и алгоритмы, используемые в процессе автоматизированного проектирования;
- ***лингвистическое*** – совокупность языков описания и форматов представления данных, используемых в процессе автоматизированного проектирования;
- ***программное*** – программные средства автоматизированного проектирования;
- ***техническое*** – аппаратные средства автоматизированного проектирования;
- ***методическое*** – методики автоматизированного проектирования;
- ***организационное*** – способы организации автоматизированного проектирования сложных устройств.

Язык описания проекта СБИС – формальная запись, предназначенная для описания функции, логической организации и (или) структурной схемы проектируемой СБИС.

VHDL, Verilog

VHDL – **V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage

Модули VHDL-описаний:

- объявление объекта проекта (**entity**) – описание интерфейса объекта проекта (входных/выходных сигналов и параметров проекта);
- архитектурное тело (**architecture body**) – описание функции, логической организации и/или структурной схемы объекта проекта;
- объявление конфигурации (**configuration declaration**);
- объявление пакета (**package**) – описание пользовательских типов данных, сонстант, интерфейсов процедур и функций;
- тело пакета (**package body**) – описание тел процедур и функций.

Библиотеки VHDL (library):

- **рабочая библиотека** (**библиотека проекта**) – совокупность файлов VHDL-описаний, находящихся в каталоге проекта;
- **библиотеки ресурсов** – совокупность файлов VHDL-описаний, находящихся вне каталога проекта, на которые имеются ссылки в данном проекте.

Стили описаний архитектурного тела:

- **поведенческий** – описание функции объекта проекта;
- **потоковый** – описание логической организации объекта проекта;
- **структурный** – описание структурной схемы объекта проекта.

Описание параллельных потоков обработки данных в VHDL

Процесс (**process**) – содержит набор операторов, выполняемых последовательно во времени, и представляющих поведенческое описание объекта проекта или одной из его частей.

Архитектурное тело (**architecture**) VHDL-описания объекта проекта может содержать один или несколько процессов. При этом последовательности операторов различных процессов выполняются параллельно.

Модели данных в VHDL:

- **переменные (variable)** – модель данных, предполагающая возможность многократного изменения значений в пределах, определяемых типом данных, при однократном выполнении процесса;
- **сигналы (signal)** – модель данных, предполагающая возможность однократного изменения значений в пределах, определяемых типом данных, при однократном выполнении процесса;
- **константы (constant)** – модель данных, не предусматривающая возможность изменения их значений;
- **параметры (generic)** – модель данных, предназначенная для описания параметров объекта проекта.

Типы данных в VHDL (type):

- **предопределенные** – не требующие объявления при их использовании в VHDL-описании;
- **непредопределенные** – требующие объявления при их использовании в VHDL-описании:
  - **стандартные** – объявление которых содержится в пакетах стандартных библиотек VHDL;
  - **пользовательские** – объявление которых выполняет пользователь в области объявлений архитектурного тела или в объявлении пакета.

Примеры типов данных в VHDL:

- **предопределенные**:

целочисленные десятичные – **integer, positive (25; -78)**;

двоичные – **bit, bit_vector (‘0’; ‘1’; “1001”; (‘1’,’0’))**;

физические - **time (1 s; 7 ms; 15 us; 5 ns; 3 ps; 2 fs)**;

- **непредопределенные**:

  - **стандартные** – **std_logic, std_logic_vector (‘0’; ‘1’; ‘Z’; ‘U’; ‘X’; ‘H’; ‘L’; ‘W’; ‘-’)**;

  - **пользовательские**.

Примеры пользовательских типов данных в VHDL:

```
type MATR is array(0 to 15) of std_logic_vector(7 downto 0);

type TWO_DIM is array(0 to N, R-1 downto 0) of std_logic;

type THREE_DIM is array(0 to 7, Y-1 downto 0, 1 to D-1) of std_logic_vector(R-1 downto 0);

type DIM_DIM is array(0 to 7, Y-1 downto 0) of MATR;

type OPEN_DIM is array(integer range<>, integer range<>) of std_logic;

type DO_DIM is array(integer range<>) of OPEN_DIM(1 to F, R-1 downto 0);

type FRENDS is (VASYA, PETYA, SERGEY, YULYA, KATYA);
```

Основные операторы VHDL:

Операторы присвоения значений:

сигналам - <= ;

остальным моделям данных (переменным, константам, параметрам) - := ;

Логические операторы:

«НЕ» - not;

«И» - and;

«ИЛИ» - or;

«Исключающее ИЛИ» - xor;

Основные операторы VHDL:

Операторы условных переходов if:

«Если - то» -

```
if <условие> then <операторы;> end if;
```

«Если – то, иначе …» -

```
if <условие> then <операторы;> else <операторы;> end if;
```

«Если – то, иначе если – то, … иначе …» -

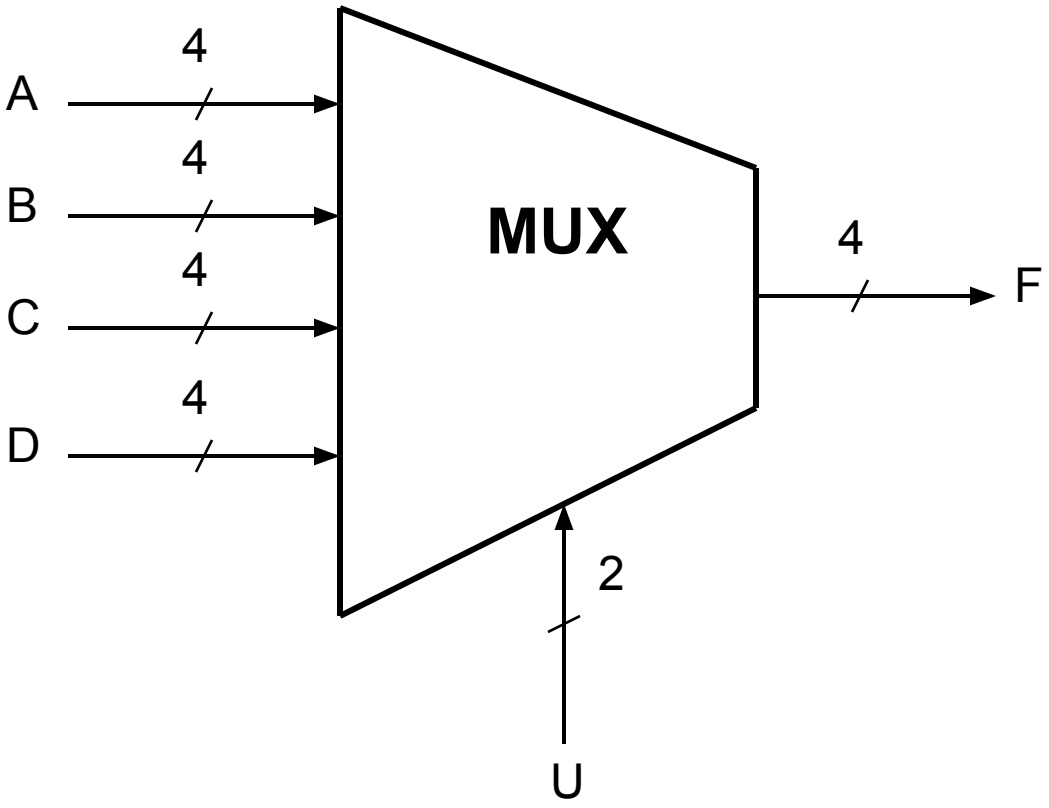
```
if <условие> then <операторы;>
elsif <условие> then <операторы;>
elsif <условие> then <операторы;>
elsif <условие> then <операторы;>
else <операторы;>
end if;
```

Основные операторы VHDL:

Операторы условных переходов case:

```
case <выражение> is
        when <значение 1> => <операторы;>
        when <значение 2> => <операторы;>
        when <значение 3> => <операторы;>
        when <значение 4> => <операторы;>
        when others => <операторы;> или <NULL;>
end case;
```

Условное обозначение мультиплексора



Поведенческое описание мультиплексора с использованием оператора if:

```
library ieee;
use ieee.std_logic_1164.all;

entity MULT1 is
    port(A, B, C, D: in std_logic_vector(3 downto 0);
             F: out std_logic_vector(3 downto 0);
             U: in std_logic_vector(1 downto 0));
end MULT1;

architecture AMULT1 of MULT1 is
begin
    process(A, B, C, D, U)
    begin
        if U = “00” then F <= A;
        elsif U = “01” then F <= B;
        elsif U = “10” then F <= C;
        else F <= D;
        end if;
    end process;
end AMULT1;
```

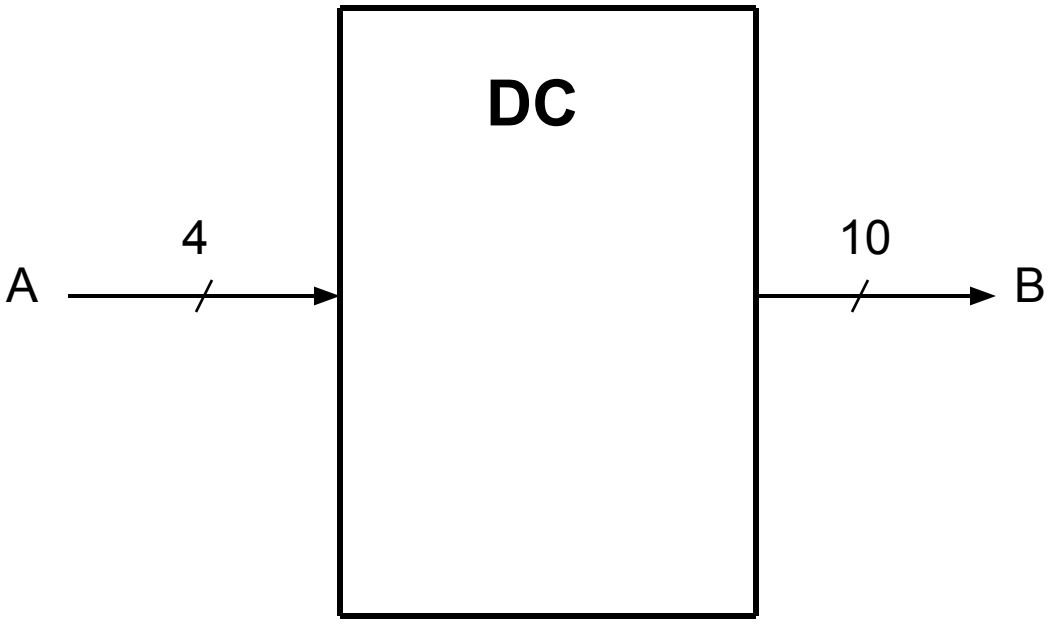
Описание мультиплексора с использованием оператора case:

```
library ieee;
use ieee.std_logic_1164.all;

entity MULT2 is
    generic(R: integer := 4);
    port(A, B, C, D: in std_logic_vector(R-1 downto 0);
             F: out std_logic_vector(R-1 downto 0);
             U: in std_logic_vector(1 downto 0));
end MULT2;

architecture AMULT2 of MULT2 is
begin
    process(A, B, C, D, U)
    begin
        case U is
            when “00” => F <= A;
            when “01” => F <= B;
            when “10” => F <= C;
            when others => F <= D;
        end case;
    end process;
end AMULT2;
```

Описание мультиплексора с использованием оператора case:

```
library ieee;
use ieee.std_logic_1164.all;

entity MULT3 is
    generic(R: integer := 4);
    port(A, B, C, D: in std_logic_vector(R-1 downto 0);
             F: out std_logic_vector(R-1 downto 0);
             U: in std_logic_vector(1 downto 0));
end MULT3;

architecture AMULT3 of MULT3 is
begin
    process(A, B, C, D, U)
    begin
        case U is
            when “00” => F <= A;
            when “01” => F <= B;
            when “10” => F <= C;
            when “11” => F <= D;
            when others => NULL;
        end case;
    end process;
end AMULT3;
```

Условное обозначение дешифратора



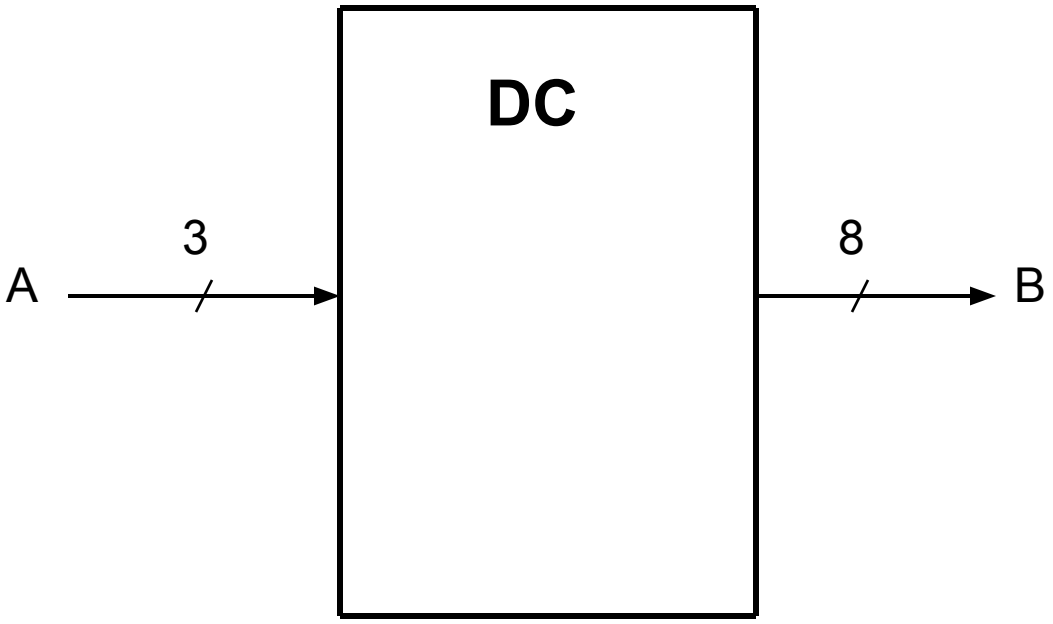
Описание дешифратора с использованием оператора if:

```
library ieee;
use ieee.std_logic_1164.all;

entity DC1 is
    port(A: in std_logic_vector(3 downto 0);
         B: out std_logic_vector(9 downto 0));
end DC1;

architecture ADC1 of DC1 is
begin
    process(A)
    begin
        if    A = "0000" then B <= "0000000001";
        elsif A = "0001" then B <= "0000000010";
        elsif A = "0010" then B <= "0000000100";
        elsif A = "0011" then B <= "0000001000";
        elsif A = "0100" then B <= "0000010000";
        elsif A = "0101" then B <= "0000100000";
        elsif A = "0110" then B <= "0001000000";
        elsif A = "0111" then B <= "0010000000";
        elsif A = "1000" then B <= "0100000000";
        elsif A = "1001" then B <= "1000000000";
        else B <= "0000000000";
        end if;
    end process;
end ADC1;
```

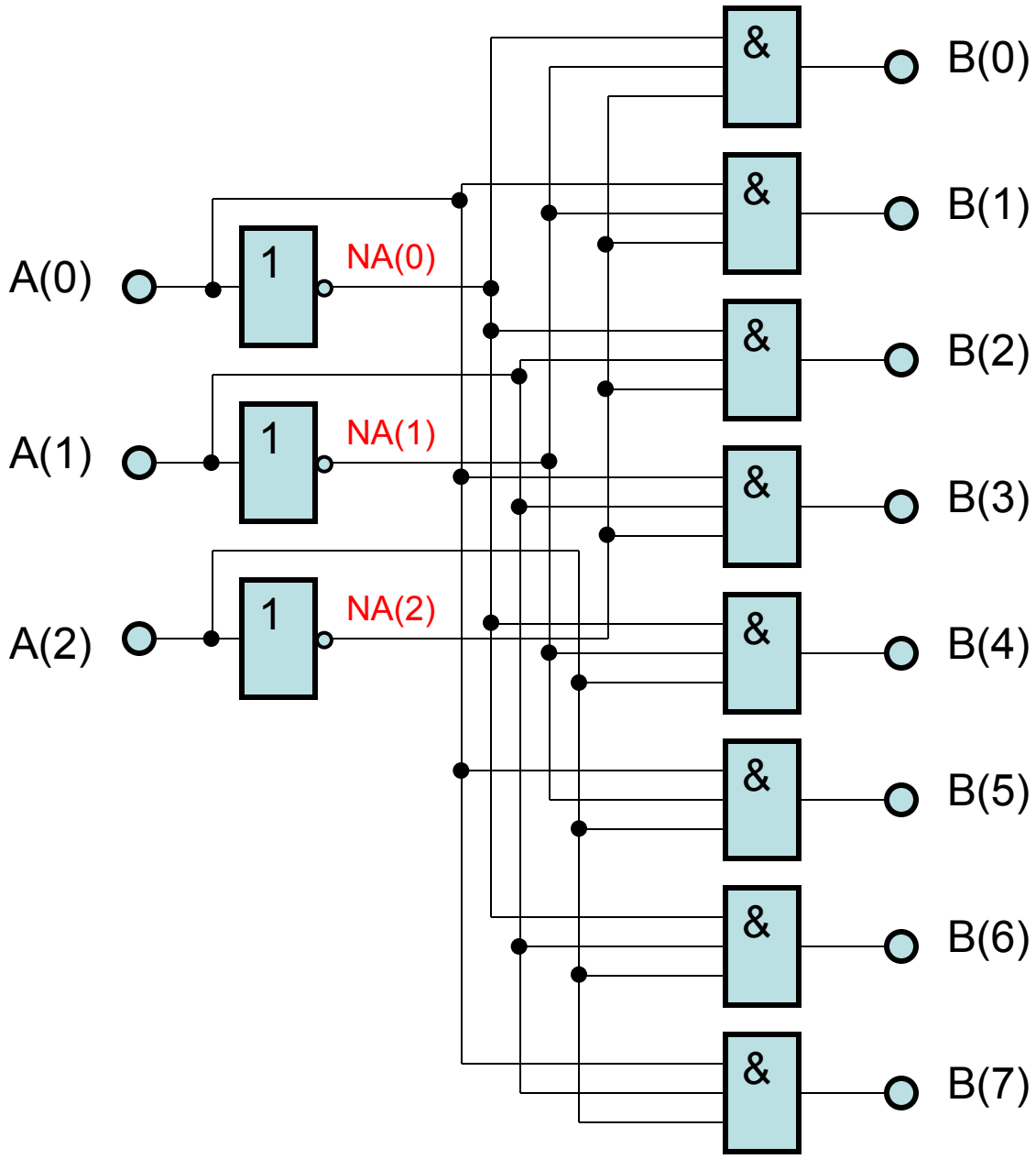
Описание дешифратора с использованием оператора case:

```
library ieee;
use ieee.std_logic_1164.all;

entity DC1 is
    port(A: in std_logic_vector(3 downto 0);
         B: out std_logic_vector(9 downto 0));
end DC1;

architecture ADC1 of DC1 is
begin
    process(A)
    begin
        case A is
            when "0000" => B <= "0000000001";
            when "0001" => B <= "0000000010";
            when "0010" => B <= "0000000100";
            when "0011" => B <= "0000001000";
            when "0100" => B <= "0000010000";
            when "0101" => B <= "0000100000";
            when "0110" => B <= "0001000000";
            when "0111" => B <= "0010000000";
            when "1000" => B <= "0100000000";
            when "1001" => B <= "1000000000";
            when others => B <= "0000000000";
        end if;
    end process;
end ADC1;
```

Условное обозначение дешифратора



Логическая схема дешифратора



Условное обозначение сумматора

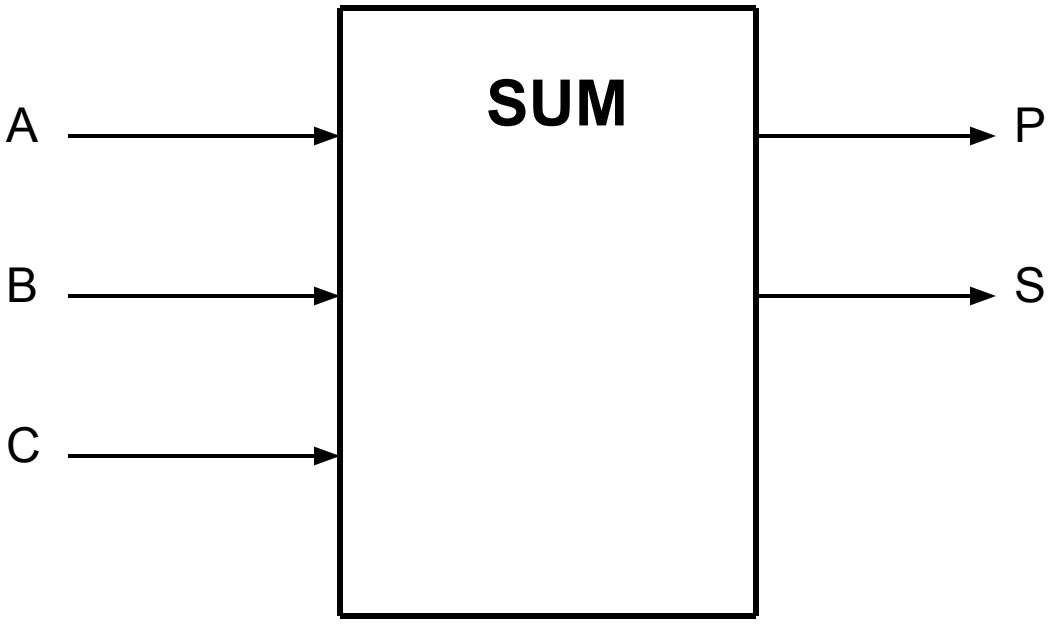
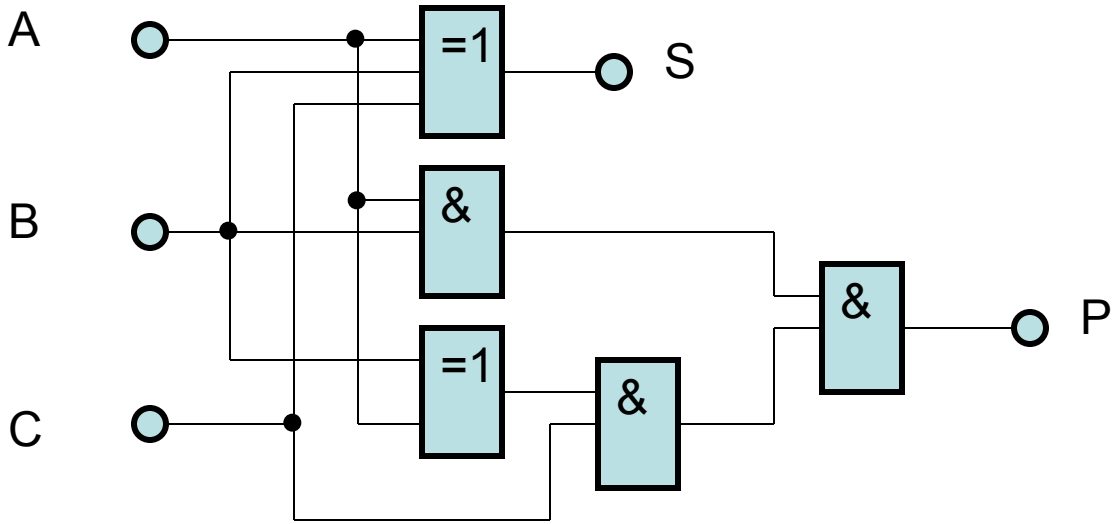


Таблица истинности сумматора

A	B	C	P	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Логическая схема сумматора



S <= A xor B xor C;
P <= (A and B) or (C and (A xor B));

Система автоматизированного проектирования специализированных СБИС

FPGA Advantage Mentor Graphics

Одним из основных отличий языков описания аппаратуры и, в частности, VHDL от традиционных языков программирования (Паскаль, С++ и др.) является возможность ветвления, распараллеливания, конвейеризации потоков данных, без чего, как правило, невозможно проектирование высокопроизводительных СБИС. В результате при проектировании систем высокого уровня сложности возникает необходимость использования наряду с текстовыми VHDL-описаниями графического интерфейса, позволяющего визуализировать параллельные потоки информации и тем самым сократить количество ошибок, время проектирования, значительно облегчить труд разработчика.

Именно таким интерфейсом, а также рядом дополнительных возможностей обладает САПР FPGA Advantage Mentor Graphics.

Общая организация САПР FPGA ADVANTAGE

САПР FPGA Advantage Mentor Graphics предназначена для автоматизированного проектирования СБИС с использованием языков VHDL или Verilog с последующей реализацией на основе ПЛИС и/или в виде заказных микросхем.

В состав САПР FPGA Advantage входят три основных подсистемы:

Renoir;

ModelSim;

Leonardo Spectrum.

Данные подсистемы объединены файловой оболочкой проектов Design Browser, позволяющей создавать, открывать, закрывать, копировать файлы проектов и библиотек, а также осуществлять запуск подсистем Renoir, ModelSim и Leonardo Spectrum для файлов любого уровня иерархии проектов.

Подсистема Renoir

Renoir - подсистема синтеза, генерации и верификации VHDL/Verilog-описаний проекта, содержащая:

- ✔ **текстовый редактор;**
- ✔ **синтаксический анализатор;**
- ✔ **редактор условных графических обозначений (Symbol);**
- ✔ **графический редактор структурных схем (Block Diagram);**
- ✔ **графический редактор конечных автоматов (State Diagram);**
- ✔ **редактор таблиц истинности (Truth Table);**
- ✔ **редактор блок-схем алгоритмов (Flow Chart);**
- ✔ **генератор VHDL/Verilog-описаний проектов на основе их графических представлений.**

Подсистема ModelSim

ModelSim - подсистема функционально-логического моделирования проекта, в том числе с учетом задержек в элементах и соединительных линиях, осуществляемого после размещения и трассировки (back-annotate моделирование).

Подсистема ModelSim обеспечивает возможность задания векторов входных сигналов для моделирования проектов графически, в специализированном текстовом формате, с использованием несинтезируемого подмножества VHDL.

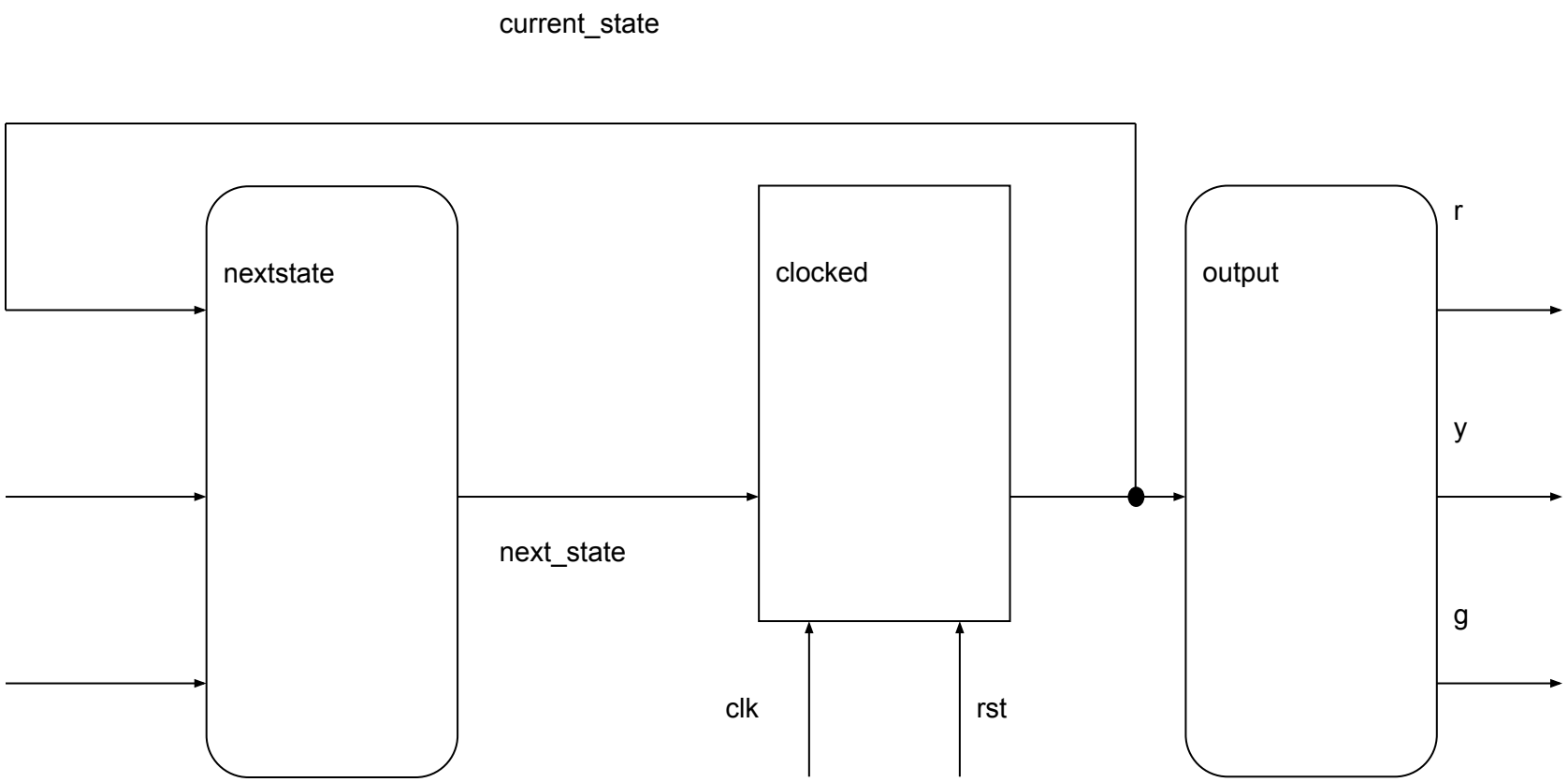
Подсистема Leonardo Spectrum

Leonardo Spectrum - компилятор VHDL/Verilog-описаний проектов СБИС в стандартные форматы описания структуры (файлы формата EDIF) и задержек в элементах и линиях связи (файлы формата SDF) для последующей реализации в виде ПЛИС или заказных микросхем с использованием любых соответствующих САПР (например, Altera MAX+plus II, Tanner Pro, Cadence и т.д.). Причем передача данных о проекте и запуск следующей САПР происходит автоматически по заданным настройкам системы.

Пример проектирования конечного автомата в САПР FPGA Advantage

Конечный автомат (КА, FSM) – это последовательностное устройство, характеризующееся некоторым количество входов, выходов и устойчивых внутренних состояний, способное переключаться в одно из этих состояний в зависимости от комбинации входных сигналов и предыдущего состояния.
Конечный автомат называется КА **Мура**, если выходные сигналы зависят ***только от текущего состояния*** автомата.
Конечный автомат называется КА **Мили**, если выходные сигналы зависят ***как от текущего состояния*** автомата, ***так и от текущей комбинации входных сигналов***.

Пример проектирования конечного автомата в САПР FPGA Advantage



Синтез VHDL-описаний проектов СБИС в подсистеме Renoir

Запуск системы. Оболочка Design Browser

После запуска системы FPGA Advantage на экране монитора появляется оболочка Design Browser, содержащая меню команд, кнопки для быстрого вызова команд, окно Sourse со списком каталогов с файлами проектов, полученными в подсистеме Renoir, окно HDL со списком каталогов с файлами VHDL/Verilog-описаний, окно Downstream с закладками ModelSim (список каталогов с исходными, управляющими и результирующими файлами моделирования проектов) и Leonardo (список каталогов с файлами, полученными в результате компиляции), а также строку подсказки (в нижней части экрана).

Design Browser

File Edit View HDL Animation Tools Options Window Help

Source

- BigBoss - E:\Aproj\BigBoss\src
- XY - C:\Designs\XY\src
- Move - C:\Designs\Move\src
- HLF_BOARD - D:\proj\HLF_BOARD\HLF_BOARD

HDL

- BigBoss - E:\Aproj\BigBoss\hdl
- XY - C:\Designs\XY\hdl
- Move - C:\Designs\Move\hdl
- HLF_BOARD - D:\proj\HLF_BOARD\HLF_BOARD_hdl

Downstream

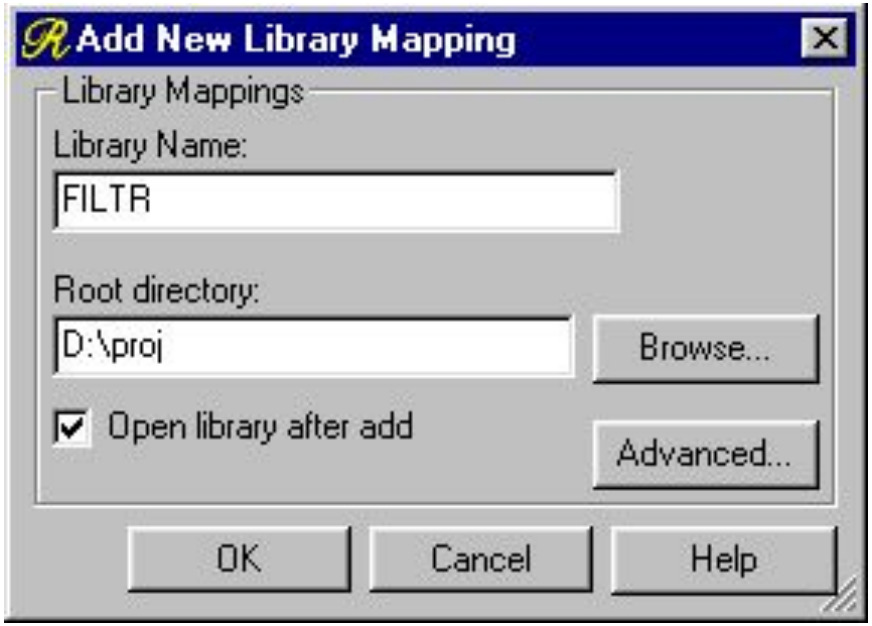
- BigBoss - E:\Aproj\BigBoss\work
- XY - C:\Designs\XY\sim
- Move - C:\Designs\Move\sim
- HLF_BOARD - D:\proj\HLF_BOARD\HLF_BOARD_sim

ModelSim | Leonardo

Ready

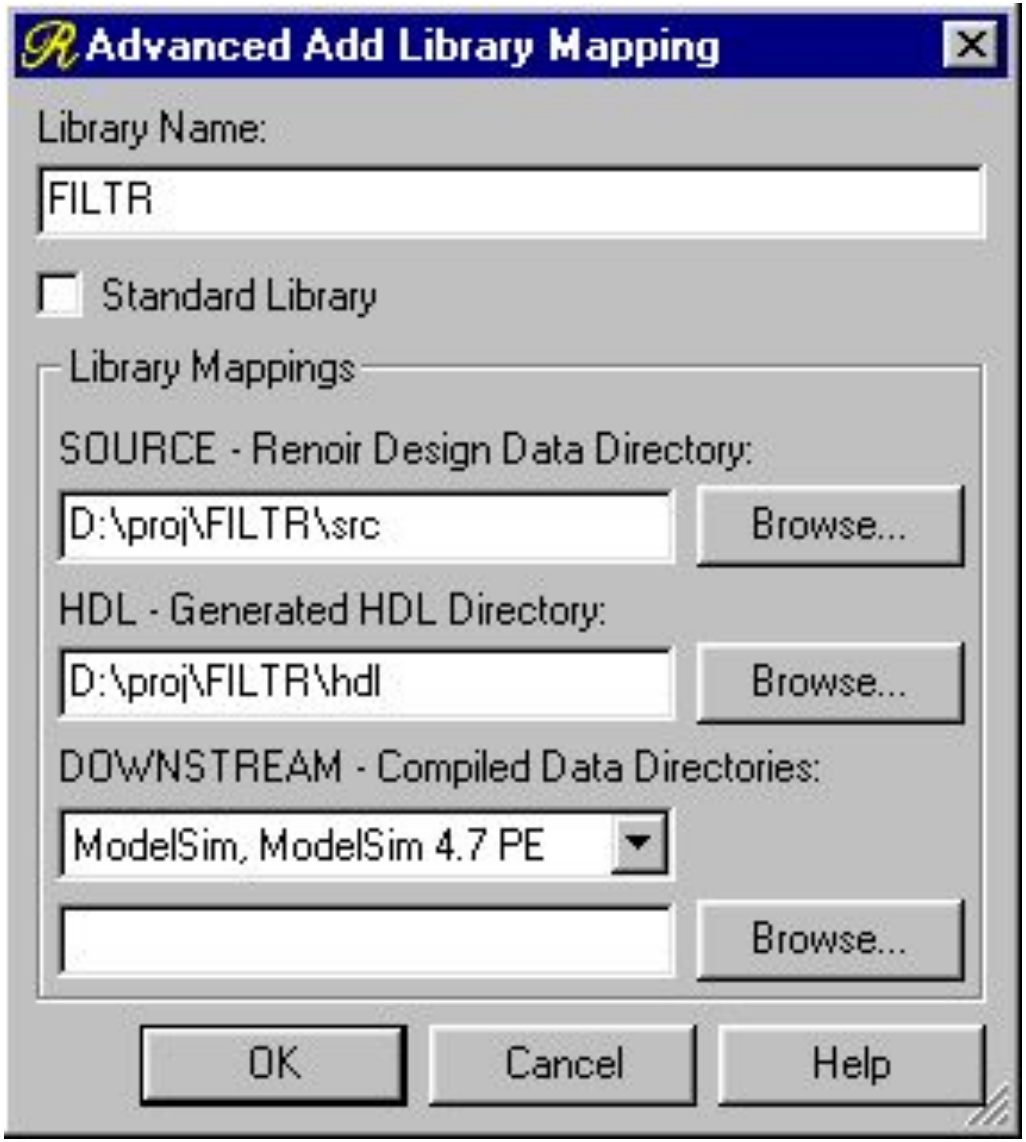
Создание библиотеки проекта

Прежде всего необходимо создать библиотеку проекта с помощью команды File/New Library главного меню. В появившемся диалоговом окне Add New Library Mapping необходимо ввести имя библиотеки проекта в поле Library Name (в приведенном примере - FILTR), корневой каталог библиотеки проекта в поле Root Directory (в приведенном примере - D:\proj), установить флажок Open Library after Add (если он не установлен) и нажать кнопку «Advanced» с помощью манипулятора «мышь».



Перед запуском FPGA Advantage необходимо установить в свойстве операционной системы Windows «Язык и стандарты» региональный стандарт «Английский (США)» (или убедиться в его установке).

После активизации кнопки «Advanced» на экране появится окно Advanced Add Library Mapping, в котором будут автоматически сформированы имя библиотеки в поле Library Name (FILTR), имя каталога с файлами проекта, полученными в подсистеме Renoir, в поле SOURCE - Renoir Design Data Directory (D:\proj\FILTR\src), имя каталога с VHDL/Verilog-файлами проекта в поле HDL - Generated HDL Directory (D:\proj\FILTR\hdl). Необходимо с помощью манипулятора «мышь» скопировать путь к корневому каталогу библиотеки проекта (например, из поля SOURCE - Renoir Design Data Directory без последнего подкаталога \src - D:\proj\FILTR) и вставить его в пустое окошко поля DOWNSTREAM - Compiled Data Directories под заголовком ModelSim, добавив в конце подкаталог \sim, после чего выполнить аналогичные действия, активизировав опускающееся меню справа от заголовка ModelSim, выбрав заголовок Leonardo и добавив в конце подкаталог \leo. После этого нажать кнопку «OK».


Advanced Add Library Mapping
Library Name:
FILTR
Standard Library
Library Mappings
SOURCE - Renoir Design Data Directory:
D:\proj\FILTR\src
Browse...
HDL - Generated HDL Directory:
D:\proj\FILTR\hdl
Browse...
DOWNSTREAM - Compiled Data Directories:
ModelSim, ModelSim 4.7 PE
Browse...
OK
Cancel
Help

Открытие библиотеки проекта

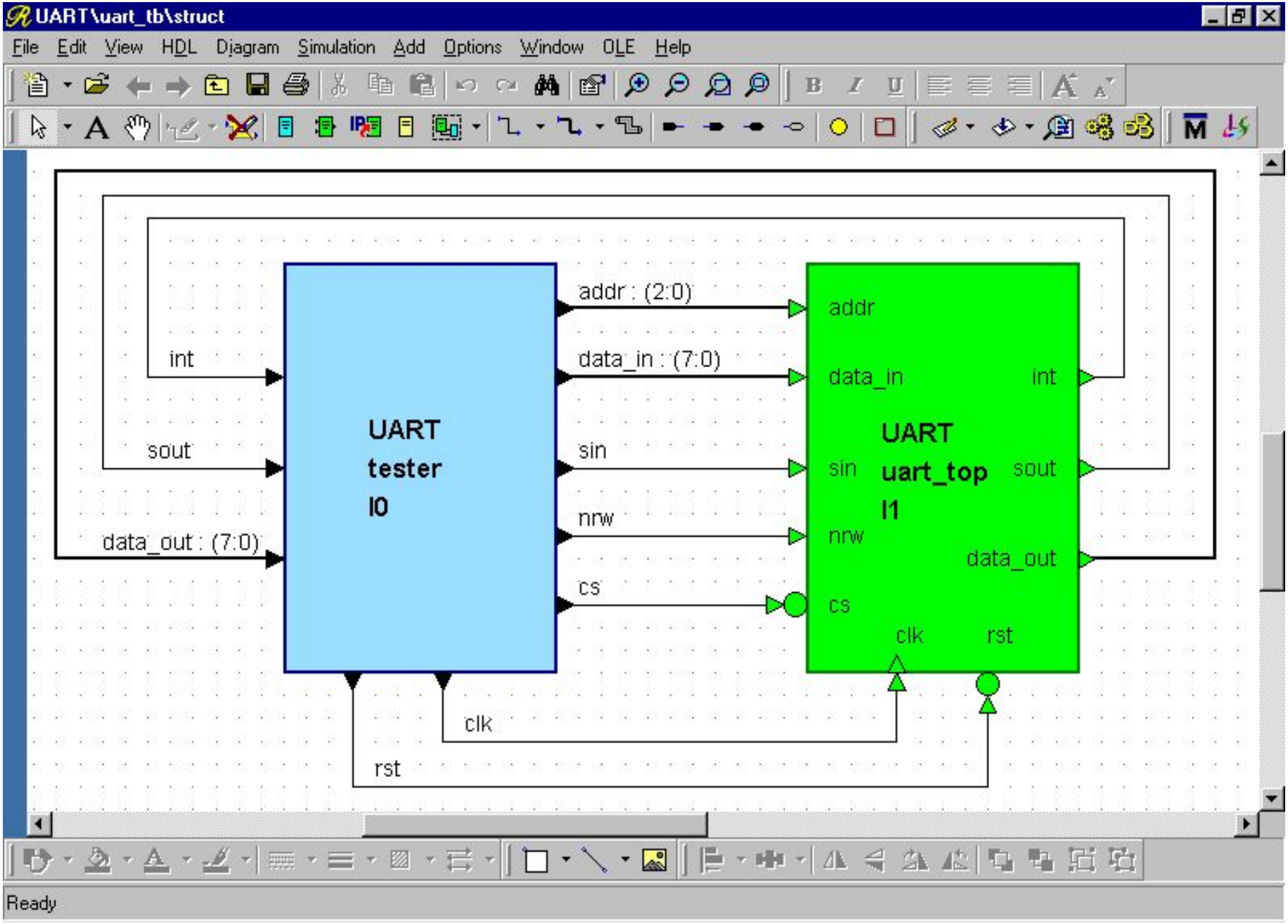
Для того, чтобы открыть вновь созданную (или ранее созданную) библиотеку проекта, необходимо воспользоваться командой основного меню File/Open Library или активизировать кнопку

В появившемся окне Open Library необходимо выделить имя открываемой библиотеки (в данном примере - FILTR), после чего нажать на кнопку «Open». При этом имена соответствующих каталогов библиотеки проекта появятся в окнах Sourse (FILTR - D:\proj\FILTR\src), HDL (FILTR - D:\proj\FILTR\hdl) и Downstream (FILTR - D:\proj\FILTR\sim на закладке ModelSim и FILTR - D:\proj\FILTR\leo на закладке Leonardo) оболочки Design Browser и будут подсвечены синим цветом, свидетельствующем о том, что данные каталоги пока не содержат файлов проекта. При создании файлов проекта в соответствующих каталогах их имена в оболочке Design Browser изменят цвет на черный.

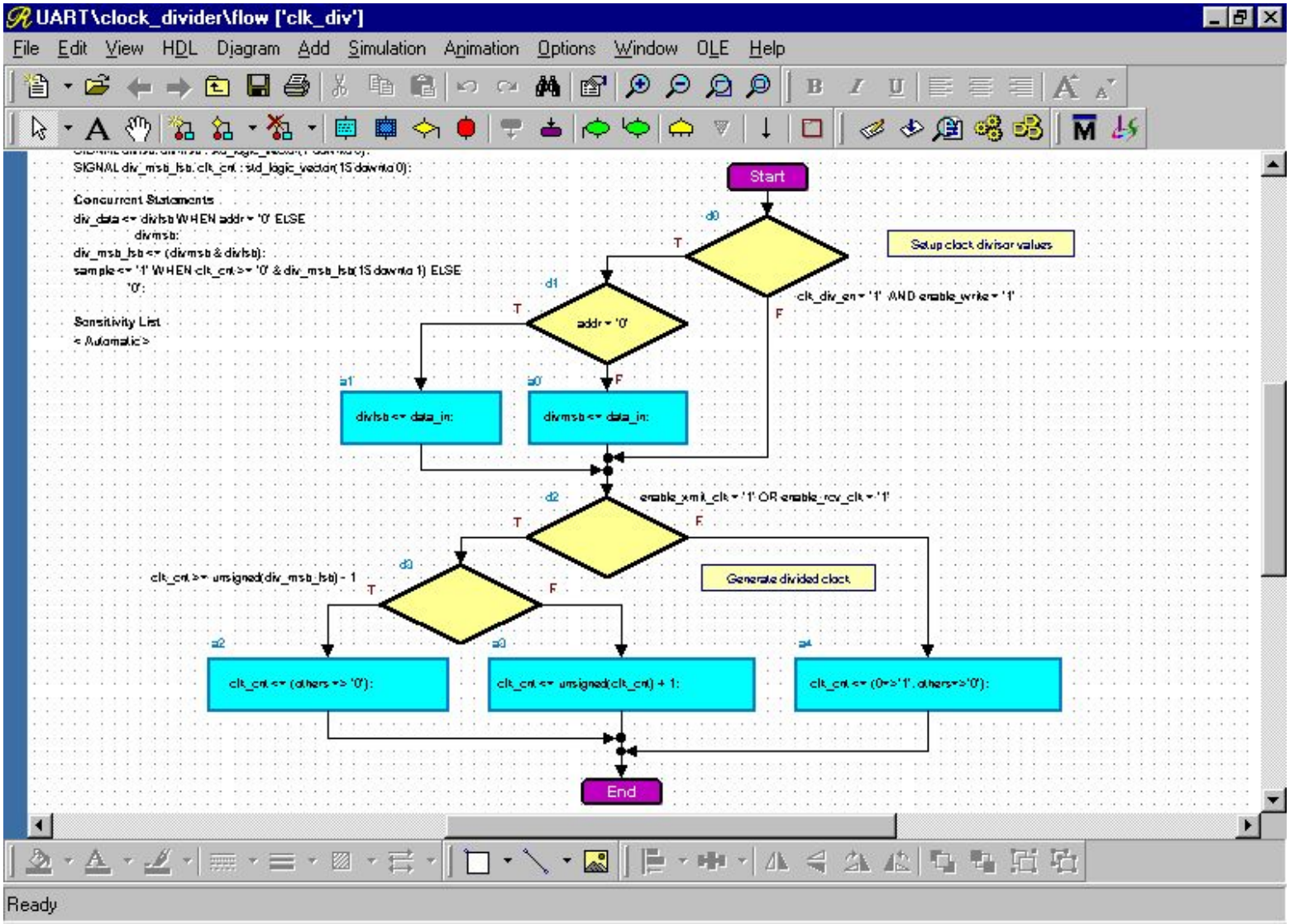
Создание файлов проекта в подсистеме Renoir

После того, как библиотека проекта создана и открыта, можно приступать к созданию файлов проекта в подсистеме Renoir с помощью команды основного меню File/New. В опускающемся меню появится список представлений:

Block Diagram - структурная схема, представление структурных компонентов в виде условных графических обозначений (УГО), соединенных между собой сигнальными линиями и шинами, передающими многоразрядные сигналы, объединенные общим именем с соответствующей индексацией. Каждый структурный компонент может быть описан любым из приведенных в данном списке способов. Сложность структурной схемы и число уровней иерархии ограничивается возможностями аппаратных средств САПР, в основном, объемом оперативной и дисковой памяти;


UART\uart_tb\struct
File Edit View HDL Diagram Simulation Add Options Window OLE Help
int
sout
data_out : (7:0)
UART
tester
I0
addr : (2:0)
data_in : (7:0)
sin
nrw
cs
clk
rst
addr
data_in
int
UART
uart_top
I1
sin
sout
nrw
data_out
cs
clk
rst
Ready

Flow Chart - графическое представление алгоритма функционирования проекта в виде блок-схемы. Каждый из блоков представлен соответствующим VHDL-описанием;



State Diagram - представление конечных автоматов - функциональное представление в виде направленного графа, вершины которого (круги) определяют различные состояния системы (текущие значения всех сигналов и переменных, записанные с помощью VHDL-назначений), а ребра (линии, соединяющие круги) - переходы между состояниями. Переходы могут быть условными или безусловными и сопровождаться выполнением определенных функций обработки данных;

UART\xmit_rcv_control\fsm ['Rcv' 1 of 2]

File Edit View HDL Diagram Add Simulation Animation Options Window OLE Help

sin='0'
enable_rcv_clk <= '1';
rcv_bit_cnt_cld <= "000";

rcv_bit_cnt_cld <= "000";

waiting

check_lock

1

2

sin='1'
enable_rcv_clk <= '0';

enable_rcv_clk <= '1';

sin='0'

rcv_locked

finish_rcv

Mentor Graphics Corporation		Project:	UART Demo
Title:	RCV receive control	Control the receiving sequence of serial input data.	
Path:	UART\xmit_rcv_control\fsm		
Edited:	by JSmith on 20 Jul 2000		

enable_rcv_clk <= '0';
rcving <= '0';
done_rcving <= '1';

Architecture Declarations
SIGNAL xmit_bit_cnt : std_logic_vector(2 downto 0) := '000' ;

Signals Status

SIGNAL	SCOPE	DEFAULT	RESET	STATUS	USE IN
done_rcving	OUT	'0'		COMB	Rcv
done_xmitting	OUT	'0'		COMB	Xmit
enable_rcv_clk	OUT	'0'		COMB	Rcv
enable_xmit_clk	OUT		'0'	CLKD	Xmit
rcv_bit_cnt	OUT		(others->'0')	CLKD	Rcv
rcving	OUT	'0'		COMB	Rcv
read_bit	OUT	'0'		COMB	Rcv

sample='1'

sample='1'

Ready

Эти условия и функции представлены VHDL-описаниями, помещенными в прямоугольники, относящимися к соответствующим ребрам. Очередность проверки условий для различных переходов из одного состояния определяется уровнем приоритета, задаваемым разработчиком и выводимым в виде числа в маленьком круге, расположенном на каждом ребре. При составлении State Diagram допускаются петли, т.е. переходы из текущего состояния в него же, сопровождающиеся определенными действиями при определенных условиях. Состояния могут быть иерархическими, т.е. текущее состояние может быть представлено, в свою очередь, конечным автоматом более низкого уровня иерархии. Работа конечного автомата синхронизируются тактовыми импульсами (например, Clock). При этом важно помнить, что при входе в состояние по фронту сигнала Clock выполняются действия, относящиеся к переходу, а при выходе - действия, относящиеся к состоянию, из которого осуществляется выход;

Symbol - создание условного графического обозначения компонента проекта. При активизации данной команды появляется окно редактора с прямоугольным УГО без внешних выводов (портов). Условные обозначения портов расставляются по периметру УГО с помощью манипулятора «мышь» при активизации следующих команд меню: Add/Input Port - добавление входного порта, Add/Output Port - добавление выходного порта, Add/InOut Port - добавление двунаправленного порта. Фиксация обозначения порта на периметре УГО осуществляется щелчком левой кнопки «мыши». Для ввода параметров порта необходимо дважды щелкнуть левой кнопкой «мыши» на его обозначении и в появившемся диалоговом окне ввести имя порта в поле Name, тип сигнала в поле Type, способ задания диапазона допустимых значений в поле Constraint (Index - диапазон индексов линий шины, Range - диапазон значений сигнала, None - для однобитных сигналов), после чего последовательно нажать кнопки «Apply» и «OK».

Если необходимо ввести параметры данного компонента (generic), необходимо навести курсор «мыши» на УГО, щелкнуть правой кнопкой и активизировать команду контекстного меню Object Properties. В появившемся диалоговом окне в закладке Generic Declarations следует ввести имя, тип и значение параметра в трех полях в нижней части окна под заголовками Name, Type и Value соответственно, после чего нажать кнопку «Add». При этом введенная информация появится в верхней части основного поля диалогового окна. При необходимости изменить введенные данные следует в основном окне выделить с помощью «мыши» строку с требуемым параметром и в нижних полях ввести нужные исправления, после чего нажать кнопку «Modify». Кнопка «Remove» служит для удаления выделенной строки из списка в основном поле. После ввода всех портов и параметров необходимо сохранить УГО с помощью команды File/Save.

Symbol Object Properties

Ports | Declarations | Generic Declarations | Text | Symbol

Name:	Type:	Value:
DB	integer	7

Add

Modify

Remove

DB | integer | 7

OK | Cancel | Apply | Help

Truth Table - представление в виде таблицы истинности;

UART\address_decode\tbl

File Edit View HDL Diagram Options Window Help

	A	B	C	D	E	F	G
1	addr	addr	clk_div_en	xmitdt_en	recvdt_en	status_en	clr_int_en
2	"000"	"000"	'1'				
3	"001"	"001"	'1'				
4	"100"	"100"		'1'			
5	"101"	"101"			'1'		
6	"110"	"110"				'1'	
7	"111"	"111"					'1'
8			'0'	'0'	'0'	'0'	'0'

Ready

Verilog Include, Verilog Module - описания на языке Verilog;

VHDL Architecture/Entity - VHDL-описание, включающее два обязательных модуля проекта: объявление объекта проекта Entity и архитектурное тело Architecture Body. Кроме объявления объекта проекта и архитектурного тела, в VHDL-описании могут использоваться еще три модуля проекта: объявление конфигурации, объявление пакета и тело пакета. В языке VHDL предусмотрен механизм пакетов для часто используемых описаний, констант, типов, сигналов. Эти описания помещаются в объявлении пакета (Package Declaration). Если пользователь осуществляет нестандартные операции или функции, их интерфейсы описываются в объявлении пакета, а тела содержатся в теле пакета (Package Body);

VHDL Package Body - VHDL-описание тела пакета;

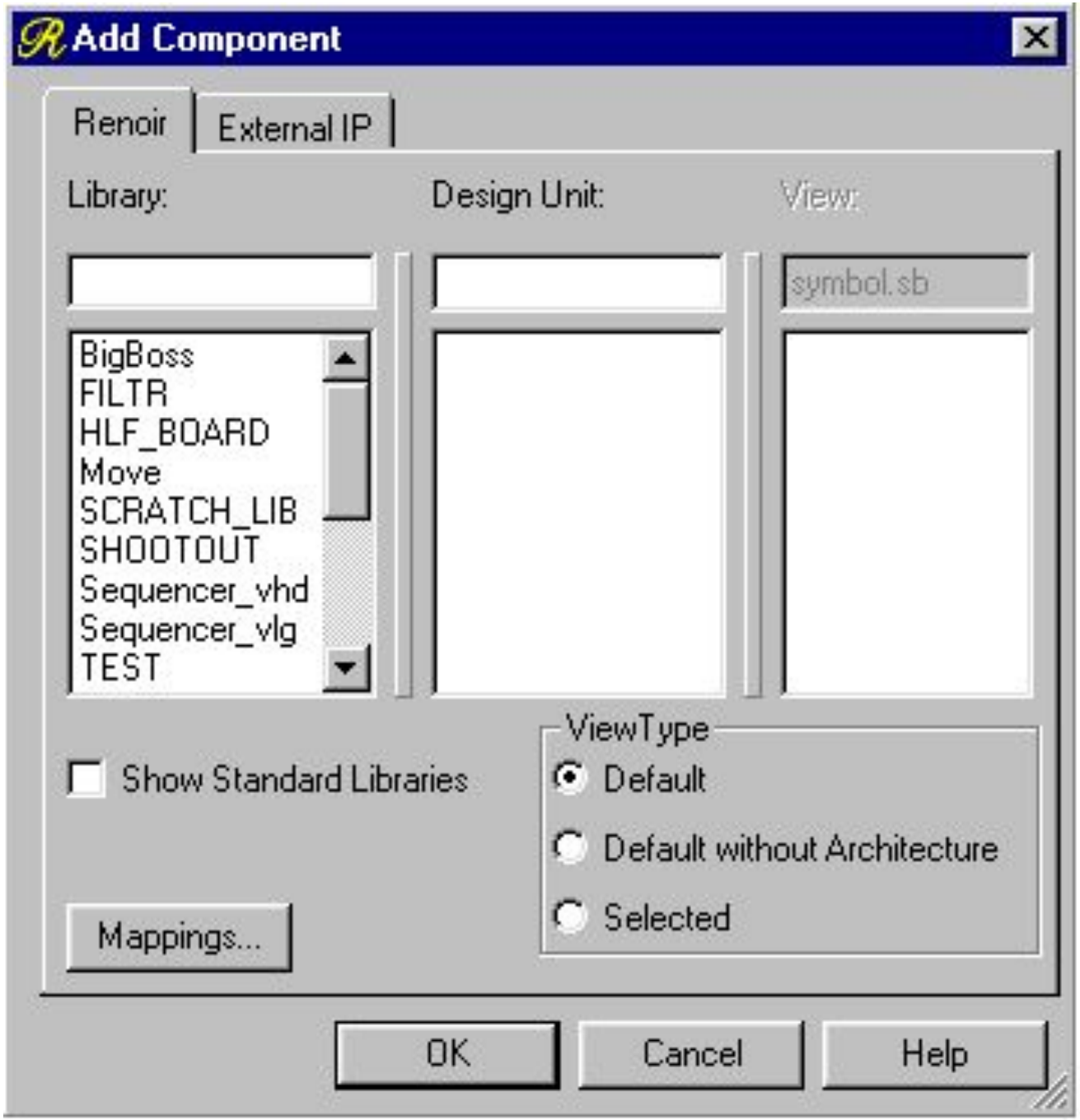
VHDL Package Header - объявление пакета.

Файл верхнего уровня иерархии проекта СБИС, как правило, наиболее целесообразно создавать в виде структурной схемы с помощью команды File/New/Block Diagram.

Помимо традиционных команд работы с файлами (File/New - «Создать новый файл», File/Open - «Открыть файл», File/Save - «Сохранить файл», File/Close Window - «Закрыть», File/Exit - «Выйти из редактора», Edit/Undo - «Отменить последнее действие», Edit/Redo - «Выполнить последнее отмененное действие», Edit/Cut - «Вырезать выделенный фрагмент», Edit/Copy - «Скопировать выделенный фрагмент», Edit/Past - «Вставить скопированный фрагмент», Edit/Delete - «Удалить выделенный фрагмент», Edit/Select All - «Выделить все содержимое файла», Edit/Find - «Найти фрагменты по ключевому признаку», Edit/Replace - «Заменить фрагменты на другие» и др.) меню содержит ряд специфических команд:

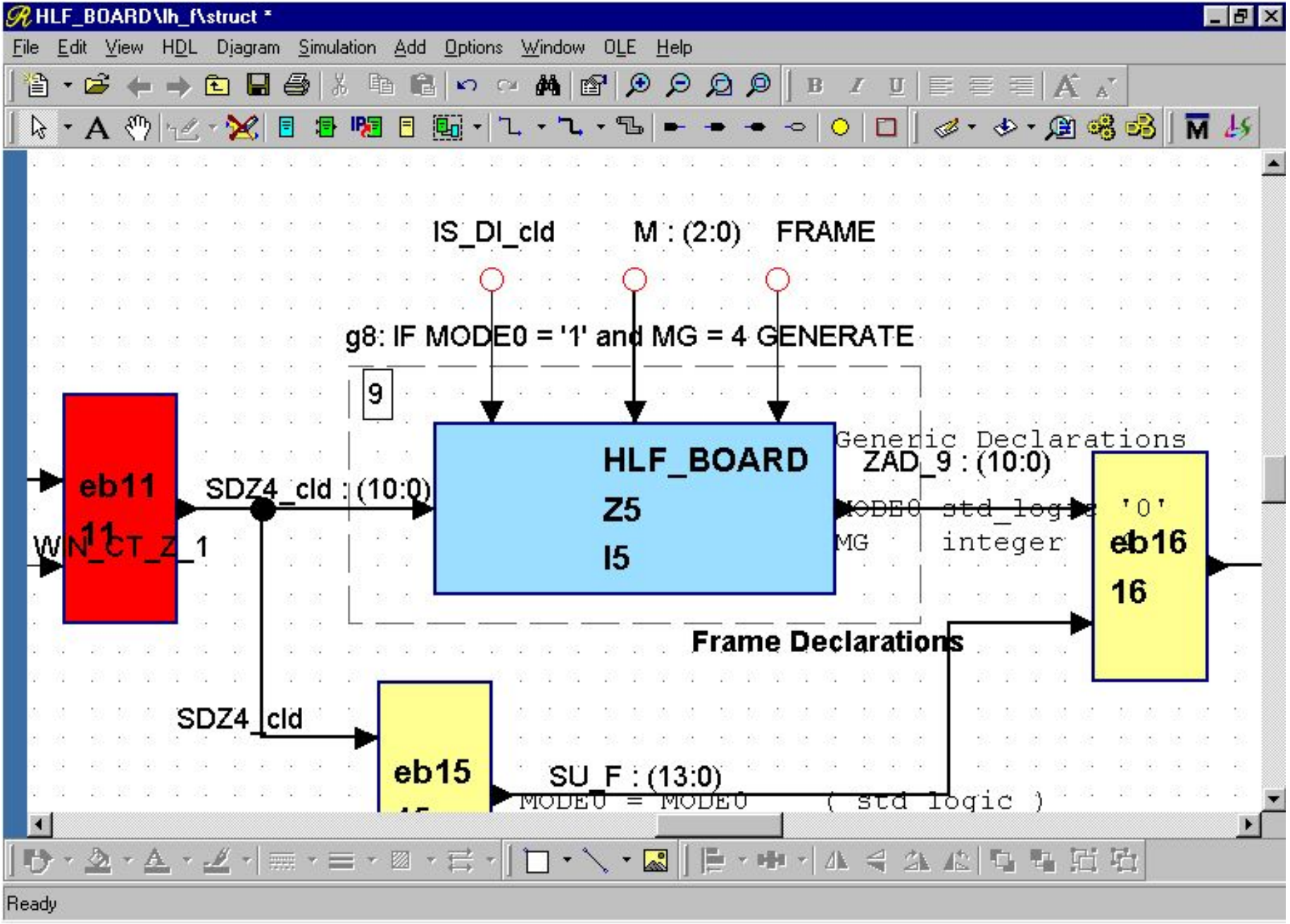
Add/Block - добавление элемента структурной схемы, называемого «Блок», выполняющего определенную функцию обработки или хранения информации, которая может быть задана, в свою очередь, структурной схемой, блок-схемой алгоритма, конечным автоматом, таблицей истинности или непосредственно VHDL-описанием, содержащим как объявление объекта проекта (модуль Entity), так и архитектурное тело (модуль Architecture). После активизации данной команды при наведении курсора «мыши» на поле редактора появляется контур УГО нового блока, который перемещается вместе с курсором. Положение УГО фиксируется нажатием левой кнопки «мыши». Отмена команды осуществляется нажатием правой кнопки «мыши». После добавления блока и сохранения файлов проекта в окне Source оболочки Design Browser в списке объектов соответствующей библиотеки проекта появится ссылка на новый блок с указанным именем;

Add/Component - добавление элемента структурной схемы, называемого «Компонент». Отличие «Компонента» от «Блока» состоит в том, что компонент имеет собственное УГО (Symbol), сохраненное в определенной библиотеке проекта в виде соответствующего файла с расширением *.sb, что позволяет непосредственно и многократно использовать компоненты как из библиотеки данного проекта, так и из других доступных библиотек. При активизации команды добавления компонента вначале появляется диалоговое окно Add Component с закладкой Renoir, содержащее три поля. Необходимо выбрать из списка в поле Library имя библиотеки, содержащей требуемый компонент. При этом в поле Design Unit появится список компонентов данной библиотеки. Необходимо с помощью «мыши» выбрать имя компонента и нажать на кнопку «ОК». В результате на поле редактора появляется контур УГО компонента, который перемещается вместе с курсором. Положение УГО фиксируется нажатием левой кнопки «мыши». Отмена команды осуществляется нажатием правой кнопки «мыши»;


Add Component
Renoir
External IP
Library:
Design Unit:
View:
symbol.sb
BigBoss
FILTR
HLF_BOARD
Move
SCRATCH_LIB
SHOOTOUT
Sequencer_vhd
Sequencer_vlg
TEST
Show Standard Libraries
ViewType
Default
Default without Architecture
Selected
Mappings...
OK
Cancel
Help

Add/Embedded Block - добавление так называемого встроенного блока, который не может быть самостоятельной единицей проекта, так как представляет собой описание лишь отдельных функций, входящих в архитектурное тело. Иными словами, VHDL-код, соответствующий встроенному блоку не содержит ни модуля Entity, ни модуля Architecture. Поэтому, в отличие от блока, содержимое встроенного блока может быть задано одним из четырех способов: в виде блок-схемы алгоритма (Flow Chart), в виде конечного автомата (State Diagram), в виде таблицы истинности (Truth Table) и в виде текстового VHDL-описания (Text). После активизации данной команды при наведении курсора «мыши» на поле редактора появляется контур УГО нового встроенного блока, который перемещается и фиксируется аналогично блоку и компоненту;

Add/Frame - добавление рамки условного синтеза. Если в проекте предусмотрены параметры (Generic), отдельные объекты структурной схемы проекта могут охватываться этими рамками и компилироваться в соответствующие топологические файлы и файлы для моделирования только при выполнении определенных условий, которые задаются пользователем и отражаются в текстовом виде над рамкой;


HLF_BOARD\lh_f\struct *
File Edit View HDL Diagram Simulation Add Options Window OLE Help
IS_DI_cld
M : (2:0)
FRAME
g8: IF MODE0 = '1' and MG = 4 GENERATE
9
HLF_BOARD
Z5
I5
Generic Declarations
ZAD_9 : (10:0)
MODE0 std_logic
MG integer
'0'
eb16
16
eb11
11
SDZ4_cld : (10:0)
WN_CT_Z_1
Frame Declarations
SDZ4_cld
eb15
SU_F : (13:0)
MODE0 = MODE0 (std_logic)
Ready

Add/Signal - добавление сигнала. После активизации данной команды следует навести курсор «мыши» в точку начала сигнальной линии, щелкнуть левой кнопкой и вести курсор до точки предполагаемого окончания линии, щелкая левой кнопкой при необходимости поворота линии. По достижении конечной точки сигнальной линии необходимо один раз щелкнуть левой кнопкой «мыши», если конечная точка является изображением входа элемента структурной схемы, и дважды щелкнуть левой кнопкой в противном случае (при этом на конце сигнальной линии появится красный кружок). Затем необходимо нажать правую кнопку «мыши», навести курсор на изображение линии и дважды щелкнуть левой кнопкой, после чего в диалоговом окне ввести имя сигнала (в поле Name), тип сигнала (в поле Type, например, std_logic) и последовательно нажать кнопки «Apply» и «OK».

Add/Bus - добавление шины (нескольких сигналов, имеющих общее имя с соответствующей индексацией). Все необходимые действия аналогичны добавлению сигнала. Но при идентификации шины необходимо, кроме имени и типа, указать также разрядность в поле Bounds диалогового окна (например, 7 downto 0 для типа std_logic_vector);

Add/Global Connector - добавление глобального сигнала (объявленного и используемого во всех блоках проекта, например, тактового сигнала). После фиксации УГО, вызываемого данной командой (круг желтого цвета), сигнал, линия которого будет с ним соединена, станет глобальным;
Add/Port In - добавление входного внешнего порта;
Add/Port Out - добавление выходного внешнего порта;
Add/Port InOut - добавление двунаправленного внешнего порта.

При описании проекта в виде конечного автомата (команда File/New/State Diagram) на экране монитора появляется окно редактора Renoir со следующим набором команд меню Add:

Add/State - добавление нового состояния, отображаемого на экране в виде круга с именем, определяемым пользователем или по умолчанию. Круг, соответствующий начальному состоянию, по умолчанию выделяется зеленым цветом с двойным контуром. Остальные - голубым цветом;

Add/Transition - добавление перехода между состояниями. Точки плавного изгиба линии перехода фиксируются щелчками левой кнопки «мыши»;

Add/Hierarchical State - добавление иерархического состояния, содержимое которого определяется вложенной диаграммой состояний;
Add/Entry Point - добавление точки входа в диаграмму состояний (только для вложенных диаграмм);
Add/Exit Point - добавление точки выхода из диаграммы состояний (только для вложенных диаграмм).
Определение содержания добавленных состояний и переходов осуществляется после двойного щелчка левой кнопкой «мыши» на изображении состояния или линии перехода.

При описании проекта в виде таблицы истинности (команда File/New/Truth Table) на экране монитора появляется окно редактора Renoir со следующим набором команд Add, которые не вынесены в данном случае в главное меню, а доступны лишь в контекстном меню, появляющемся при нажатии правой кнопки «мыши» на выбранной клетке таблицы:

Add/Column - добавление нового столбца таблицы истинности;

Add/Row - добавление новой строки таблицы истинности.

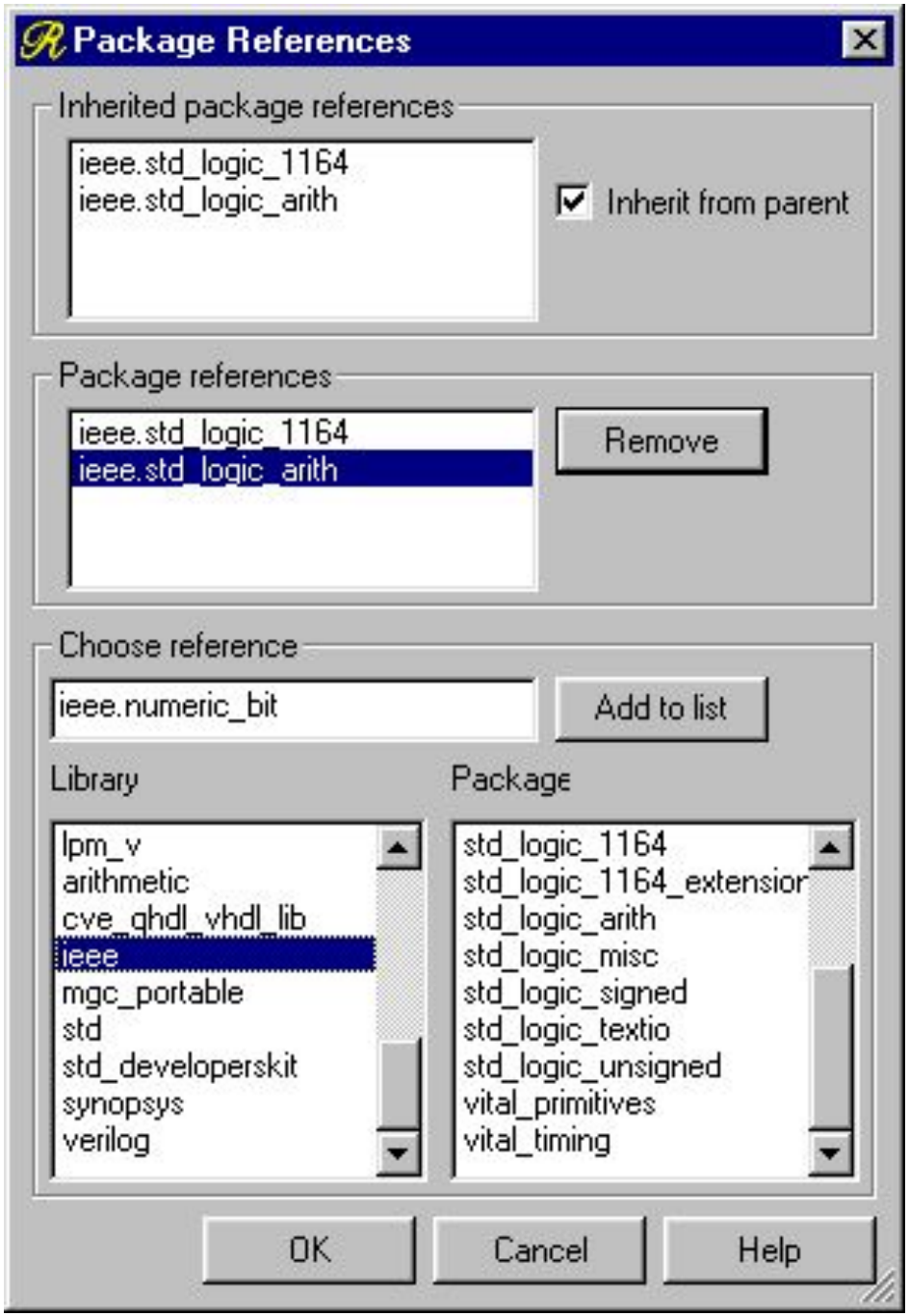
Имеются также команды Delete Column, Delete Row для удаления столбца или строки, на пересечении которых находится выделенная ячейка таблицы. Столбцы входов подсвечены голубым, а выходов - желтым цветом. Для заполнения клеток таблицы необходимо установить курсор «мыши» на нужную клетку, щелкнуть левой кнопкой, после чего вводить необходимые данные (имена и значения входных и выходных сигналов) с клавиатуры.

Описание содержимого каждого из введенных элементов структурной схемы проекта может, в свою очередь, быть выполнено любым из выше перечисленных способов, что позволяет создавать иерархические описания любого уровня сложности (определяемого аппаратными ресурсами САПР). Ввод и редактирование параметров объектов проекта (блоков, компонентов, встроенных блоков и др.) можно осуществить, наведя курсор «мыши» на УГО объекта, щелкнув правой кнопкой и выбрав в контекстном меню команду Object Properties. Команда Reconcile Interface из этого же меню позволяет автоматически согласовать различия в описаниях интерфейсов (внешних выводов) объектов на текущем и нижнем уровнях иерархии.

Подключение библиотек ресурсов в подсистеме Renoir

Помимо рабочей библиотеки, в которой размещаются все файлы проекта, VHDL предусматривает возможность использования ссылок на объекты из внешних библиотек (так называемых библиотек ресурсов).

Подключение и отключение библиотек ресурсов в подсистеме Renoir осуществляется с помощью команды меню Diagram/Package References, при активизации которой выводится соответствующее диалоговое окно


Package References
Inherited package references
ieee.std_logic_1164
ieee.std_logic_arith
Inherit from parent
Package references
ieee.std_logic_1164
ieee.std_logic_arith
Remove
Choose reference
ieee.numeric_bit
Add to list
Library
Package
lpm_v
arithmetic
cve_qhdl_vhdl_lib
ieee
mgc_portable
std
std_developerskit
synopsys
verilog
std_logic_1164
std_logic_1164_extensior
std_logic_arith
std_logic_misc
std_logic_signed
std_logic_textio
std_logic_unsigned
vital_primitives
vital_timing
OK
Cancel
Help

В верхнем поле Inherent Package References отображается список пакетов, «унаследованных» от объектов верхних уровней иерархии. Наследование может быть отменено при удалении значка справа от данного поля.

В поле Package References отображается список подключенных в данный момент пакетов. Удаление отдельных пакетов из данного списка выполняется при помощи кнопки «Remove» справа от поля списка.

Для добавления к списку нового пакета необходимо выделить требуемую библиотеку из списка в поле Library, выделить пакет из списка в поле Package и нажать кнопку «Add to list». При этом имя пакета появится в списке Package References, после чего необходимо нажать кнопку «ОК».

Генерация файлов VHDL-описаний в подсистеме Renoir

Следует отметить, что каким бы из вышеперечисленных способов не было задано содержимое блоков и компонентов проекта, в конечном итоге, после автоматической генерации, проект будет представлен соответствующими VHDL-файлами. Генерация VHDL-описаний может быть выполнена на любом уровне иерархии проекта, причем как для текущего, так и для всех нижележащих уровней. Вне зависимости от способа описания проекта в подсистеме Renoir, генерация описаний осуществляется с помощью одной из следующих команд меню:

HDL/Generate/Single Level - генерация VHDL-файла только текущего уровня описания проекта;

HDL/Generate/Hierarchy - генерация VHDL-описаний проекта с учетом иерархии блоков;

HDL/Generate/Hierarchy Through Components - полная генерация VHDL-описаний проекта с учетом иерархии блоков и содержимого компонентов.

При успешном выполнении генерации, т.е. при отсутствии грамматических и синтаксических ошибок, правильном с точки зрения стандарта VHDL описании всех модулей проекта, в окне Log Window появится сообщение: «Generation completed successfully». В противном случае будут выведены сообщения об ошибках. Причем можно быстро перейти к структурному элементу или строке VHDL-описания, содержащим ошибки, выделив конкретное сообщение об ошибке и щелкнув левой кнопкой «мыши» на одной из кнопок

КОМПИЛЯЦИЯ VHDL-ОПИСАНИЙ В ПОДСИСТЕМЕ RENOIR

После успешного завершения генерации VHDL-файлов необходимо устранить все ошибки, связанные не с нарушением грамматики или синтаксиса, а с некорректным с точки зрения технического задания описанием выполняемых устройством функций. С этой целью выполняют функционально-логическое моделирование проекта, для выполнения которого необходимо провести компиляцию VHDL-описаний, в результате которой будут автоматически выявлены некоторые ошибки и после их устранения созданы файлы в специальном формате для подсистемы ModelSim.

Компиляция VHDL-описаний осуществляется с помощью следующих команд:

HDL/Compile/Single Level - компиляция VHDL-файла только текущего уровня описания проекта;

HDL/Compile/Hierarchy - компиляция VHDL-описаний проекта с учетом иерархии блоков;

HDL/Compile/Hierarchy Through Components - полная компиляция VHDL-описаний проекта с учетом иерархии блоков и содержимого компонентов.

При успешном выполнении компиляции в окне Log Window появится сообщение: «Data preparation step completed, check transcript...». В противном случае будут выведены сообщения об ошибках.

После компиляции VHDL-описаний можно осуществить функционально-логическое моделирование проекта в подсистеме ModelSim, нажав кнопку

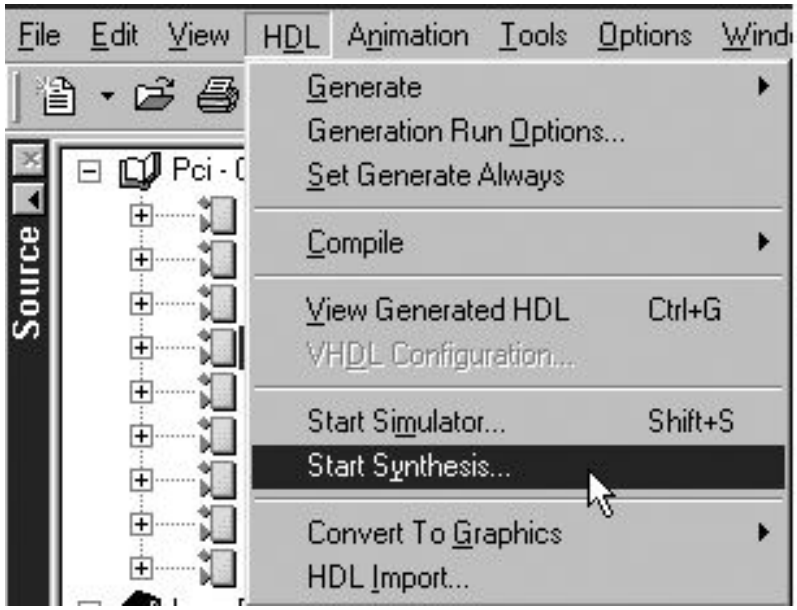
на верхней панели.

СИНТЕЗ ФАЙЛОВ ТОПОЛОГИИ

Синтез топологии проектируемой СБИС осуществляется в отдельной САПР, выбираемой в зависимости от метода реализации СБИС (например, САПР MAX+plus II для реализации на ПЛИС фирмы Altera, САПР Tanner Pro для реализации в виде заказной СБИС и т.д.). Подготовка файлов стандартных форматов описания структуры (например, файлы формата EDIF) и задержек в элементах и линиях связи (файлы формата SDF) осуществляется в подсистеме Leonardo Spectrum. Для выполнения этих операций необходимо нажать кнопку на верхней панели.

Запуск системы логического синтеза Leonardo Spectrum и передача в качестве исходного задания высокоуровневого VHDL-описания выполняется из графической оболочки Renoir с помощью меню "HDL/Start Synthesis", или специализированной кнопки - Start Synthesis.

Кроме того, имеется возможность выполнить весь маршрут подготовительных операций в пакете Renoir (верификация описания, генерация HDL-файла, запуск Leonardo и передача ему проекта) с помощью специализированной кнопки - Synthesis Flow.


File
Edit
View
HDL
Animation
Tools
Options
Source
Generate
Generation Run Options...
Set Generate Always
Compile
View Generated HDL Ctrl+G
VHDL Configuration...
Start Simulator... Shift+S
Start Synthesis...
Convert To Graphics
HDL Import...

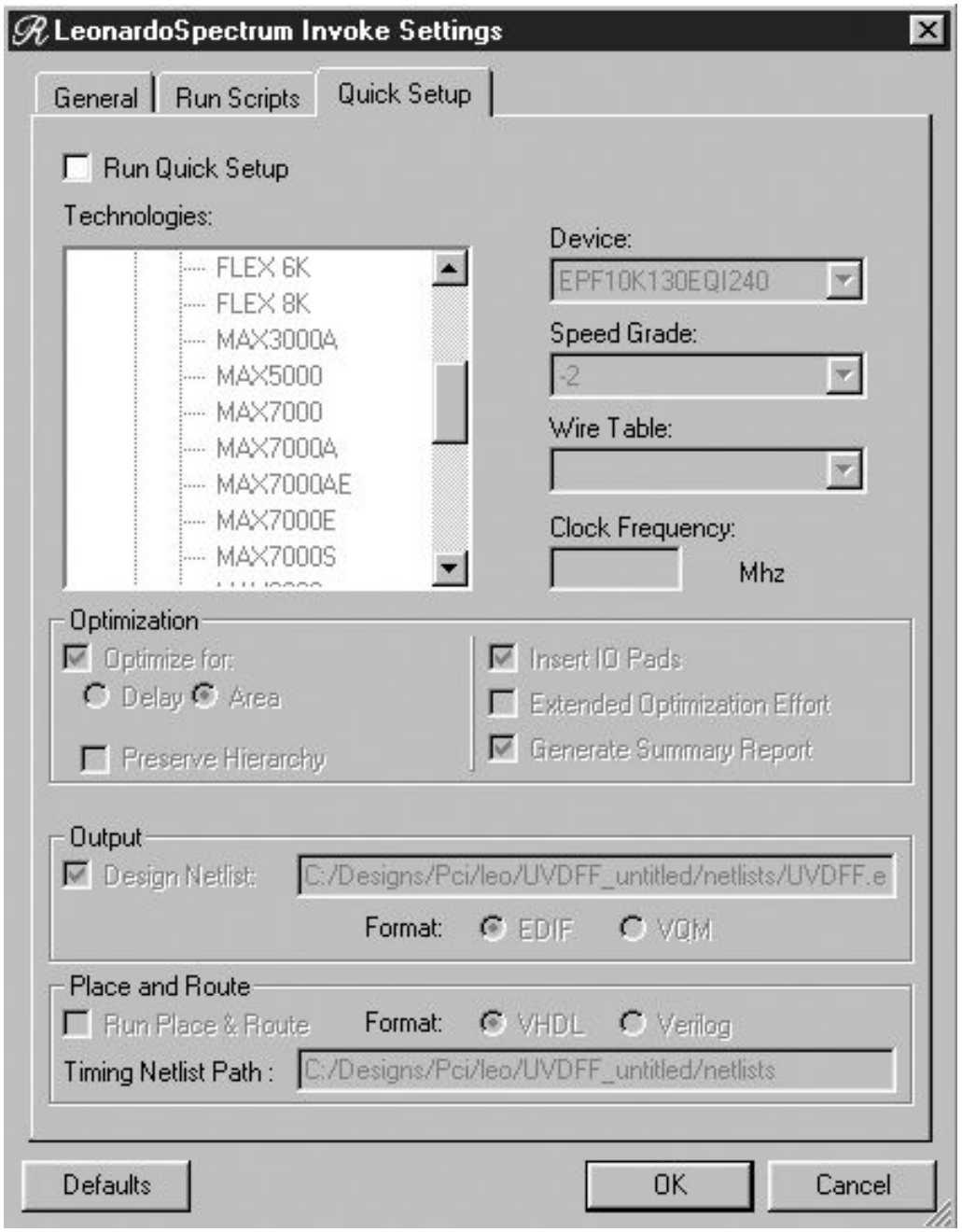
HDL
Start Synthesis
Pci - C:\Designs\Pci\hdl
lpm - D:\CAD\FPGADV40\RENOIR\examples\lpm\hdl

HDL
Synthesis Flow
Pci - C:\Designs\Pci\hdl
lpm - D:\CAD\FPGADV40\RENOIR\examples\lpm\hdl

При этом происходит передача всех файлов, составляющих проект. Появляется окно, позволяющее настроить параметры запуска Leonardo. Как правило, нет необходимости изменять установленные настройки. После нажатия на OK запускается основное окно графического интерфейса Leonardo.

Запуск описанным способом обеспечивает автоматическое считывание VHDL/Verilog-файлов проекта системой Leonardo Spectrum.

В открывшемся окне доступны семь закладок, предназначенных для настройки параметров синтеза и передачи результатов работы в САПР размещения и трассировки.


LeonardoSpectrum Invoke Settings
General
Run Scripts
Quick Setup
Run Quick Setup
Technologies:
FLEX 6K
FLEX 8K
MAX3000A
MAX5000
MAX7000
MAX7000A
MAX7000AE
MAX7000E
MAX7000S
Device:
EPF10K130EQI240
Speed Grade:
-2
Wire Table:
Clock Frequency:
Mhz
Optimization
Optimize for:
Delay
Area
Preserve Hierarchy
Insert IO Pads
Extended Optimization Effort
Generate Summary Report
Output
Design Netlist:
C:/Designs/Pci/leo/UVDFF_untitled/netlists/UVDFF.e
Format:
EDIF
VQM
Place and Route
Run Place & Route
Format:
VHDL
Verilog
Timing Netlist Path :
C:/Designs/Pci/leo/UVDFF_untitled/netlists
Defaults
OK
Cancel

Exemplar Logic - LeonardoSpectrum Level 3 - [Command Line]

File Edit View Tools Window Help

Technol... | Input | Constrai... | Optimize | Report | Output | Place & ...

Click ASIC or FPGA to extend device tree and select a library. Click on the selected technology logo to open the vendor website. Use all defaults.Set advanced technology options under "Advanced Settings". Press "Apply" or "I Library" to apply settings.

ALTERA

- Altera
  - ACEX 1K
  - APEX 20K
  - APEX 20KE
  - FLEX 10K
  - FLEX 10KA
  - FLEX 10KB
  - FLEX 10KE
  - FLEX 6K
  - FLEX 8K
  - MAX3000A
  - MAX5000
  - MAX7000
  - MAX7000A
  - MAX7000AE
  - MAX7000E

Part: EPF10K130EQI240

Speed: -2

☑ Map IO Registers

Load Library | Help

Technology Settings | Advanced Settings

```
->set_working_dir "C:/Designs/Pci/leo/UVDFF_untit
Info, Working Directory is now 'C:\Designs\Pci\le
->source "C:/Designs/Pci/leo/UVDFF_untitled/scrip
Info: Renoir Synthesis run started
-- Reading file D:\CAD\FPGADV40\EXEMPLAR\\data\st
-- Loading package standard into library std
-- Reading vhdl file C:/Designs/Pci/hdl/uvdff_unt
-- Reading file D:\CAD\FPGADV40\EXEMPLAR\\data\st
-- Loading package std_logic_1164 into library ie
-- Searching for SYNOPSYS package std_logic_arith
-- Reading file D:\CAD\FPGADV40\EXEMPLAR\\data\sy
-- Loading package std_logic_arith into library i
-- Loading entity UVDFF into library pci
-- Loading architecture untitled of UVDFF into li
-- Compiling root entity UVDFF(untitled)
Info: Renoir Synthesis run finished
```

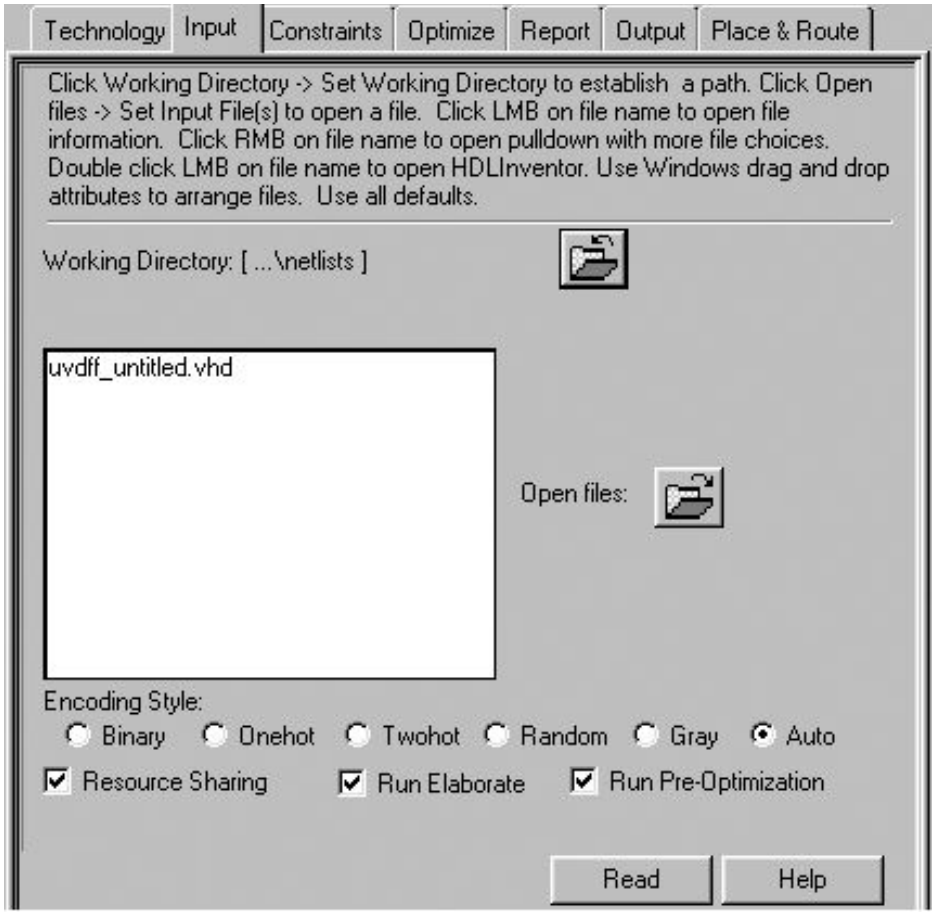
```
1:  set_working_dir "C:/Designs/Pci/leo/UVDFF_untit
2:  source "C:/Designs/Pci/leo/UVDFF_untitled/scrip
3:
```

Transcript | Filtered Transcript

Ready | Working Directory: ...\netlists | Line 3 Col 1

Система Leonardo поставляется с набором готовых библиотек элементов, предназначенных для реализации в рамках конкретной целевой технологии. В набор включены наборы элементов ПЛИС известных производителей: Actel, Altera, Atmel, Cypress, Lattice, Lucent, Minc, QuickLogic, Xilinx.
Выполнение логического синтеза в определенном технологическом базисе требует настройки Leonardo на использованной необходимой библиотеки. Это осуществляется выбором производителя ПЛИС в окне со списком производителей на первой закладке «Technology», а также выбором необходимого семейства микросхем в том же списке, заданием конкретного прибора в поле Part и варианта его исполнения в поле Speed.
Загрузка выбранной библиотеки осуществляется нажатием на кнопку Load Library.

Настройка системы FPGA Advantage такова, что при считывании файлов пакетом Leonardo происходит предварительная оптимизация логической схемы проекта на основании настроек, заданных по умолчанию на закладке «Input». При необходимости эти настройки можно изменить.



Выбор организации кодирования состояния конечных автоматов выбирается в поле Encoding Style. Стиль Binary обеспечивает обычный для двоичной арифметики способ кодирования (например, для трех бит 0 - 000, 1 - 001, 2 - 010, 3 – 011, 4 - 100 и т.д.). Подобный стиль позволяет экономно расходовать триггеры для хранения данных, однако уступает стилю Onehot по быстродействию, при котором для кодирования каждого состояния расходуется отдельный регистр (например, 0 - 0000, 1 - 0001, 2 - 0010, 3 - 0100, 4 - 1000). Как видно, при таком подходе расходуется значительно больше регистров, особенно для больших разрядностей.

Стили Twohot и Random являются разновидностями рассмотренных. Стиль Gray аналогичен Binary, однако использует для кодирования код Грея. Основная его особенность заключается в том, что каждое последующее число отличается от предыдущего только одним битом. Это свойство полезно при построении помехоустойчивых схем.

По умолчанию установлен режим Auto, при котором Leonardo Spectrum автоматически выбирает наиболее оптимальный стиль кодирования.

Отметка свойства Resource Sharing обеспечивает сокращение расходуемых аппаратных ресурсов за счет их совместного использования в разных частях проекта (если это возможно). Например, два ресурсоёмких сумматора могут быть заменены на один сумматор и два мультиплексора, управляющих входами и выходами сумматора.

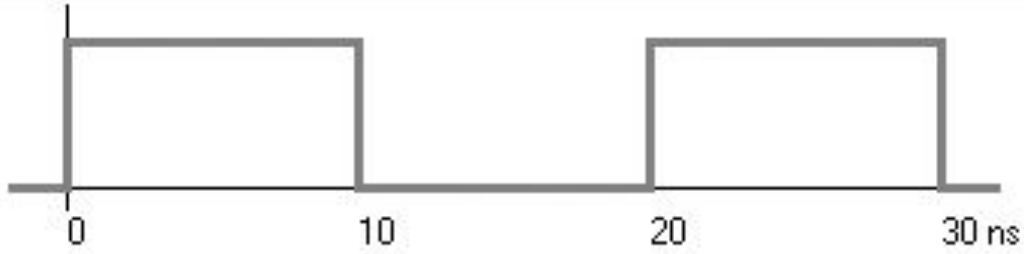
Отметка полей Run Elaborate и Run Pre-Optimization обеспечивают выполнение предварительной оптимизации при считывании.

На следующей закладке «Constraints» имеется возможность задать основные требования к результатам синтеза схемы устройства в определенном технологическом базисе.
В поле Specify Clock Frequency задается требование к частоте тактового сигнала. В случае, если необходимо задать период сигнала, нужное значение заносится в поле Specify Clock Period.
Для того, чтобы задать специфические задержки, служат поля Input Ports to Registers, Registers to Registers, Registers to Output Ports, Inputs to Outputs.

Technology | Input | Constraints | Optimize | Report | Output | Place & Route

Specify clock frequency, clock cycle, and global path constraints for the entire design. The smallest design for a given frequency is then created. All paths between ports and registers are constrained to one clock period. You can customize delays between ports and registers by specifying a Maximum Delay between each. The clock reference time is zero.

- (•) Specify Clock Frequency: 50 MHz
- ( ) Specify Clock Period: 20 ns
- ( ) Specify Maximum Delay Between all:
  - Input Ports to Registers: 20 ns
  - Registers to Registers: 20 ns
  - Registers to Output Ports: 20 ns
  - Inputs to Outputs 20 ns



Apply | Help

Global | Clock | Input | Output | Signal | Module | Path | Report

Если в схеме имеется несколько синхронизирующих сигналов, временные требования к каждому из них можно специфицировать на подзакладке «Clock», выбираемой в нижней части поля закладки «Constraints»

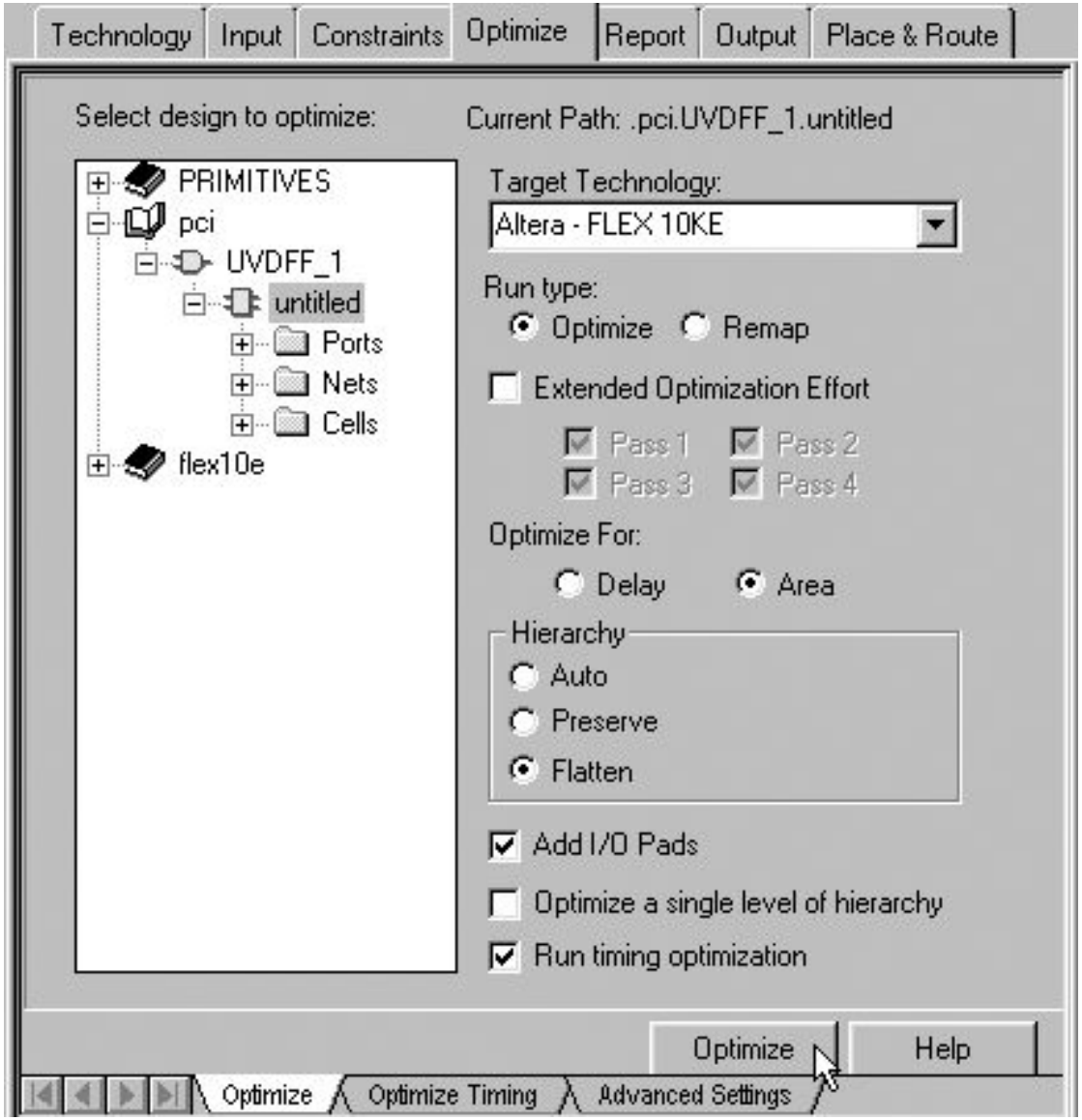
Подзакладки «Input» и «Output» обеспечивают возможность задать ограничения для каждого из портов схемы.

На подзакладке «Signal» выделяются сигналы, которые не должны быть удалены при выполнении оптимизации схемы. На подзакладке «Module» аналогично отмечаются модули, не подлежащие общей оптимизации, и устанавливаются критерии оптимизации для каждого из них.

Подзакладка «Path» предназначена для выделения путей, подлежащих исключению из общих требований и ограничений.

На подзакладке «Report» имеется возможность просмотреть все установленные требования и сохранить их в файл для дальнейшего использования, а также загрузить ранее сохраненные подобные файлы.

Запуск оптимизации проекта в выбранном технологическом базисе выполняется на закладке «Optimize» нажатием на кнопку Optimize



Если необходимо выполнить только переложение схемы в библиотечные элементы технологического базиса, в поле Run type выбирается отметка Remap, иначе оптимизация будет выполнена.

Отметкой поля Extended Optimization Effort настраивается количество итераций при выполнении оптимизации.

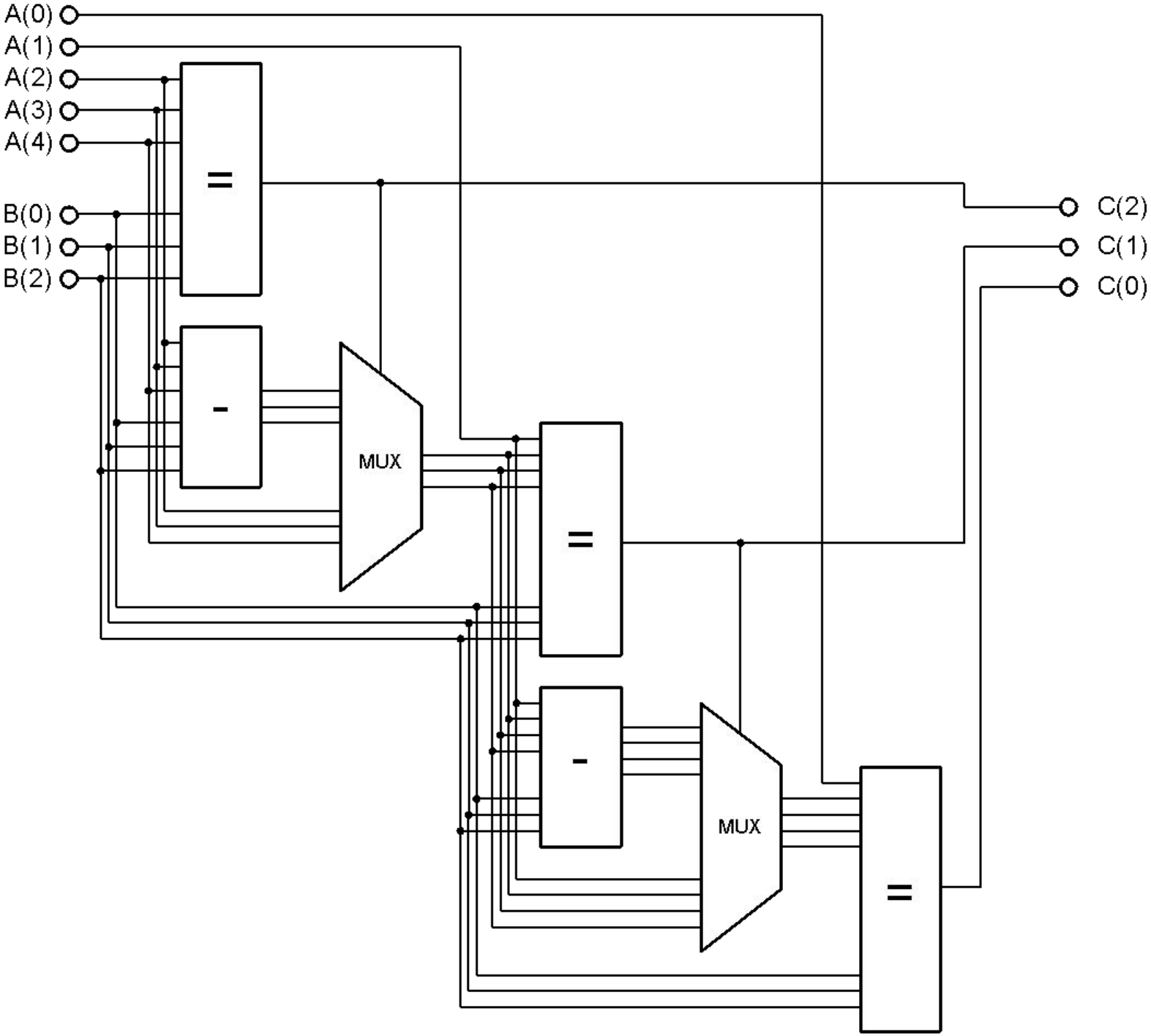
В поле Optimize For указывается основной критерий оптимизации – достижение наибольшего быстродействия, или наименьшей занимаемой площади кристалла.

Поле Hierarchy определяет возможность удалять иерархическую структуру проекта. В случае, если выбрана отметка Preserve, иерархия проекта сохраняется, то есть все компоненты схемы останутся отдельными блоками. Выбор значения Flatten обеспечивает преобразование схемы устройства в один уровень иерархии, что, как правило, обеспечивает достижение более качественных результатов оптимизации. Отметка Auto позволяет системе Leonardo Spectrum автоматически определить наиболее оптимальный выбор.

Поле Add I/O Pads позволяет включить в схему буферные библиотечные элементы, подключаемые к контактным площадкам. Это используется при синтезе полной схемы, размещаемой на одном кристалле.

Поле Optimize a single level of hierarchy обеспечивает выполнение оптимизации только для верхнего уровня иерархии. Выбор этого свойства полезен при использовании тщательно отработанных технологически зависимых схемных решений отдельных структурных компонентов.

Отметка поля Run timing optimization обеспечивает выполнение всех временных и частотных требований.


A(0)
A(1)
A(2)
A(3)
A(4)
B(0)
B(1)
B(2)
=
-
MUX
=
-
MUX
=
C(2)
C(1)
C(0)