

681.32 (075)

Ш....

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ  
Технологический институт  
Федерального государственного образовательного  
учреждения высшего профессионального образования  
«Южный федеральный университет»  
ПРИОРИТЕТНЫЙ НАЦИОНАЛЬНЫЙ ПРОЕКТ «ОБРАЗОВАНИЕ»**

**Шеболков В.В.**

**Современные средства автоматизированного  
проектирования цифровых устройств на ПЛИС**

**Учебное пособие**

**Часть 1**

**Таганрог 2007**

УДК 621.317

В.В. Шеболков. Современные средства автоматизированного проектирования цифровых устройств на ПЛИС. Учебное пособие. - Таганрог: Изд-во ТТИ ЮФУ, 2007.- 65с.

Дается краткая характеристика современных ПЛИС и технологий схемотехнического проектирования цифровых устройств на них. Рассмотрены типичные системы автоматизированного проектирования цифровых устройств на ПЛИС (MAX Plus, Foundation Series, ISE). Описана технология создания проекта иерархической структуры в варианте “снизу-вверх” и на примере системы MAX Plus II проиллюстрированы принципы проектирования цифрового устройства с иерархической структурой. Рассмотрены варианты технологии работы над проектами нижних уровней при графическом вводе информации о проектируемом устройстве и на языке VHDL.

Табл. 3. Илл. 31. Библиогр.; 10 назв.

Рецензенты:

© Технологический институт  
Южного федерального университета, 2007  
© В.В. Шеболков, 2007

ВВЕДЕНИЕ .....	4
1. СОВРЕМЕННЫЕ ПЛИС И ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ УСТРОЙСТВ .....	6
2. ХАРАКТЕРИСТИКА СИСТЕМ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ УСТРОЙСТВ НА ПЛИС .....	12
2.1. САПР MAX+Plus II .....	12
2.2 САПР Foundation Series.....	16
2. 3. САПР Integrated Software Environment .....	17
3. ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ СХЕМ В СИСТЕМЕ MAX+PLUS II .....	20
3.1. Интерфейс САПР MAX+Plus II .....	20
3.2. Принципы работы с системой.....	23
3.3. Разработка проекта нижнего уровня с помощью графического редактора.....	27
3.3. Разработка проектов цифровых узлов на основе HDL - языков .....	43
3.4. Создание графического файла проекта верхнего уровня иерархии.....	51
ЗАКЛЮЧЕНИЕ .....	62
Библиографический список .....	62

## **ВВЕДЕНИЕ**

Программируемые логические интегральные схемы (ПЛИС) представляют одно из самых интересных и быстро развивающихся направлений современной цифровой микроэлектроники. Последние 10 – 15 лет наблюдался бурный рост рынка этих устройств и существенное улучшение их характеристик.

С появлением ПЛИС появилась возможность проектировать и выпускать небольшие партии уникальных цифровых устройств в условиях проектно-конструкторских подразделений небольших промышленных предприятий, исследовательских, учебных и даже домашних радиолюбительских лабораториях. Промышленно выпускаемые «заготовки» программируемых микросхем с электрическим программированием делают проектирование новых цифровых устройств, сравнимым с разработкой программного обеспечения.

В настоящее время ведущими мировыми производителями ПЛИС являются фирмы Xilinx и Altera. Каждая из них выпускает широкий спектр продукции, включая ПЛИС с различной архитектурой, флеш-ПЗУ для хранения конфигурации, системы автоматизированного проектирования (САПР), средства программирования и отладки. Немаловажным является тот факт, что САПР минимальной конфигурации этих фирм распространяется бесплатно, а их возможности вполне достаточны для освоения данной технологии и разработки цифровых устройств начального и среднего уровня.

Как правило, в современных САПР цифровых устройств предусматривается несколько вариантов ввода информации о проектируемом устройстве: в графическом виде – в виде схемы или графа состояний; текстовом виде – на одном из HDL (Hardware Description Language) языков: VHDL, Verilog, AHDL и т.д. Современные САПР поддерживают создание проектов иерархической структуры, созда-

ние и поддержку пользовательских библиотек, импорт и экспорт проектов для других САПР.

Достаточно развитые бесплатные версии САПР и документацию по архитектуре и применению ПЛИС фирмы «Altera» можно бесплатно «скачать» с сайта [www.altera.com](http://www.altera.com), фирмы «Xilinx» – с сайта [www.xilinx.com](http://www.xilinx.com).

## 1. СОВРЕМЕННЫЕ ПЛИС И ТЕХНОЛОГИИ ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ УСТРОЙСТВ

Первые образцы программируемых логических интегральных схем (ПЛИС) или программируемые логические устройства (ПЛУ – Programmable Logic Devices – PLD) появились в начале 70-х годов в виде программируемых постоянных запоминающих устройств (ППЗУ – Programmable Read Only Memory – PROM). Вначале они использовались исключительно для хранения данных, а затем их стали применять для реализации логических функций. Однако необходимость приведения логических функций к совершенной дизъюнктивной нормальной форме (СДНФ) ограничила применение PROM для реализации функций больших размеров.

Специально для реализации систем булевых функций (СБФ) большого числа переменных были разработаны программируемые логические матрицы (ПЛИМ – Programmable Logic Arrays – PLAs). PLA получили очень широкое распространение в качестве универсальной элементной базы цифровых устройств, поэтому именно PLA можно считать первыми PLD.

Совершенствование архитектуры PLA привело к появлению программируемых матриц логики (ПИМЛ – Programmable Array Logics – PALs), которые до настоящего времени определяют наиболее популярную архитектуру PLD. С момента своего появления PAL стали успешно конкурировать с PLA и в настоящее время благодаря ряду присущих им положительных свойств практически полностью вытеснили программируемые пользователем PLA.

Дальнейшее совершенствование технологии производства интегральных схем в начале 90-х годов привело к возможности реализации на одном кристалле нескольких PAL, объединяемых программируемыми соединениями. Подобные архитектуры получили название *сложных* ПЛУ (Complex PLD – CPLD), а все ранее разработанные PLD стали называть *стандартными ПЛУ* (Standart PLD – SPLD) или

*классическими ПЛУ (Classic PLD).*

Параллельно с PLD также развивались архитектуры *вентильных матриц* (Gate Array – GA) и *матриц логических ячеек* (Logic Cell Array – LCA), в отечественной литературе получившие название *базовых матричных кристаллов* (БМК). Первые вентильные матрицы были полузаказными, т.е. программировались во время изготовления, что сдерживало их широкое практическое использование. Однако в 1985 году фирма Xilinx выпустила *программируемую пользователем вентильную матрицу* (Field Programmable Gate Array – FPGA) [25]. Это дало сильный толчок к широкому распространению вентильных матриц и конкуренции их с PLD.

В отечественной литературе нет четкого разделения между PLD, PAL, PLA, SPLD, CPLD и FPGA. Чаще всего все эти устройства называют *программируемыми логическими интегральными схемами*.

ПЛИС представляют собой новую элементную базу, обладающую гибкостью заказных БИС и доступностью традиционной "жесткой" логики. Наиболее важным отличительным свойством ПЛИС является возможность их настройки на выполнение заданных функций самим пользователем. Современные ПЛИС характеризуются низкой стоимостью, высоким быстродействием, широкими функциональными возможностями, возможностью многократностью перепрограммирования, низким потреблением энергии. При этом время разработки даже достаточно сложных цифровых устройств на основе ПЛИС может составлять всего несколько часов. По существу, разработка устройств на основе ПЛИС представляет собой новую технологию проектирования электронных схем, включая их изготовление и сопровождение.

Простейшим видом программируемого логического устройства является обычная микросхема ПЗУ. Действительно, обладая  $N$  адресными линиями и  $M$  линиями данных, микросхема ПЗУ может реализовать  $M \cdot N$ -входовых логических функций. Содержимое ПЗУ может рассматриваться как таблица истинности некоторой цифровой схемы, содержащей комбинаторную логику. Использование микросхем ПЗУ в

качестве логических генераторов долгое время являлось эффективным приемом, позволявшим резко уменьшить число корпусов микросхем, выполнявших элементарные логические операции. Программируемые устройства подобного типа называются PAL (Programmable Array of Logic).

Недостатком PAL-устройств является отсутствие в их составе триггеров. Несмотря на то, что, например, RS-триггер может быть реализован с использованием базиса Пирса или Шеффера (т. е. с применением только комбинаторных логических устройств), очевидно, что такое использование ресурсов ПЛИС весьма непродуктивно, поскольку вместо N-входовых логических элементов применяются только 2-входовые.

Для более эффективного использования логических ресурсов в состав программируемой микросхемы вводят триггеры. Однако в этом случае, кроме программирования таблиц истинности логических элементов, необходимо реализовать также внутренние соединения между выходами логических элементов и входами триггеров. Вместо металлических соединений в ПЛИС используются соединения, коммутируемые программируемыми ключами. Для нормального функционирования этих соединений в ПЛИС существует теневая (конфигурационная) память, хранящая таблицу соединений.

В настоящее время наиболее распространенные серии ПЛИС имеют следующую архитектуру [3, 6, 9]:

- a) CPLD – устройства, использующие для хранения конфигурации энергонезависимую память (Flash или EEPROM);
- b) FPGA – устройства, использующие для хранения конфигурации энергозависимую память, которая требует инициализации после включения питания.

CPLD – это ПЛИС, содержащая несколько матричных логических блоков (MLB), объединенных коммутационной матрицей. Такие ПЛИС имеют высокую степень интеграции (до 10000 эквивалентных вентилях, 256 макроячеек). К этому классу относятся, например, ПЛИС семейства MAX5000 MAX7000 фирмы «Altera», схемы



XC7000 и XC9500 фирмы «Xilinx».

FPGA – это ПЛИС, построенная из программируемых вентилярных матриц (ПВМ), состоящих из логических блоков (LB) и коммутирующих программируемых матриц соединений. Для реализации управляющих и интерфейсных схем в FPGA совместно с ПВМ применяют CPLD. К этому классу относятся ПЛИС XC2000, EC3000, XC4000, Spartan, Virtex фирмы «Xilinx», семейство FLEX8000, FLEX10K фирмы «Altera». Характерным для FPGA-архитектуры являются наличие элементов ввода-вывода (Input/Output Blocks, IOBs), позволяющих реализовать двунаправленный ввод/вывод, третье (высокоимпедансное) состояние и т.п.

В настоящее время наблюдается бурное развитие архитектур CPLD и FPGA, снижение их стоимости, повышение быстродействия и функциональной мощности. Это позволяет предположить, что в ближайшей перспективе основу элементной базы цифровых систем будут составлять CPLD и FPGA.

Современные ПЛИС выпускаются с возможностью программирования в системе непосредственно на плате. Для программирования и загрузки конфигурации устройств можно использовать загрузочный кабель ByteBlaster и ByteBlaster MV [10] (см. прил. 1). Конфигурационные ПЗУ EPC2 и EPC16 позволяют выполнять программирование с помощью этого устройства, тем самым отпадает нужда в программаторе, что, безусловно, снижает стоимость овладения технологией.

По сравнению со специализированными цифровыми микросхемами (Application Specific Integral Circuit, ASIC), цикл разработки устройств на ПЛИС занимает значительно меньшее время и неизмеримо дешевле (благодаря тому, что изменение принципиальной электрической схемы выполняется путем перепрограммирования одного и того же экземпляра микросхемы).

Особенностями современных ПЛИС являются возможность программирования и реконфигурирования в системе, тестирования узлов с помощью порта JTAG (B-scan), а также наличие внутреннего генератора (Osc) и схем управления последовательной конфигураци-

ей. Понятие «программирование в системе» относится к тем ПЛИС, которые позволяют выполнить их программирование непосредственно в составе системы без использования программатора, на смонтированной плате, причем программирование ПЛИС или конфигурационного ПЗУ может производиться многократно. Реконфигурирование в схеме позволяет произвести перезагрузку данных в ПЛИС, построенной по SRAM-технологии «на лету», то есть без выключения питания системы и последующей загрузки новой конфигурации.

Дальнейшее развитие ПЛИС идет по пути создания комбинированных архитектур, сочетающих удобство реализации алгоритмов ЦОС на базе таблиц перекодировок и реконфигурируемых модулей памяти, характерных для FPGA-структур и многоуровневых ПЛИС с удобством реализации цифровых автоматов на CPLD-архитектурах. Так, ПЛИС APEX20K фирмы «Altera» содержат в себе логические элементы всех перечисленных типов, что позволяет применять ПЛИС как основную элементную базу для «систем на кристалле» (System-On-Chip, SOC). В основе идеи SOC лежит интеграция всей электронной системы в одном кристалле (например, такой чип объединяет процессор, память, и т.д.). Компоненты этих систем разрабатываются отдельно и хранятся в виде файлов параметризуемых модулей. Окончательная структура SOC-микросхемы выполняется на базе этих «виртуальных компонентов» с помощью программ – систем автоматизации проектирования (САПР) электронных устройств – EDA (Electronic Design Automation).

Достаточно подробное описание архитектуры ПЛИС разных типов можно найти в работе [9].

Проектирование цифровых устройств на основе схем программируемой логики высокой сложности выполняется только с помощью САПР. Схема алгоритма проектирования показана на рис. 1.1.

Проектирование на концептуальном уровне возлагается на проектировщика, и слабо связано с автоматизацией. На этом уровне по существу определяется логика работы устройства, множества

входных и выходных сигналов, их характер и взаимосвязь, осуществляется разбиение проекта на части и т. д. Результаты концептуального синтеза вводятся в САПР, которая выполняет компиляцию проекта, т. е. синтезирует устройство в базе библиотек своих моделей. Созданный проект требует тщательной проверки, поэтому за этапом синтеза следует этап анализа, проводимого моделированием и теоретической верификацией.

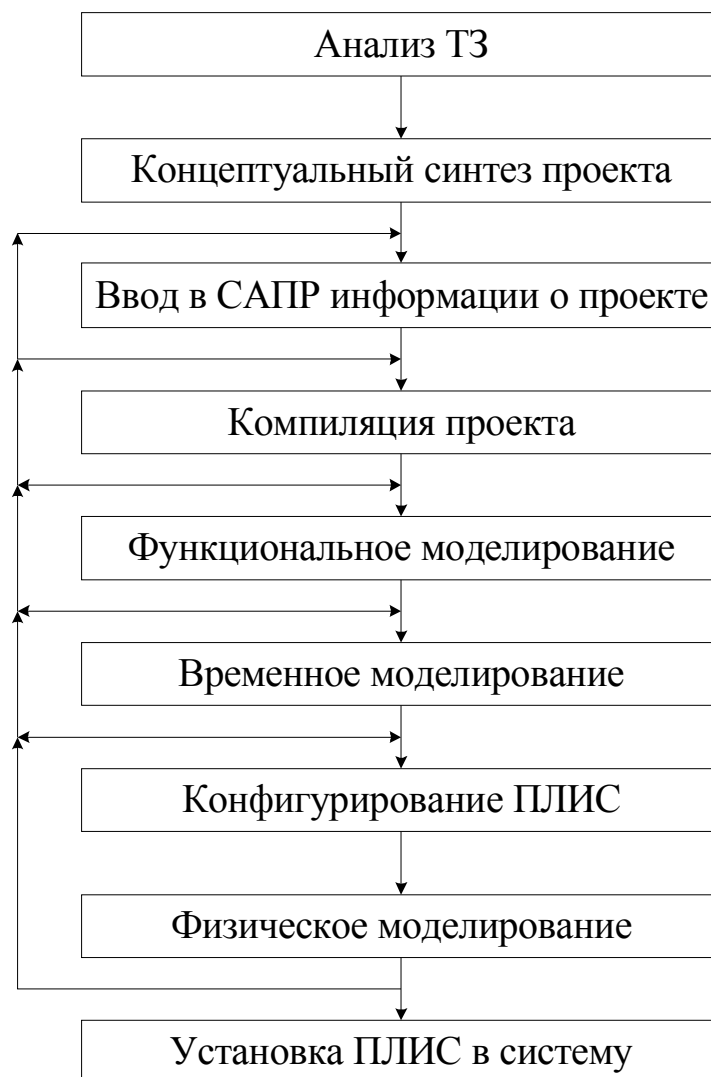


Рис. 1.1

Моделирование решает несколько различных задач, отражающих различные свойства проектируемого объекта. Оно может быть функциональным – проверяющим правильность логической структу-

ры устройства, временным – учитывающим задержки сигналов в схемах устройства и т. д. В результате моделирования могут выявиться ошибки, требующие исправления, что придает процессу проектирования итеративный характер с возвратами к прежним этапам и необходимой корректировкой проекта.

Далее выполняется конфигурирование ПЛИС, после чего проверяется работа реального устройства – физическое моделирование проекта. При успешном завершении физического моделирования устройство готово к установке в систему.

## **2. ХАРАКТЕРИСТИКА СИСТЕМ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ УСТРОЙСТВ НА ПЛИС**

### **2.1. САПР MAX+Plus II**

К сегодняшнему дню разработаны и широко используются целый ряд программных средств и САПР, предназначенных для проектирования цифровых устройств на ПЛИС [1, 5, 8]: MAX Plus, Quartus фирмы «Altera», Foundation Series, ISE (Integrated Software Environment) фирмы «Xilinx», Active HDL, OrCAD и другие. Эти пакеты обладают всеми чертами современных САПР: они обеспечивают сквозной цикл проектирования цифровых устройств от ввода информации о схеме или логике работы проектируемого устройства до верификации проекта:

1) ввод информации о входных и выходных сигналах и логике работы проектируемой схемы;

2) моделирование (симуляция) работы проектируемой схемы и анализ временных соотношений между сигналами в ней;

3) выбор типа микросхемы ПЛИС, на которой предполагается реализация схемы и указание предпочтений при размещении тех или иных групп или компонентов электронной схемы в теле кристалла микросхемы;

4) программирование или конфигурирование ПЛИС;

5) тестирование ПЛИС на плате и итерационная корректировка проекта (осуществляется в случае некорректной работы устройства).

Типичная современная система автоматизированного проектирования цифровых устройств обработки информации на ПЛИС представляет собой замкнутую программную оболочку, в которой может выполняться полный цикл проектирования от ввода информации о логике работы цифровой схемы до тестирования на плате образца спроектированного устройства. Такие САПР, как правило, позволяют вводить информацию о логике работы проектируемой схемы в графическом или текстовом виде.

Графический ввод информации может осуществляться как в виде электрической схемы проектируемого устройства, которая строится либо из логических примитивов, либо из серийно выпускаемых цифровых компонентов, так и (в наиболее развитых САПР) в виде диаграммы состояний цифрового устройства.

Все без исключения САПР поддерживают ввод информации о проектируемом устройстве на одном или нескольких HDL-языках (VHDL, Verilog, AHDL, EDIF).

Одной из наиболее простых САПР цифровых схем на ПЛИС, обладающей типичными чертами современной САПР, является система автоматизированного проектирования MAX+PLUS II, созданная фирмой Altera.

Название системы MAX+PLUS II является аббревиатурой от Multiple Array Matrix Programmable Logic User System. Она содержит программные средства для ввода проекта, его компиляции и отладки, а также непосредственного программирования программируемых логических интегральных схем фирмы «Altera».

Программное обеспечение системы MAX+PLUS II содержит 11 приложений (табл. 2.1), объединенных с помощью управляющей программы. Любое из приложений активизируется из управляющей программы — это позволяет переключаться между ними щелчком мыши

или с помощью команд меню. Во время работы с одним из приложений другое может выполняться в фоновом режиме (например, компилятор, симулятор, анализатор синхронизации или программатор). Одноименные команды разных приложений выполняют одинаковые функции.

Таблица 2.1

## Описание приложений системы MAX+PLUS II

Приложение	Выполняемая функция
<b>Hierarchy Display</b>	<b>Обзор иерархии</b> – отображает текущую иерархическую структуру файлов в виде дерева с ветвями, представляющими собой проекты нижних уровней иерархии. Очень удобное средство для навигации в проекте и для оперативного перехода между разными файлами проекта
<b>Graphic Editor</b>	<b>Графический редактор</b> – позволяет разрабатывать схемный логический проект в формате реального отображения на экране. Оперирует как с графическими примитивами поставляемыми в составе пакета MAX+PLUS II так и с графическими символами компонентов пользователя
<b>Symbol Editor</b>	<b>Редактор символов</b> – предназначен для редактирования существующих символов и создания новых
<b>Text Editor</b>	<b>Текстовый редактор</b> – средство для создания и редактирования текстовых файлов проектов на языках AHDL, VHDL, Verilog
<b>Waveform Editor</b>	<b>Сигнальный редактор</b> – инструмент для ввода тестовых цифровых сигналов и наблюдения результатов моделирования работы файлов проекта
<b>Floorplan Editor</b>	<b>Поуровневый планировщик</b> – графический редактор для назначения выводов ПЛИС и распределения их ресурсов
<b>Compiler</b>	<b>Компилятор</b> – обрабатывает проекты: анализирует логику работы проектируемых устройств, проводит оптимизацию их логической структуры, проводит синтез и трассировку электрической схемы устройства для назначенного типа ПЛИС
<b>Simulator</b>	<b>Симулятор</b> – позволяет тестировать логику работы и временные соотношения в проектируемой логической схеме. Возможны три режима тестирования: функциональное, временное и тестирование нескольких соединенных между собой устройств

Окончание табл.2.1

Приложение	Выполняемая функция
<b>Timing Analyzer</b>	<b>Временной анализатор</b> – анализирует работу проектируемой логической схемы после того, как она была синтезирована и оптимизирована компилятором, позволяет оценить задержки, возникающие в схеме
<b>Programmer</b>	<b>Программатор</b> – предназначен для программирования, конфигурирования, проводит верификацию и тестирование проектируемого устройства на ПЛИС фирмы «ALTERA»
<b>Message Processor</b>	<b>Генератор сообщений</b> – выдает на экран сообщения об ошибках, предупреждающие и информационные сообщения в процессе проектирования

С точки зрения проектировщика система MAX+PLUS II представляет собой замкнутую программную оболочку, в которой может выполняться полный цикл проектирования цифровых устройств обработки информации на ПЛИС фирмы «Altera» . Этот цикл включает в себя следующие этапы:

1) ввод исходной информации о проектируемом устройстве: его электрической схеме или логике работы, допустимых временных параметрах, типах ПЛИС, на которых предполагается реализовать устройство, о желательном размещении выводов и отдельных частей устройства внутри ПЛИС и т.д.; для решения этих задач используются графический (**Graphic Editor**) и текстовый (**Text Editor**) редакторы и поуровневый планировщик (**Floorplan Editor**);

2) компиляция проекта включает построение списка соединений (**Netlist Extractor**), базы данных проекта (**Data Base Builder**), логический синтез (**Logic synthesis**), извлечение временных и функциональных параметров проекта (**SNF Extractor**), разбиение проекта на части (**Partioner**), трассировку (**Fitter**) и формирование файла для программирования или загрузки ПЛИС (**Assembler**); для решения этих задач используются перечисленные программные модули, заключен-

ные в общую оболочку – **Compiler**;

3) верификация проекта включает проверку правильности логики работы цифровых схем и временных соотношений между сигналами; для решения этих задач используются программа моделирования работы цифровых устройств (**Simulator**), сигнальный редактор (**Waveform Editor**) и временной анализатор (**Timing Analyzer**);

4) программирование физических устройств, на которых реализуется проект, или загрузка конфигурации спроектированного устройства в память конфигурационных ПЛИС выполняются модулем программатора (**Programmer**) с использованием соответствующего аппаратного обеспечения (конфигурация ПЛИС может осуществляться через последовательный или параллельный порты либо с использованием интерфейса JTAG [1, 2, 10]).

Разработка проекта может быть ускорена путем использования стандартных функций (в том числе примитивов), мегафункций, библиотеки параметризованных модулей (LPM) а также макрофункций широко известных микросхем 74 серии.

## 2.2 CAIP Foundation Series

CAIP Foundation Series представляет собой систему сквозного проектирования цифровых устройств на базе ПЛИС Xilinx. Этот пакет включает в себя средства схемотехнического ввода, языки описания аппаратуры (HDL) - VHDL, Verilog и Abel, средства моделирования, синтеза структуры кристалла и программирования. CAIP Foundation Series работает под управлением операционных систем Windows 98, Windows 2000.

Для запуска и нормальной работы достаточен объем ОЗУ превышающий 64 Мб. Преимущества от установки большего объема памяти проявляются при работе с ПЛИС объемом более 1 млн. вентилей, однако в настоящее время для таких проектов рекомендуется использовать оболочку проектирования ISE.

Для установки требуется около 500 Мб дисковой памяти (реко-



мендуется иметь дополнительно около 100 Мб свободного дискового пространства). Этих ресурсов достаточно для размещения библиотек поддержки всех серий ПЛИС. Кроме того, при интенсивной работе может потребоваться до 300-500 Мб для хранения созданных проектов.

В настоящее время данный пакет несколько устарел и его поддержка фирмой Xilinx прекращена. Последняя из выпущенных версий - 4.2, которая существовала параллельно с первыми версиями САПР ISE. Несмотря на это, в течение какого-то времени Эта САПР будет сохранять свою актуальность. Прежде всего, только в этом пакете сохранена поддержка старых серий ПЛИС - XC3000, XC5200, XC4000, Spartan/Spartan-XL.

Кроме того, в САПР Foundation Series интегрирована система логического синтеза FPGA Express, разработанная фирмой Synopsys. Поддержка этой системы для новых САПР Xilinx прекращена. При необходимости воспользоваться данной системой также можно применять пакет Xilinx Foundation Series.

### **2. 3. САПР Integrated Software Environment**

Integrated Software Environment (ISE) - новая система автоматизированного проектирования фирмы Xilinx, заменяющая в настоящее время САПР Foundation Series. Версии 4.x САПР Xilinx включают в себя как Foundation Series, так и ISE, что позволяет разработчикам плавно перейти от одной системы проектирования к другой. В настоящее время существуют версии ISE 4.x, 5.x и 6.x.

Бесплатная версия САПР ISE WebPack доступна для загрузки с сайта Xilinx ([www.xilinx.com](http://www.xilinx.com)). Она имеет ограничение в 300 тыс. логических вентилях и не включает в себя некоторые инструментальные приложения. Интерфейс ISE Foundation аналогичен интерфейсу бесплатной версии САПР WebPack.

Пакет ISE работает под управлением операционных систем Win-

dows 2000+SP2 или Windows XP. Кроме того, возможна работа под управлением операционных систем Solaris 2.8/5.8 и Linux/WINE (Red Hat Linux 7.2 + WINE v.20010731). Для операционной системы Linux в версиях 5.x возможен только консольный режим работы, в версиях 6.x добавлена поддержка графической оболочки САПР. Требования к памяти сильно зависят от емкости проектируемых ПЛИС и изменяются от 128 Мб до 2 Гб. требования к памяти не являются безусловными и отражают уровень устойчивой работы САПР при трассировке ПЛИС с достаточно большим коэффициентом заполнения. Требования к процессору не являются критичными (достаточно производительности на уровне комфортной работы в выбранной операционной системе). Версия ISE 6.2 занимает до 1,2 Гб дискового пространства при установке поддержки всех серий ПЛИС.

Все конфигурации ISE имеют интерфейсы к САПР Synplify и Leonardo Spectrum. Версия ISE Alliance специально предназначена для интеграции со средствами синтеза сторонних производителей.

В ISE применен новый подход к трансляции исходных текстов проекта. Если для Foundation Series внутреннее представление проекта являлось схмотехническим, т. е. модули, написанные на HDL, представлялись в виде элементарных внутренних модулей ПЛИС, то в ISE используется внутреннее представление на основе HDL. Несомненным преимуществом такого подхода является резкое увеличение скорости трансляции проекта. Однако представление схемы в виде HDL-описания ставит эффективность готового проекта в зависимость от используемых в САПР алгоритмов синтеза цифровых узлов. Кроме того, при создании проекта в ISE версии 5.x необходимо выбрать используемый HDL (VHDL или Verilog), который и будет применен для внутреннего представления проекта. В версии 6.x это ограничение снято, и в рамках одного проекта можно использовать модули, описанные на любом из этих языков.

Для внутреннего представления проектируемых устройств ISE использует HDL-модули и не использует графический ввод в качестве основного средства представления проекта. В целом можно отметить,

что в подобных особенностях графического редактора определенным образом просматривается ориентация на HDL. Действительно, хотя ISE поддерживает графический ввод, его основным назначением является формирование подобия структурной или блок-схемы для анализа основных взаимосвязей проекта. Поэтому для комфортной работы в графическом редакторе рекомендуется ориентироваться на использование HDL-компонентов, а не библиотечных элементов.

Тем не менее, что, несмотря на «потерю контроля» разработчика над тонкостями реализации отдельных цифровых узлов, применение HDL в качестве внутреннего представления проекта вполне оправданно. В действительности, при работе с ПЛИС объемом в сотни тысяч эквивалентных вентилях, которые уже сейчас являются вполне доступными, на первый план выходит скорость трансляции проекта и удобство реализации типовых модулей цифровой обработки сигналов. При этом имеющиеся в настоящее время алгоритмы трансляции обеспечивают вполне приемлемую эффективность.

САПР ISE не имеет в своем составе встроенного приложения для функционального моделирования. Вместо этого в комплекте поставляется внешнее приложение ModelSim

Для включения в САПР ISE поддержки какой-либо серии требуется перейти к новой технологии проектирования и создать для этой серии ПЛИС библиотеки компонентов, алгоритмы трансляции и т. п. . Объемы ПЛИС для устаревших серий, находятся в среднем в пределах десятков тысяч эквивалентных вентилях и использование современных алгоритмов трансляции для них не приводит к такому существенному выигрышу, как, например, для семейств Spartan-II или Virtex.

### 3. ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ СХЕМ В СИСТЕМЕ MAX+PLUS II

#### 3.1. Интерфейс САПР MAX+Plus II

3.1.1. Главное окно управляющей оболочки программы и его элементы показаны на рис. 3.1.

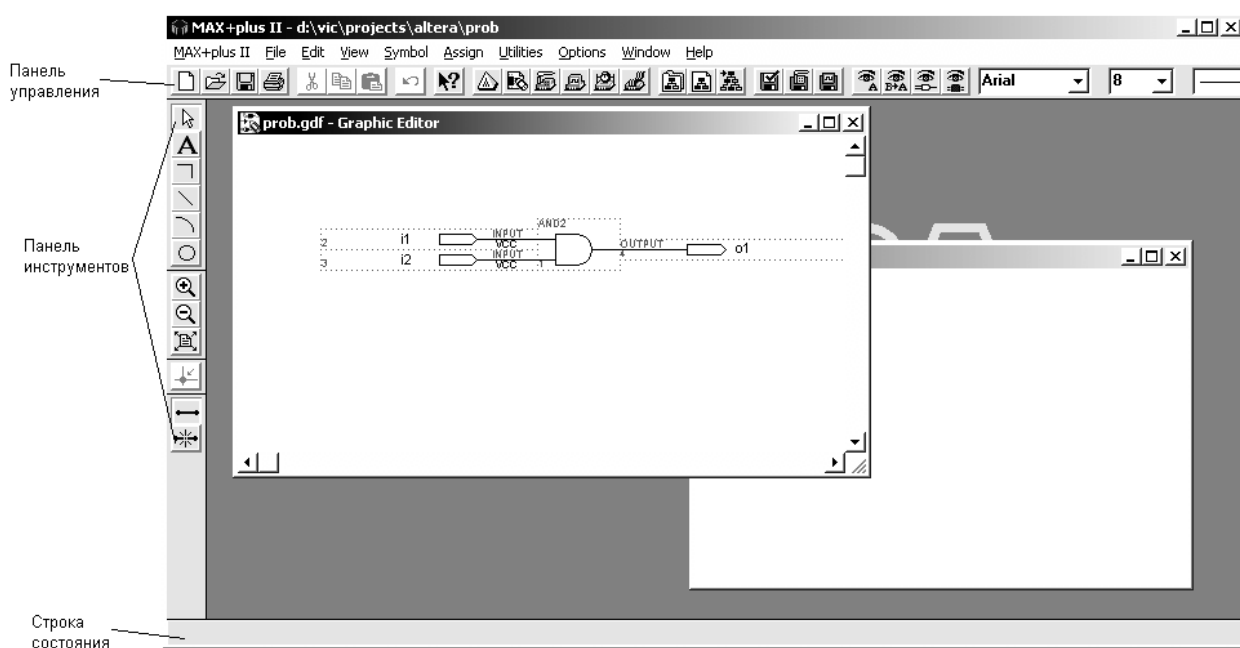


Рис.3.1

Из этого окна осуществляется конфигурация параметров системы и проекта в целом, вызов отдельных программных модулей для выполнения тех или иных технологических операций в процессе проектирования: текстового или графического редактора, компилятора, редактора сигналов, модуля моделирования (симулятора), временного анализатора и т.д. Все программные модули MAX+PLUS II содержат панель управления и строку состояния, которые можно включить или отключить, используя команду Options| Preferences (предпочтения). Панель инструментов отображает кнопки и раскрывающиеся списки, которые обеспечивают быстрый доступ к часто используемым командам. В некоторых приложениях на панели инструментов могут быть доступны дополнительные кнопки. Строка состояния представляет

краткое описание выделенной команды меню, или элемента на панели инструментов или кнопки на инструментальном наборе, когда вы перемещаете на них указатель мыши.

Вид окна, как и содержание главного меню, изменяются в зависимости от активированного программного модуля.

При запуске программы открывается окно MAX+PLUS II Manager (менеджер MAX+PLUS II). В строке заголовка отображается название программы (MAX+PLUS II), название диска и директории (например, c:\max2work\tutorial). Название текущего проекта прибавляется к названиям диска и директории. Все программные модули MAX+PLUS II содержат панель управления (Toolbar) и строку состояния (Status Bar), которые можно включать и выключать, используя команду Options| Preferences.

3.1.2. Большая часть команд MAX+PLUS II может быть вызвана различными способами: через главное меню, через кнопки панели управления, через всплывающее (контекстное) меню и с помощью нажатия определенных комбинаций клавиш на клавиатуре – “горячих клавиш”. Панели кнопок вызова команд располагаются на верхней и на левой сторонах окна. Например, нажатие кнопки Zoom In (Увеличить изображение) на панели инструментов соответствует вызову команды Zoom In. Вы можете попробовать различные способы вызова команд и определить, какие из них для вас наиболее удобны.

*Примечание. В разделе **Shortcuts** (быстрые вызовы) справочной системы MAX+PLUS II для каждого программного модуля перечислены все способы быстрого вызова: кнопки мыши, "горячие клавиши" на клавиатуре и кнопки на панели инструментов.*

Часто команды являются контекстно-зависимыми, то есть могут зависеть от положения указателя мыши или от элемента (элементов), выделенного на экране. Вызов команды с помощью левой кнопки мыши, который выполняется посредством двойного щелчка, является контекстно-зависимым. Например, в графическом редакторе можно открыть диалоговое окно Enter Symbol (Ввод символа) просто с помо-

щью двойного щелчка левой кнопки мыши на свободном пространстве окна. В отличие от этого, двойной щелчок этой кнопки на символе макрофункции откроет макрофункцию, которую представляет этот символ.

Вызов команды с помощью правой кнопки мыши выполняется путем ее щелчка для отображения на дисплее всплывающего меню и является также контекстно-зависимым. Например, можно удалить выделенный объект или участок текста с помощью щелчка правой кнопки на выделенном элементе и выбора команды Cut (вырезать) из всплывающего меню.

Возможен быстрый вызов команд с помощью клавиатуры. Например, набор на клавиатуре Ctrl+P является быстрым вызовом для команды Print (печать). Быстрые вызовы с помощью клавиатуры перечислены в справочной системе MAX-PLUS II.


3.1.3. Справочная система MAX PLUS II позволяет получать тематические, индексные и контекстно-зависимые справки.

Для получения тематической справки необходимо выбрать команду Help в главном меню и далее указать требуемый тематический раздел справки.

Поиск по индексу (команда Help| Search for Help on ) используется для быстрого поиска информации по ключевому слову или фразе.

Контекстно-зависимая справка позволяет получать информацию о тех компонентах и событиях, с которыми пользователь работает в момент обращения к справочной системе. Доступ к контекстно-зависимой справке можно получить тремя способами (табл.3.1).

## Контекстно-зависимая справка

Способ вызова	Описание
1) <b>Shift+F1</b> или кнопка контекстно-зависимой справки на панели инструментов 	Поместить указатель мыши в виде знака вопроса на графическое изображение, ключевое слово текстового файла или команду меню и щелкнуть по выбранному объекту левой кнопкой мыши для получения справки
2) Клавиша <b>F1</b>	Когда активизируется команда меню или отображается любое окно программных модулей системы MAX PLUS II, нажать клавишу F1
3) Кнопка <b>Help on Message</b> (справка о сообщении)	В окне процессора сообщений можно выделить сообщение левой кнопкой мыши и нажать кнопку <b>Help on Message</b> для получения справки об этом сообщении

## 3.2. Принципы работы с системой

Технология проектирования цифровых схем в системе MAX+PLUS II включает выполнение следующих операций:

- 1) ввод информации о логике работы проектируемой схемы;
- 2) обозначение точек входа и выхода схемы;
- 3) моделирование (симуляция) работы проектируемой схемы и анализ временных соотношений между сигналами в нем;
- 4) выбор типа микросхемы ПЛИС, на которой предполагается реализация схемы;
- 5) указание предпочтений при размещении тех или иных групп или компонентов электронной схемы в теле кристалла микросхемы;
- 6) программирование или конфигурирование ПЛИС;
- 7) тестирование ПЛИС на плате.

Система позволяет выполнить сквозное проектирование в объеме п. 1-6, а если разработчик располагает программатором или подключаемым к LPT-порту компьютера устройством ByteBlaster [10, 1], то в

объеме п. 1-7.

Все файлы проекта целесообразно размещать в одной папке. Проект может иметь иерархическую структуру, причем проекты более высоких уровней иерархии могут объединять файлы проектов более низких уровней:

1) графические (.gdf);

2) текстовые с применением разных языков программирования цифровых схем: AHDL (.tdf), VHDL (.vhd), Verilog HDL (.v), схемные файлы OrCAD (.sch); входные файлы EDIF (.edf).

При разработке иерархического проекта могут применяться две технологии: “сверху – вниз” и “снизу – вверх”.

Необходимо заметить, что система MAX+PLUS II очень чувствительна к отступлениям от технологии работы с ней, поэтому при работе с системой эту технологию необходимо выдерживать очень строго.

Рекомендуется следующий порядок проектирования.

1. Создать папку, в которой будут размещаться все файлы проекта. Имя папки и имена файлов проекта должны состоять только из символов латинского алфавита. Имя папки целесообразно выбрать таким же, как и имя главного файла проекта.

2. Создать первый файл проекта (команда File| New...), который будет являться проектом цифрового узла самого нижнего уровня иерархии. Этот файл может быть графическим (\*.gdf) или текстовым (\*.tdf, \*.vhd, \*.v, \*.sch, \*.edf). Вид файла выбирается в диалоговом окне, которое открывается после выполнения команды File| New..., с помощью графического или текстового редактора.

3. Сохранить файл в созданной ранее папке, назначить этот файл текущим файлом проекта (команда File| Project| Set project to Current File) и проверить его на отсутствие ошибок (команда File| Project| Save&Check или ее клавиатурный эквивалент Ctrl+K). Устранить ошибки.

4. Откомпилировать отредактированный файл (команда File| Project| Save & Compile, клавиатурный эквивалент Ctrl+L).



5. Создать файл входных и выходных сигналов. Этот файл создается с помощью редактора сигналов Wave Form Editor (команда MAX+PLUS II| Wave Form Editor) и сохранить его.

6. Промоделировать работу спроектированного узла (команда MAX+PLUS II| Simulator), выявить причины обнаруженных несоответствий и конфликтов и устранить их. На этом этапе удобно пользоваться командой File| Project| Save, Compile & Simulate (клавиатурный эквивалент Ctrl+Shift+K), которая совмещает операции сохранения файла, его компиляции и моделирования работы проектируемого устройства.

7. Создать макромодель для спроектированного узла (команда File| Create Default Symbol). Этой командой создается графический образ (Symbol) узла, который можно использовать в графическом редакторе наряду с библиотечными узлами в узлах более высоких уровней иерархии.

8. Далее таким же образом создаются второй и последующие файлы проекта (повторяется выполнение п.2...п.7), а завершается выполнение проекта объединением созданных проектов нижних уровней иерархии в проект самого верхнего уровня иерархии, его компиляцией, моделированием работы, проверкой временных соотношений между сигналами в контрольных точках и “прошивкой” соответствующей ПЛИС.

При работе с системой MAX+PLUS II необходимо различать понятия *текущий файл проекта*, *вспомогательный файл* и *главный файл проекта*.

*Текущий файл проекта* – это графический или текстовый файл, являющийся частью его иерархического дерева, который содержит информацию о логике работы проектируемой цифровой схемы (или ее функционального узла) и который назначен на текущий момент времени головным файлом проекта. Этот файл обрабатывается компилятором. Текущий файл проекта может быть создан с помощью графического или текстового редакторов системы MAX+PLUS II или других редакторов, поддерживающих используемые MAX+PLUS II

стандарты (*EDIF*, *VHDL* и *Verilog HDL*). Этот файл содержит исходную информацию о проектируемой схеме (или ее части) для проекта MAX+PLUS II.

Текущие файлы могут быть следующих типов: графические (*.gdf*); текстовые на языке AHDL (*.tdf*); сигнальные (*.wdf*); текстовые на языке VHDL (*.vhd*); текстовые на языке Verilog HDL (*.v*); текстовые файлы OrCAD (*.sch*); входные текстовые файлы EDIF (*.edf*); файлы формата *Xilinx Netlist* (*.xnf*); файлы проекта *Altera* (*.adf*); файлы цифрового автомата (*.smf*).

*Вспомогательные файлы* – это файлы, связанные с проектом MAX+PLUS II, но не являющиеся частью его иерархического дерева. Большинство таких файлов не содержит логики проекта. Некоторые из них создаются автоматически приложением системы MAX+PLUS II, другие – пользователем. Примерами вспомогательных файлов являются файлы назначений и конфигурации (*.acf*), символные файлы (*.sym*), файлы отчета (*.rpt*) и файлы тестовых векторов (*.vec*).

*Главный файл проекта* содержит иерархический список файлов проекта и вспомогательных файлов. Система MAX+PLUS II выполняет компиляцию, тестирование, анализ синхронизации и программирование проекта. Для каждого проекта желательно создавать отдельный подкаталог в рабочем каталоге системы MAX+PLUS II.

В иерархической структуре проекта на любом уровне допускается смешанное использование файлов типов «*.gdf .tdf .vhd .v .edf .sch*», но файлы типов «*.wdf .xnf .adf .smf*» должны либо располагаться на самом нижнем иерархическом уровне проекта, либо быть единственными.

Основой системы MAX+PLUS II является компилятор, обеспечивающий мощные средства обработки проекта, причем можно задавать нужные режимы работы компилятора. Вначале он извлекает информацию об иерархических связях между файлами проекта и проверяет проект на простые ошибки ввода, создает организационную карту проекта и затем, комбинируя все файлы проекта, превращает их в базу данных без иерархии, которую можно эффективно обрабаты-

вать. Если проект слишком большой, чтобы быть реализованным в одной ПЛИС, компилятор может автоматически разбить его на части для реализации в нескольких устройствах того же самого семейства ПЛИС, при этом минимизируется число соединений между устройствами. Существует возможность задать обработку проекта в соответствии с указаниями разработчика. Например, можно задать стиль логического синтеза проекта и другие параметры: временные требования в рамках всего проекта, разбить большой проект на части для реализации в нескольких устройствах и выбрать варианты параметров устройств, которые будут применены для всего проекта, назначить номера выводов ПЛИС для тех или иных сигналов и т.д. Эти операции доступны во всех приложениях MAX+PLUS II с помощью команд из меню Assign.

Ниже будет рассмотрена технология “снизу – вверх” на примере построения проекта четырехразрядного двоичного счетчика с преобразователем двоично-десятичного кода в код для семисегментного индикатора. Вначале с помощью графического и текстового редакторов будут созданы проекты *counter\_10.gdf*, *counter\_10\_txt.vhd* и *decoder.vhd* нижнего уровня иерархии, затем на основе этих проектов с помощью графического редактора будет создан проект верхнего уровня *led.gdf*.

### **3.3. Разработка проекта нижнего уровня с помощью графического редактора**

**3.2.1. Начало работы над проектом.** Прежде чем начинать новый проект средствами операционной системы или каким-либо файловым менеджером (например, Windows Commander, Far и пр.), необходимо создать папку для хранения всех файлов проекта. Предположим, что такая папка создана и названа по имени проекта *led*.

Для создания проекта с помощью графического редактора системе MAX+PLUS II необходимо выполнить следующую последова-

тельность действий.

1) Создать новый файл (в рассматриваемом примере – *counter\_10.gdf*), который будет главным файлом этого уровня иерархии проекта, и назначить его текущим файлом проекта.

2) Ввести изображение проектируемой схемы с помощью графического редактора MAX+PLUS II и проверить файл схемы на отсутствие формальных (синтаксических) ошибок.

3) Создать графическое изображение (Symbol) файла текущего проекта, для использования его в проектах верхних уровней иерархии.

**3.2.2. Создание графического файла проекта.** Чтобы создать новый графический файл проекта, можно поступить следующим образом.

Введите команду File|New, затем в диалоговом окне New выберите опцию Graphic Editor file (файл графического редактора) и расширение имени файла *.gdf* в раскрывающемся списке. В результате этих действий откроется окно графического редактора (рис. 3.2), файлу которого будет присвоено имя *Untitled1.gdf*. Сохраните этот файл в созданной ранее папке ...\\led (команда File|Save as) под именем *counter\_10.gdf*.

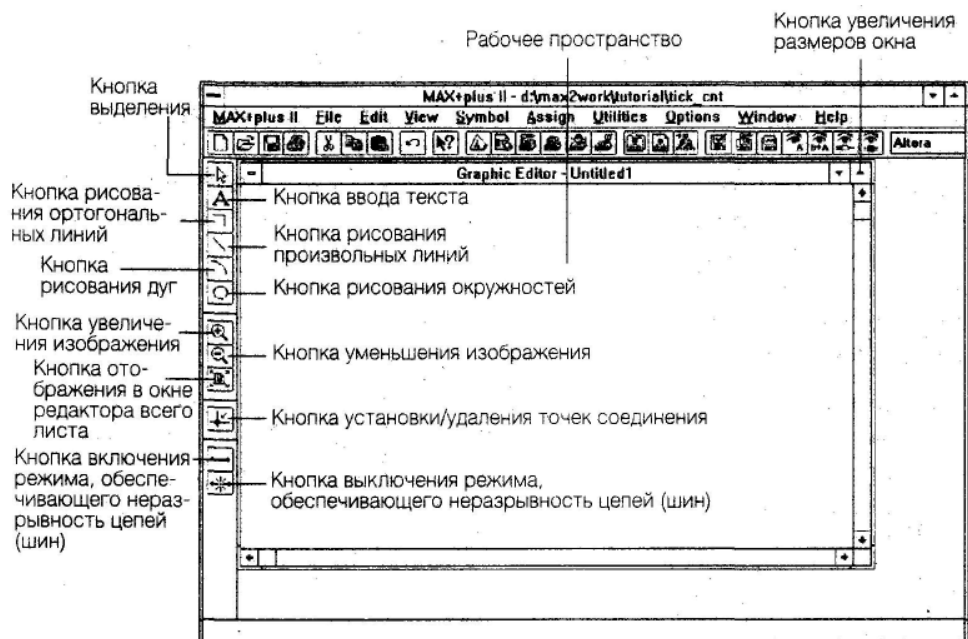


Рис. 3.2

**3.2.3. Назначение файла текущим файлом проекта.** Для компиляции или любой другой пакетной обработки (например, моделирования) файла цифровой схемы в системе MAX+PLUS II необходимо назначить этот файл в качестве главного файла текущего проекта. Такое назначение удобно делать при создании файла схемы сразу после его сохранения.

Введите команду File |Project|Name... . В диалоговом окне Project Name (рис. 3.3), если необходимо, выключите опцию Show Only Tops of Hierarchies (показывать только верхний уровень иерархии), выделите *counter\_10.gdf* в поле File и нажмите кнопку ОК. Строка заголовка MAX+PLUS II Manager изменится, отображая название текущего проекта: MAX+PLUS II Manager - ...\\led\\counter\_10.

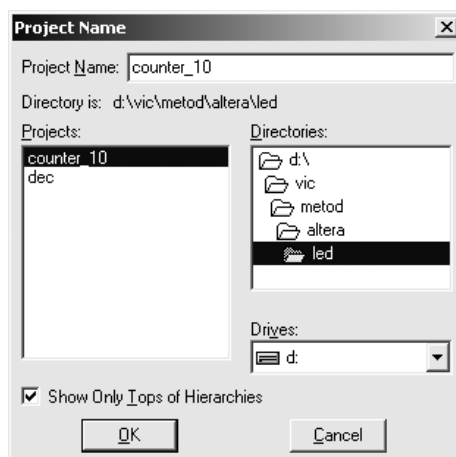


Рис. 3.3

Альтернативой команде File |Project|Name... является команда File |Project |Set Project to Current File (назначить имя проекта по имени текущего файла), когда открыт соответствующий файл (в нашем случае *counter\_10.gdf*) в активном окне редактора.

**3.2.4. Ввод изображения проектируемой схемы** предполагает выполнение следующих технологических операций:

- 1) ввод символов компонентов схемы;
- 2) ввод входных и выходных контактов;
- 3) назначение имен контактов;
- 4) соединение символов функциональных модулей;
- 5) соединение цепей и шин посредством имен;

б) проверка файла на наличие формальных (синтаксических) ошибок.

Последовательность выполнения первых четырех операций не имеет значения, пятая и шестая операции являются завершающими.

Ввод символов компонентов схемы предполагает наличие этих компонентов либо во встроенных библиотеках системы MAX+PLUS II, либо в файлах проектов более низких уровней иерархии. Библиотеки MAX+PLUS II содержат большое количество компонентов и функциональных модулей (базовых элементов, мегафункций и макрофункций), изображения которых можно использовать в файлах графического редактора. Перечень этих библиотек выводится в поле Symbol Libraries диалогового окна Enter Symbol (рис. 3.4). Щелкнув два раза правой кнопкой мыши по имени соответствующей библиотеки, можно вывести ее содержание в поле Symbol Files.

Для ввода символа требуемого компонента щелкните два раза левой кнопкой мыши в том месте окна графического редактора, где должен располагаться этот компонент. Затем в поле Symbol Name окна Enter Symbol наберите название требуемого компонента (если вы его знаете), либо найдите его в поле Symbol Files, предварительно раскрыв содержание соответствующей библиотеки двойным щелчком левой кнопки мыши по ее имени в поле Symbol Libraries.

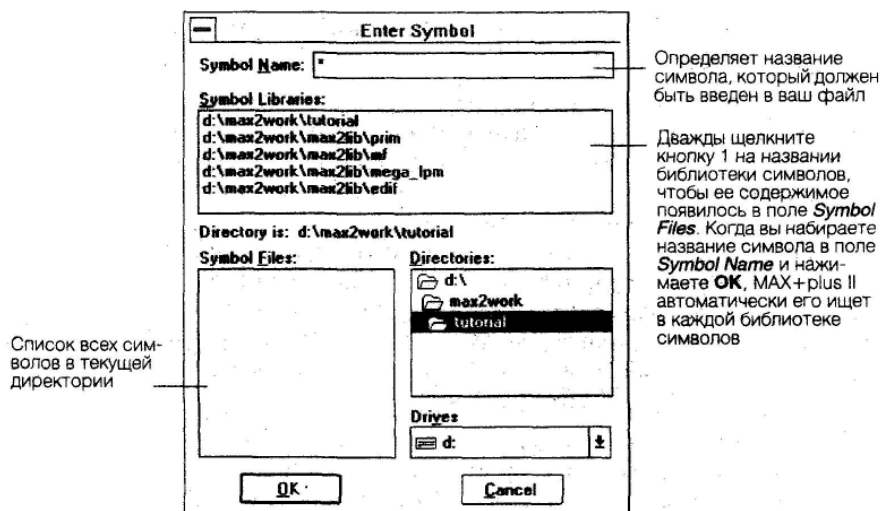


Рис. 3.4

Справку по мегафункциям, базовым элементам и макрофункциям можно получить путем выбора строк Megafunction/LPM, Primitives или Old-Style Macrofunctions соответственно из меню Help. Щелкнув левой кнопкой мыши по символу компонента можно получить контекстно-зависимую справку об этом компоненте. Такая справка существует для всех базовых элементов, мегафункций и макрофункций, разработанных фирмой «Altera» в системе MAX+PLUS II.

Примеры способов ввода символов компонентов в файл текущего проекта *counter\_10.gdf*.

1. Дважды щелкните левой кнопкой мыши по свободному пространству окна графического редактора. Такое действие одновременно определяет точку вставки и открывает диалоговое окно Enter Symbol (вставить символ).

2. Наберите TFF (макрофункция TFF представляет собой модель синхронного Т-триггера) в поле Symbol Name.

3. Нажмите кнопку ОК. В окне графического редактора появится символ TFF, причем его верхний левый угол попадает в точку вставки.

4. Снова сделайте двойной щелчок левой кнопкой мыши по свободному пространству окна графического редактора и нажмите кнопку ОК в окне Enter Symbol – в месте щелчка появится еще один символ TFF.

5. Скопируйте символ TFF в буфер обмена (Ctrl+C), щелкните левой кнопкой мыши по свободному месту окна графического редактора и вставьте на это место символ TFF из буфера обмена (Ctrl+V). Для ввода однотипных компонентов не обязательно пользоваться технологией копирования через буфер обмена. Достаточно выделить требуемый компонент, щелкнув по нему левой кнопкой мыши, а затем, нажав клавишу Ctrl и захватив выделенный компонент левой кнопкой мыши, можно переместить копию выделенного компонента в нужное место.

6. Введите элементы 2И (and2), 3И (and3), 4И (and4), 6И

(and6): дважды щелкните левой кнопкой мыши по свободному пространству окна графического редактора и откройте диалоговое окно Enter Symbol. В поле Symbol Libraries сделайте двойной щелчок левой кнопкой мыши по строке `c:\maxplus2\max2lib\prim` и в поле Symbol Files сделайте двойной щелчок левой кнопкой мыши по соответствующей строке (and2, and3 и т.д.). В окне графического редактора появится изображение требуемого элемента. Аналогично введите три элемента НЕ (not).

Размещенный на поле графического редактора символ компонента можно перемещать, вращать и отражать, предварительно выделив его. Для перемещения символа компонента его достаточно протащить на требуемое место мышью, нажав левую кнопку. Если символ компонента необходимо повернуть или зеркально отразить относительно одной вертикальной или горизонтальной оси, щелкните по нему правой кнопкой мыши и выберите из контекстного меню требуемое действие:

Flip Horizontal – отразить по горизонтали,

Flip Vertical – отразить по вертикали,

Rotate – повернуть.

Когда вы вводите или перемещаете два символа так, что их границы и контакты касаются, то символы становятся логически связанными. Если в дальнейшем вы перемещаете один из этих символов при включенном режиме Rubberbanding (он обеспечивает неразрывность цепей и шин при перемещении компонентов схемы – раздел Options главного меню), то между соединенными контактами двух символов автоматически образуется линия нового проводника или шины.

При вводе символа любого компонента ему автоматически присваивается идентификационный номер символа (Symbol ID), который располагается в его левом нижнем углу. Он соответствует порядковому номеру ввода символов (первому введенному символу присваивается номер 1, второму – 2 и т.д.). Идентификационный номер однозначно определяет каждую копию символа в gdf- файле.

Для улучшения читаемости схем символы компонентов рекоменду-



ется располагать вдоль горизонтальных и вертикальных линий разметки. Вы можете задать расстояния между линиями разметки и отобразить или скрыть их на экране.

Чтобы задать расстояние между линиями разметки, введите команду Options| Guideline Spacing (расстояние между линиями разметки) и в диалоговом окне Guideline Spacing в полях *X (Horizontal) Spacing* и *Y (Vertical) Spacing* (задание расстояний между горизонтальными и вертикальными линиями разметки соответственно) укажите требуемые расстояния (например, 15 пикселей).

Для отображения линий разметки введите команду Options| Show Guidelines (показать линии разметки).

*Ввод входных и выходных контактов.* Входные и выходные контакты – это компоненты INPUT и OUTPUT и они вводятся таким же образом, как и любые другие компоненты через диалоговое окно Enter Symbol. Это окно, как уже упоминалось выше, вызывается двойным щелчком левой кнопки мыши по свободному полю окна графического окна. Для ввода контактов наберите input в поле *Symbol Name*, и нажмите **ОК**. На экране появится компонент INPUT. Аналогично введите компонент OUTPUT. Нажмите клавишу Ctrl на клавиатуре и затем левую кнопку мыши на компоненте OUTPUT. Удерживая нажатыми Ctrl и левую кнопку мыши, переместите указатель мыши вниз для создания копии компонента и поместите эту копию под оригиналом (символ копируется, но не помещается в буфер обмена).

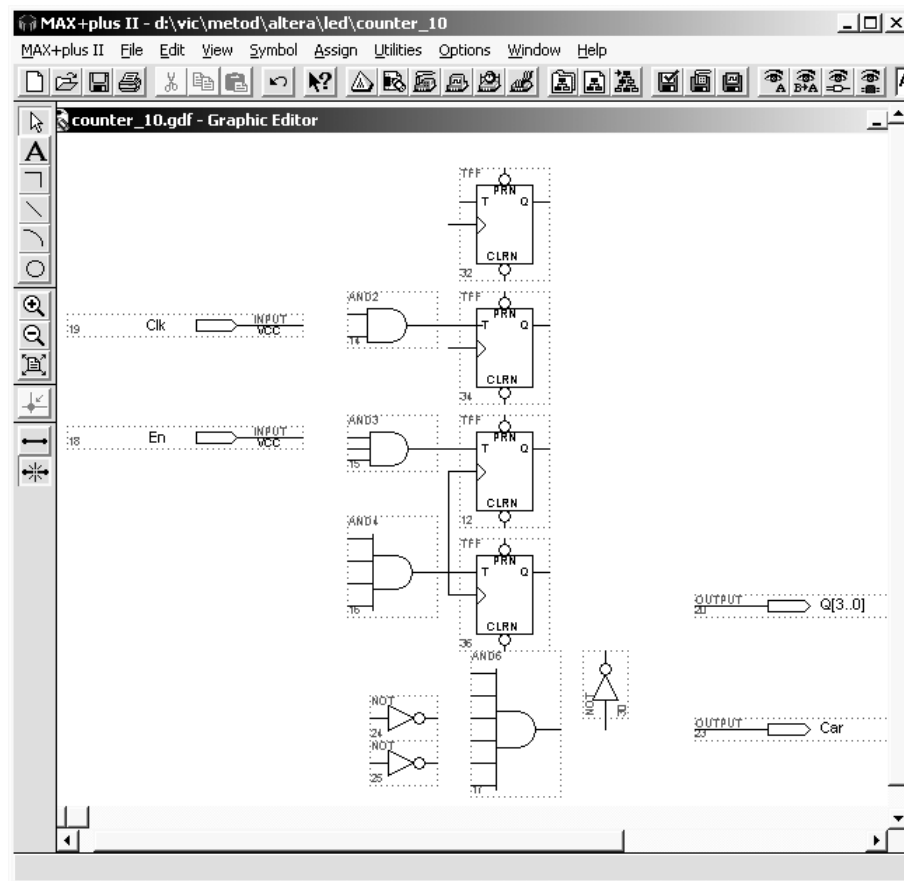


Рис. 3.5


*Назначение имен входных и выходных контактов.* При вводе контактов (как и любых других компонентов) система автоматически присваивает им идентификационные номера в соответствии с порядком ввода компонента и стандартные имена "PIN\_NAME". Для назначения имени контакта переместите указатель мыши на имя контакта "PIN\_NAME" и дважды щелкните левой кнопкой мыши для его выделения. Затем наберите новое имя и нажмите клавишу Enter, либо щелкните левой кнопкой мыши по свободному полю графического редактора.

Обозначьте входные контакты как Clk и En, а выходные как Q[3..0] и Car.

*Соединение выводов компонентов.* Электрические цепи для соединения выводов компонентов друг с другом могут быть введены одним из трех следующих способов:

- 1) с помощью изображений одиночных проводников;

- 2) с помощью изображений группы проводников (шин);
- 3) посредством одинаковых имен.

Для соединения выводов компонентов с помощью изображений проводников достаточно перейти в режим Select в файле графического редактора (выбрать инструмент  или нажать клавишу Esc), установить указатель мыши на нужный вывод и, нажав левую кнопку мыши, провести линию, соединяющую его с другим выводом.

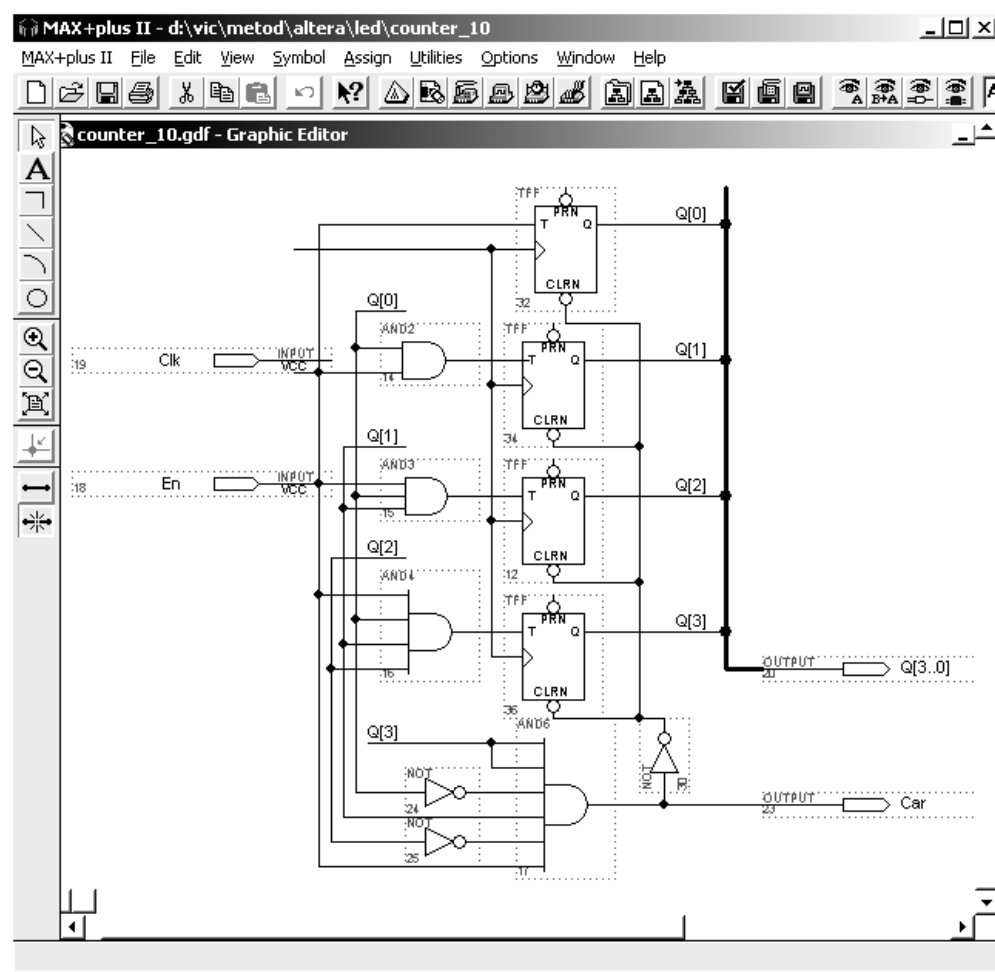




Рис. 3.6

Указатель режима Select  автоматически принимает форму указателя рисования ортогональных линий ( + ), когда он находится над выводом компонента или над концом линии. Выводы компонентов можно соединять ортогональными линиями и линиями с произвольным углом наклона, выбрав инструмент  на панели инструментов с левой стороны окна графического редактора. Одиночные проводни-



ки проводят тонкими линиями, шины – толстыми. Тип линии можно выбрать из контекстного меню, для вызова которого необходимо щелкнуть правой кнопкой мыши по полю графического редактора, а затем выбрать в меню пункт Line Style. Этой же командой можно изменить тип ранее проведенной линии, предварительно выделив её. С помощью инструмента для рисования ортогональных линий можно рисовать прямые линии или линии с одним изгибом. Если необходимо провести линию с большим количеством изгибов, ее строят из нескольких линий с одним изгибом. Линии соединяются автоматически, если они одного и того же типа. Для удаления линии выделите её и нажмите клавишу Del или Backspace.

Чтобы соединить цепи или контакты посредством одинаковых имен, выделите первую цепь щелчком левой кнопки мыши в режиме Select, введите имя цепи и нажмите Enter. Затем повторите эту процедуру со второй цепью или контактом, введя то же имя – объекты окажутся электрически соединенными, хотя это и не будет отображаться в форме линий. Примеры таких соединений показаны на рис. 3.6 (цепи Q[0], Q[1], Q[2], Q[3]).

**3.2.5. Проверка и сохранение файла.** Чтобы убедиться, что ввод файла логически корректен, его необходимо сохранить и проверить его на наличие формальных (синтаксических) ошибок. Для этого выполните команду File |Project Save & Check (сохранить и проверить проект). Созданный файл сохраняется и открывается окно компилятора MAX+PLUS II. Модуль Compiler Netlist Extractor (экстрактор списка соединений компилятора) проверит файл на ошибки, обновит дерево иерархии проекта и выведет на экран сообщение, показывающее число ошибок и предупреждений. Исправьте ошибки, если они допущены, и повторите команду File |Project Save & Check.

*Если компилятор выдает какие-либо сообщения об ошибках или предупреждения, а окно процессора сообщений не выводится, то вы можете открыть его командой MAX+PLUS II |Message Processor (процессор сообщений). Выделите сообщение в окне процессора сообщений и нажмите кнопку Locate (найти местоположение) для нахождения ис-*

точника (источников) сообщения или нажмите кнопку Help on message (справка на сообщение) для объяснения ошибки.

**3.2.6. Создание графического изображения (символа) для файла текущего проекта.** Теперь можно создать графическое изображение файла текущего проекта Symbol File (символ файла), которое может быть использовано в любом другом графическом файле проекта (Graphic Design File) более высокого уровня иерархии для замещения созданного проекта. Это изображение хранится в файле с расширением (.sym). Выберите команду File |Create Default Symbol (создать стандартный символ текущего файла) из меню. Если символ для файла уже существует, то MAX+PLUS II спросит, нужно ли переписать существующий символ. Нажмите кнопку ОК, чтобы полностью обновить информацию в символьном файле. Созданный стандартный символ может быть отредактирован с помощью редактора Symbol Editor (команда MAX+PLUS II |Symbol Editor).

**3.2.7. Проверка логики работы устройства, созданного в текущем проекте.** Эта операция включает выполнение следующих процедур:

- 1) указание типа ПЛИС, на которой предполагается реализовать проект;
- 2) компиляция проекта;
- 3) создание файла временных диаграмм (scf- файл) для записи входных и выходных сигналов при моделировании;
- 4) моделирование работы устройства и анализ результатов.

Чтобы указать тип ПЛИС, на которой предполагается реализовать проект, выполните команду Assign |Device... и в открывшемся диалоговом окне Device выберите из списка семейство (Device Family) ПЛИС и тип Devices.

Для компиляции проекта выполните команду MAX+PLUS II |Compiler, затем нажмите в диалоговом окне компилятора кнопку Start и дождитесь вывода на экран информационного окна MAX+PLUS II - Compiler с информацией о результатах компиляции. В результате компиляции будет создан файла списка соединений для симулятора (Simu-

lator Netlist File – snf-файл). На начальных этапах компиляции удобно пользоваться командой синтаксического контроля схемы File|Project|Save and Check (Ctrl+L).

*Файл временных диаграмм* (scf-файл) может отражать все или некоторые сигналы в цепях из файла списка соединений для симулятора (snf-файла). Этот scf- файл может редактироваться, чтобы задать входные сигналы для моделирования.

Чтобы создать файл временных диаграмм введите команду MAX+PLUS II |Waveform Editor либо команду File| New, выберите опцию *Waveform Editor file* (файл для редактора временных диаграмм), выделите расширение .scf в окошке раскрывающегося списка и нажмите **ОК** для создания нового файла без названия.

Введите команду File|End Time (время окончания моделирования) и наберите 20us. Это время определяет, когда симулятор прекратит использование входных векторов в процессе моделирования. Затем укажите шаг сетки для редактора временных диаграмм (команда Options| Grid Size), например 250 ns, и нажмите **ОК**.

Щелкните правой кнопкой мыши на поле окна редактора либо введите команду Node| Enter Nodes from SNF (ввод цепей из snf - файла). На экране отобразится диалоговое окно Enter Nodes from SNF (рис. 3.7).

Выключите опцию *Group* (группа) под заголовком *Type* (тип). Опции *Inputs* (входы) и *Outputs* (выходы) должны остаться включенными. Нажмите кнопку **List** (сформировать список) для получения списка доступных входных (I) и выходных (O) цепей. Нажмите кнопку ( $\Rightarrow$ ) для копирования выделенных цепей в окошко Selected Nodes & Groups (выбранные цепи и группы). Далее нажмите кнопку **ОК**.

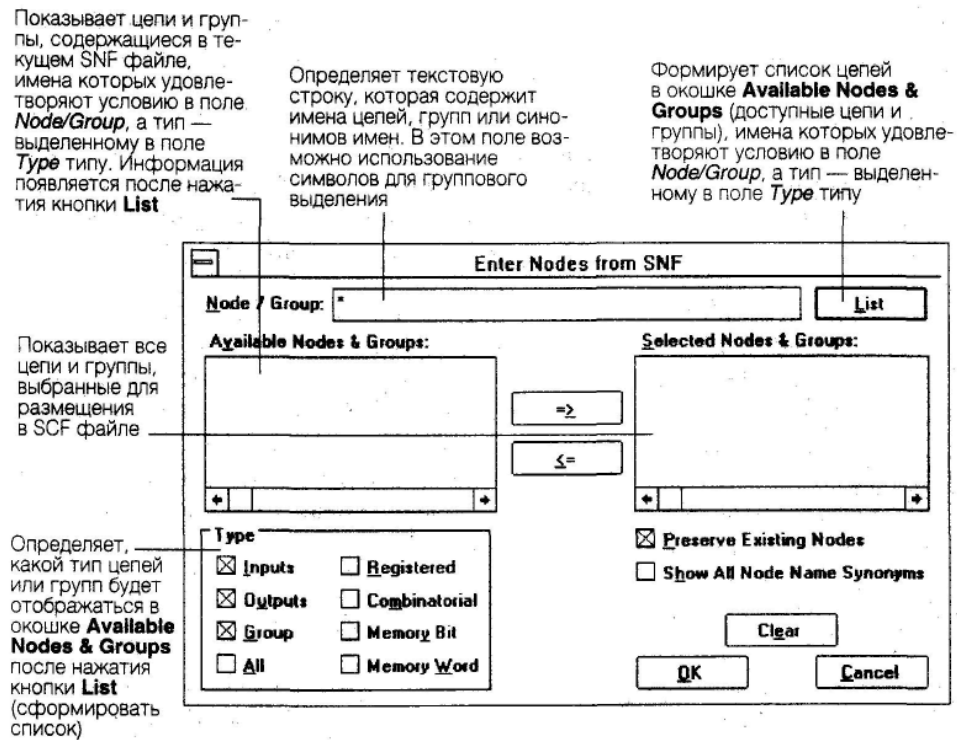


Рис. 3.7

Редактор временных диаграмм добавит выбранные цепи и группы в scf- файл без названия. Все временные диаграммы для входных цепей, но умолчанию имеют низкий (0) логический уровень, и все временные диаграммы для выходных и внутренних цепей по умолчанию имеют неопределенный (X) логический уровень, как показано на рис. 3.8. Сохраните файл (команда File| Save as) под названием *counter\_10.scf*.

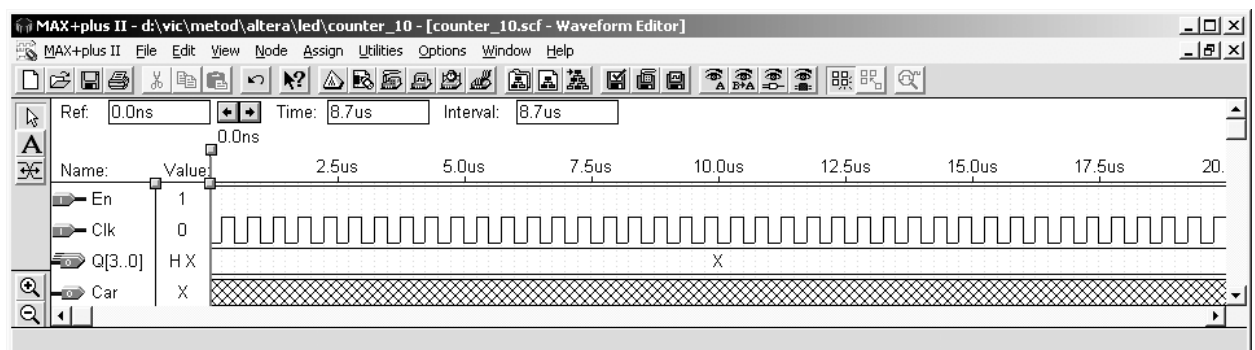


Рис. 3.8

**3.2.8. Редактирование временных диаграмм для входных цепей.** Чтобы задать исходные данные для моделирования, необходимо отредактировать временные диаграммы входных цепей. В процессе модели-

рования симулятор автоматически переписывает неопределенные логические уровни внутренних и выходных цепей на значения, которые получаются в результате обработки входных сигналов.

Для редактирования временных диаграмм, прежде всего, установите такой временной масштаб, чтобы на экране отображался весь интервал моделирования (в рассматриваемом примере 10 us). Это можно сделать командой View|Time Range, либо многократным выполнением команды View|Zoom Out (клавиатурный эквивалент Ctrl+Shift+Space), либо с помощью кнопки “минус” на панели инструментов. Далее левой кнопкой мыши выделите на поле временной диаграммы для требуемого сигнала нужный временной интервал и с помощью команд Edit|Overwrite|..., Edit|Insert|..., Edit|Copy|, Edit|Delete и т.д. изобразите требуемые временные диаграммы сигналов. Для удобства работы часть указанных команд дублируется кнопками на панели инструментов.

Процедура редактирования файла временных диаграмм в рассматриваемом примере может быть организована следующим образом. Щелкните левой кнопкой мыши по полю *Value* для цепи enable и введите команду Edit|Overwrite|High (1) (задание высокого уровня), чтобы задать высокий логический уровень на всем интервале моделирования для этой цепи. Данный сигнал разрешает работу преобразователя кода. Альтернативным вариантом ввода указанной команды является ее вызов с помощью всплывающего меню при нажатии правой кнопки мыши после выделения соответствующего участка временной диаграммы.

Для задания тактовых импульсов (сигнал clock) с периодом 0,5 us (шаг сетки 0,25 us) выделите всю временную диаграмму clock щелчком левой кнопки мыши на поле *Value* и введите команду Edit|Overwrite Clock. На экране отобразится диалоговое окно Overwrite Clock. Нажмите **ОК** для принятия параметров по умолчанию.

Сохраните и закройте файл временных диаграмм (команда File|Save). Графическое изображение временных диаграмм входных сигналов показано на рис. 3.9.



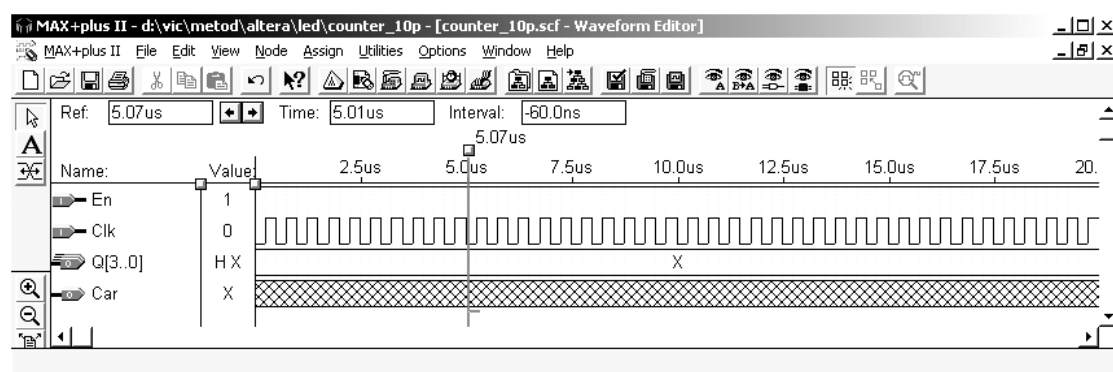



Рис. 3.9

**3.2.9. Моделирование текущего проекта.** Программа моделирования работы спроектированного цифрового узла использует Simulator Channel File (файл временных диаграмм) с расширением (.scf) или Vector File (векторный файл) с расширением (.vec) в качестве источника входных сигналов для моделирования. Для вызова программы моделирования необходимо ввести команду File|Project|Save & Simulate (клавиатурный вариант Ctrl+Shift+L), либо команду MAX+PLUS II|Simulator, и после ее выполнения открыть файл временных диаграмм, в который выводятся результаты моделирования. Этот файл можно открыть либо из диалогового окна **Simulator: Timing Simulation**, которое автоматически выводится после окончания моделирования, нажав кнопку **Open SCF**, либо командой File|Open главного меню, либо дважды щелкнув по значку SCF в иерархическом дереве текущего проекта.

Изображение иерархического дерева можно вызвать командой MAX+PLUS II|Hierarchy Display главного меню, либо нажав кнопку  (“пирамидка”) на панели управления. Изображение иерархического дерева удобно использовать для контроля текущего проекта и навигации по его файлам.

Результаты моделирования работы счетчика представлены на рис. 3.10.

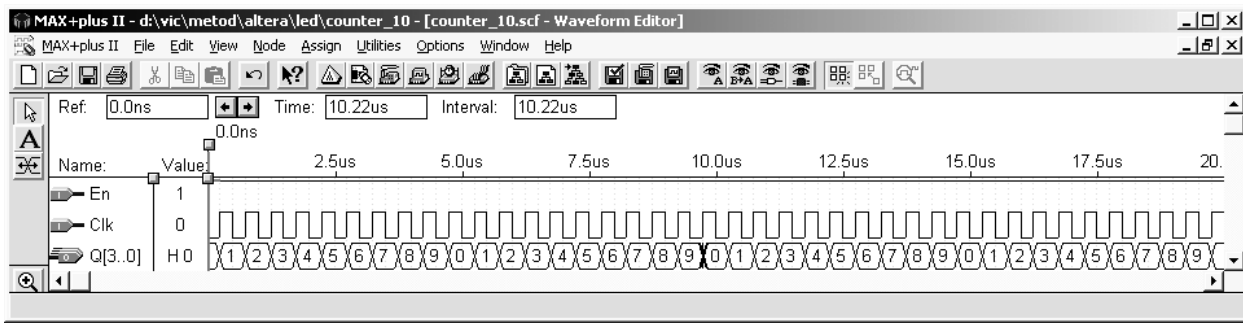


Рис. 3.10

Сигнал Q[3..0] отображает в шестнадцатеричном коде группу сигналов Q[3], Q[2], Q[1], Q[0] на соответствующих выходах счетчика. Для более наглядного изображения временных диаграмм можно развернуть сигналы шины Q[3..0] в четыре отдельных сигнала Q0, Q1, Q2, Q3 (рис. 3.11). Они соответствуют выходам счетчика Q[0], Q[1], Q[2], Q[3]. Чтобы сделать это, необходимо щелкнуть правой кнопкой мыши по сигналу Q[3..0] (рис. 3.10) и в открывшемся всплывающем меню выбрать команду Ungroup. Возможна и обратная процедура: выделив левой кнопкой мыши при нажатой клавише Shift группу сигналов и введя из главного меню команду Node|Enter Group...(либо Enter Group...из всплывающего меню), можно объединить эту группу в одну шину, значения сигналов на которой будут представлены в шестнадцатеричном коде.

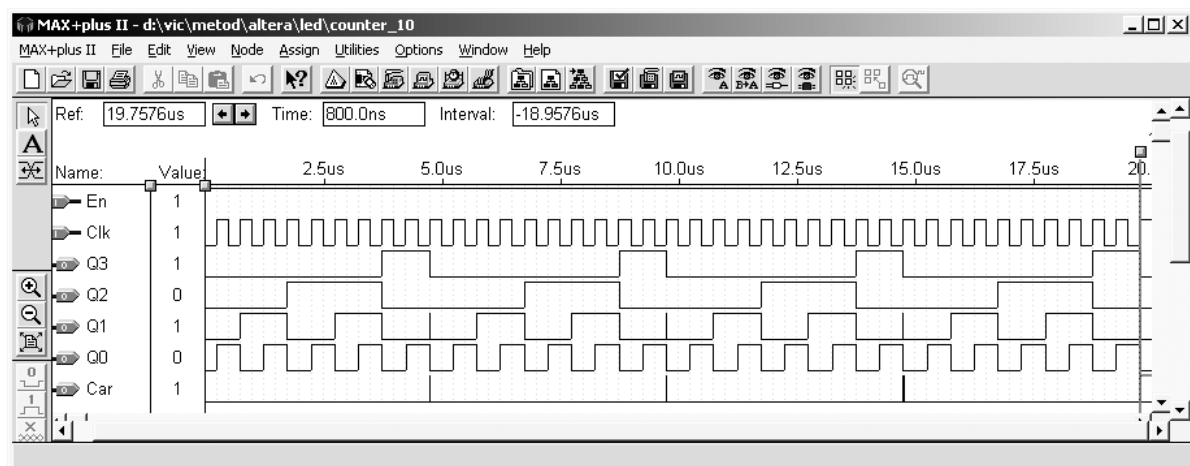


Рис. 3.11

Для анализа временных диаграмм удобно пользоваться времен-

ным маркером, который показывает время изменения логического уровня любого сигнала scf- файла. Для перемещения временного маркера от одного момента времени к другому используются кнопки ► и ◀, расположенные между окошками Ref и Time в scf- файле (рис. 3.10). Временное положение маркера отображается в окошке Ref (Reference).

Если по результатам моделирования появилась необходимость откорректировать исходный файл проекта и оперативно проверить влияние результатов корректировки на работу узла, то после корректировки удобно пользоваться командой File| Project| Save, Compile & Simulate (клавиатурный вариант Ctrl+Shift+K), объединяющей три операции: сохранение откорректированного файла, его компиляцию и моделирование.

**3.2.10. *Заккрытие файла текущего проекта*** является заключительной операцией при создании проекта любого уровня иерархии. Эта операция необходима, если вы хотите начать работу над другим текущим проектом. Заккрыть файл можно командой File|Close (клавиатурный вариант Ctrl+F4), либо щелкнув по значку “х” в правом верхнем углу окна головного файла текущего проекта (в данном случае counter\_10.gdf).

### **3.3. Разработка проектов цифровых узлов на основе HDL -языков**

**3.3.1. *Вводные замечания.*** Основные операции проектирования цифровых схем с помощью языков описания цифровых устройств для проектов нижнего уровня иерархии в системе MAX+PLUS II совпадают с аналогичными технологическими операциями при выполнении графического проекта. Отличие заключается лишь в технологии создания и отладки файла, описывающего логику работы цифровой схемы. Преимущества проектирования цифровых схем с помощью языков описания цифровых устройств (HDL – Hardware Description Language) в сравнении с традиционным “схемотехническим” способом заключаются в резком снижении трудоемкости с увеличением

сложности схемы и снижении требований к уровню схемотехнической подготовки проектировщика. Дополнительным преимуществом является возможность определенного абстрагирования от вариантов схемотехнической реализации цифрового устройства на начальных этапах проектирования и концентрации внимания лишь на логике его работы.

Как упоминалось выше, система MAX+PLUS II позволяет использовать ряд различных языков: VHDL (.vhd); AHDL (.tdf); Verilog HDL (.v) [2]. Технология проектирования одинакова для всех указанных языков описания цифровых устройств. Рассмотрим ее на примерах проектирования описанного выше десятичного счетчика и декодера, преобразующего двоично-десятичный код с выхода четырехразрядного счетчика в семиразрядный код для семисегментного индикатора. В качестве языка проектирования выберем VHDL [4, 5, 7].

*Примечание. В системе MAX+PLUS II реализовано одно из подмножеств стандарта VHDL-93. Вследствие этого действуют ограничения на применение некоторых конструкций и операторов.*

**3.3.2. Проект нижнего уровня иерархии** на языке VHDL выполняется в следующей последовательности.

1. Создается текстовый файл с описанием проектируемого цифрового устройства на языке VHDL.

2. Созданный текстовый файл назначается головным файлом проекта, выбирается тип ПЛИС, на которой предполагается реализовать проектируемую цифровую схему, затем проводится синтаксический контроль файла, устраняются синтаксические ошибки, и выполняется компиляция данного файла.

3. Осуществляется моделирование работы схемы во временной области, анализ временных диаграмм сигналов и вносятся, в случае необходимости, корректировки в текст файла.

4. Создается графический образ цифрового узла для включения его в графические проекты более высоких уровней иерархии.

5. В VHDL-файле делаются необходимые комментарии и все

файлы проекта сохраняются.

Технология выполнения п. 2-5 полностью совпадает с рассмотренной выше технологией проектирования цифровых узлов на основе графического файла, поэтому ниже основное внимание будет уделено методике выполнения п.1.

**3.3.3. Пример проектирования десятичного счетчика на языке VHDL.** Спроектируем на VHDL рассмотренный выше десятичный счетчик.

Создаем новый файл (команда File|New) и в открывшемся диалоговом окне выбираем опцию Text Editor File, затем сохраняем пока еще пустой файл в созданной ранее папке .../Led под именем Counter\_10\_txt.vhd (команда File| Save As...).

Вводим информацию о проектируемой схеме на языке VHDL. Текст программы, описывающей работу проектируемого счетчика с комментариями, представлен на рис. 3.12.

```

use IEEE.std_logic_unsigned.all;

-- Объявление интерфейса программы: описание входных и выходных сигналов счетчика
ENTITY Counter_10_Txt IS -- Counter_10_Txt - название цифрового узла (оно должно
    -- совпадать с именем файла)
    PORT(
        Clk : in STD_LOGIC;           -- сигнал синхронизации
        En  : in STD_LOGIC;           -- сигнал разрешения счета
        Q   : inout STD_LOGIC_VECTOR (3 downto 0); -- выходной сигнал
        Car : out STD_LOGIC);         -- сигнал переполнения (сигнал переноса)
    END entity Counter_10_txt ;

-- описание варианта реализации счетчика
ARCHITECTURE aaa OF Counter_10_txt IS
BEGIN
    process(clk) -- активизация программы по изменению сигнала clk
    BEGIN
        if (Clk'event and Clk='1') then -- обнаружение переднего фронта синхронимпульса
            if (En='1') then -- проверка разрешения счета
                if (Q="1010") then Q<="0000"; Car<='1'; -- проверка переполнения счетчика и
                -- формирование сигнала переноса
            else Car<='0'; Q<=Q+1; end if; -- увеличение содержимого счетчика на 1
            end if;
        end if;
    end process;
end ARCHITECTURE aaa;

```

Рис. 3.12

Для ускорения ввода текста программы можно воспользоваться шаблонами конструкций языка VHDL. Команда Template| VHDL

Template выводит на экран окно со списком шаблонов на конструкции языка VHDL. Щелчком левой кнопки мыши по требуемому названию шаблона можно вывести его в то место экрана, где находится курсор. Для облегчения синтаксического контроля текста программы можно включить его синтаксическую раскраску (команда Options| Syntax Coloring).

Сохраняем созданный файл (Ctrl+S) и назначаем его головным файлом текущего проекта (команда File| Project| Set Project to Current File).

Проводим пробную компиляцию проекта (команда Ctrl+K или File| Project| Save & Check), устраняем синтаксические ошибки. Эти шаги повторяются до тех пор, пока все ошибки не будут устранены. При устранении синтаксических ошибок необходимо учитывать особенности построения системы диагностики ошибок MAX+PLUS II: устранение ошибок следует начинать с первой по порядку ошибки, выводимой в диалоговом окне компилятора с сообщениями об ошибках. Большинство остальных ошибок оказываются “наведенными” — они исчезают при устранении первой по порядку ошибки. Для нахождения места первой ошибки в тексте программы достаточно дважды щелкнуть левой кнопкой мыши по строке с информацией об ошибке в диалоговом окне компилятора.

После устранения всех ошибок проводим полную компиляцию проекта (команда Ctrl+L или File| Project| Save & Compile), создаем графический символ спроектированного цифрового узла (команда File| Create Default Symbol) и проводим моделирование его работы во временной области таким же образом, как это делалось выше для графического проекта (см. п.п. 3.2.7 – 3.2.9). Особенностью данного проекта является необходимость удалить из файла временных диаграмм (scf -файла) перед проведением моделирования сигналов Q0–Q3, помеченных знаком (I) – *input*, если вы перенесли их в этот файл из snf -файла при формировании списка временных диаграмм. Если этого не сделать, то будет нарушена нормальная работа программы моделирования. Необходимость выполнения этой операции вызвана

тем, что сигнальная шина Q объявлена в интерфейсе проектируемого узла как inout (двунаправленная, см. комментарий ниже в п. 3.3.4).

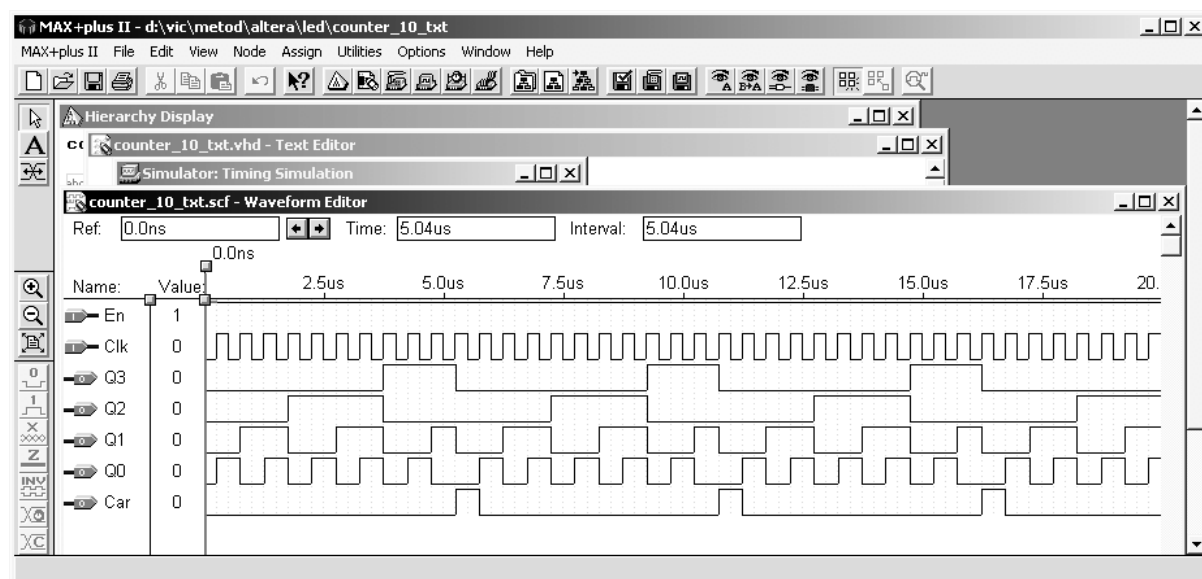


Рис. 3.13

Результаты моделирования работы счетчика представлены на рис. 3.13.

3.3.4. *Программа на языке VHDL* (рис. 3.12) содержит три раздела:

1) раздел подключения библиотек (начало – ключевое слово `library`) – в нем перечисляются используемые программой библиотеки и пакеты;

2) раздел объявления интерфейса (начало – ключевое слово `entity`) – в нем указываются входные и выходные сигналы схемы и их типы;

3) раздел реализации (начало – ключевое слово `architecture`) – в нем описывается логика работы цифровой схемы.

Комментарии отделяются от текста программы двумя черточками (“минусами”).

Особенностями рассматриваемого варианта программы является использование поведенческого стиля описания проектируемой схемы без привязки к вариантам его схемотехнической реализации. Выходной сигнал счетчика  $Q[3..0]$  в интерфейсной части описан как `inout` по той причине, что в разделе реализации он используется в выражениях слева и справа от оператора присваивания (`<=`). Линия для сигнала вида `inout` позволяет передавать информацию в двух направлениях,

что с одной стороны дает возможность более гибко распределять ресурсы ПЛИС, но с другой стороны может вызвать определенные трудности как при моделировании работы схемы, так и при ее использовании. В листинге 1 приведен вариант программы без использования сигнальных линий типа inout. Для организации работы счетчика используется внутренний сигнал qq, что предполагает при синтезе схемы на ПЛИС использование дополнительных ресурсов для хранения этого сигнала.

Листинг 1

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter_10_t is
  port(
    clk: in std_logic;
    en : in std_logic;
    q: out std_logic_vector (3 downto 0);
    car : out std_logic);
end counter_10_t ;

architecture bbb of counter_10_t is
  signal qq: std_logic_vector (3 downto 0);
begin
  process(clk)
  begin
    if (clk'event and clk='1') then
      if (en='1') then
        if qq="1010" then qq<="0000"; car<='1';
          else car<='0'; qq<=qq+1; q<=qq; end if;
        end if;
      end if;
    end process;
  end bbb;

```

**3.3.5. Проектирование декодера.** По описанной выше технологии спроектируем еще один узел – декодер, который должен преобразовывать четырехразрядный двоично-десятичный код с выхода счетчика в семиразрядный код, управляющий сегментами семисегментного индикатора. Каждому сегменту этого индикатора соответствует



один разряд кода (рис. 3.14).

Символ	Разряды кода						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
E	1	0	0	1	1	1	1

Рис. 3.14

Например, для индикации символа “2” необходимо зажечь сегменты a,b,d,e,g и не зажигать сегменты c и f. Таким образом, семиразрядный код символа “2” будет следующим: {abcdefg}={1101101}. Следовательно для отображения на индикаторе символа “2” декодер должен преобразовать четырехразрядный натуральный код 0010 числа 2 с выхода счетчика в семиразрядный код 1011011. В таблице на рис. 3.14 приведены коды символов всех десятичных цифр и символа ошибки, а на рис. 3.15 - главный файл проекта декодера decoder.vhd на языке VHDL.

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY decoder IS
    PORT(
        hex      : IN  STD_LOGIC_VECTOR(3 downto 0); -- 4-х разрядный код с выхода счетчика;
        led      : OUT STD_LOGIC_VECTOR(6 to 0)); -- 7-х разрядный код для индикатора
    END decoder;

    ARCHITECTURE aaa OF decoder IS
        BEGIN
            with hex select
                led<="1111110" when "0000", -- изображение символа 0
                    "0110000" when "0001", -- изображение символа 1
                    "1101101" when "0010", -- изображение символа 2
                    "1111001" when "0011", -- изображение символа 3
                    "0110011" when "0100", -- изображение символа 4
                    "1011011" when "0101", -- изображение символа 5
                    "1011111" when "0110", -- изображение символа 6
                    "1110000" when "0111", -- изображение символа 7
                    "1111111" when "1000", -- изображение символа 8
                    "1111011" when "1001", -- изображение символа 9
                    "1001111" when others; -- изображение символа ошибки
            END aaa;
    END decoder;
  
```

Рис. 3.15

После компиляции проекта создаем файл временных диаграмм (команда File|New| Waveform Editor File) для моделирования декодера. Сохраняем его под именем decoder.scf и командой Node|Enter Nodes from SNF переносим в него сигнальные шины hex и led. Командой File|End Time установите время моделирования 20us, затем, щелкнув правой кнопкой мыши по изображению шины hex, выберите из всплывающего меню команду Overwrite| Count Value, укажите опцию Binary и нажмите кнопку **ОК**. В результате этих действий на шине hex будут сформированы сигналы натурального двоичного кода, изменяющиеся через временной интервал, равный шагу сетки (устанавливается командой Options|Grid Size). Этот временной интервал можно устанавливать кратным шагу сетки, введя в поле Multiplied By диалогового окна Overwrite Count Value соответствующее целое число. Из этого же окна можно устанавливать шаг изменения кода – устанавливается в поле Incremented By. Щелкнув правой кнопкой мыши по изображению шины hex, введите из всплывающего меню команду Ungroup и просмотрите сигналы на сигнальных линиях hex0÷hex3. Проведите моделирование работы декодера ( команда File|Project| Save & Simulate), откройте файл временных диаграмм и просмотрите результаты (рис. 3.16).

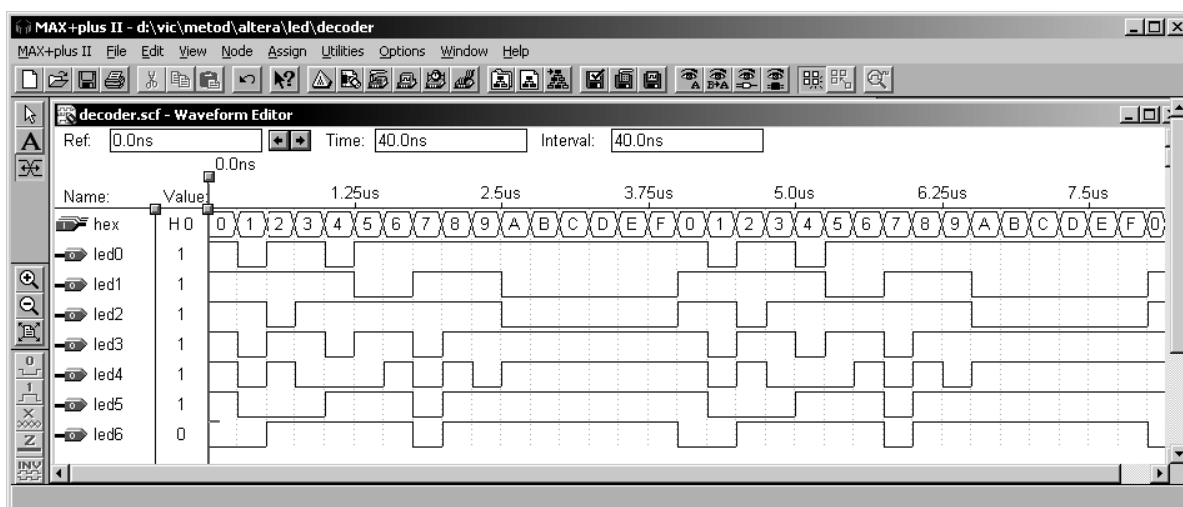


Рис. 3.16

Активизируйте головной файл проекта ( это удобно сделать с помощью изображения иерархического дерева – кнопка “пирамидка”) и

создайте графический образ декодера (команда File|Create Default Symbol). Закройте проект (команда File|Close).

### 3.4. Создание графического файла проекта верхнего уровня иерархии

3.4.1. Графический файл проекта верхнего уровня иерархии создается аналогично тому, как создавались выше графические файлы проектов нижних уровней иерархии. Причем в проекте верхнего уровня могут использоваться созданные ранее графические символы компонентов проектов нижних уровней иерархии. Эти символы доступны непосредственно в поле Symbol Files окна Enter Symbol, если файлы всех проектов нижних уровней иерархии создавались в той же папке, в которой создается проект верхнего уровня. Если же проекты уровней иерархии создавались в других папках, то соответствующий символ компонента будет доступным, если выбрать требуемую папку в поле Directories окна Enter Symbol (рис 3.17).

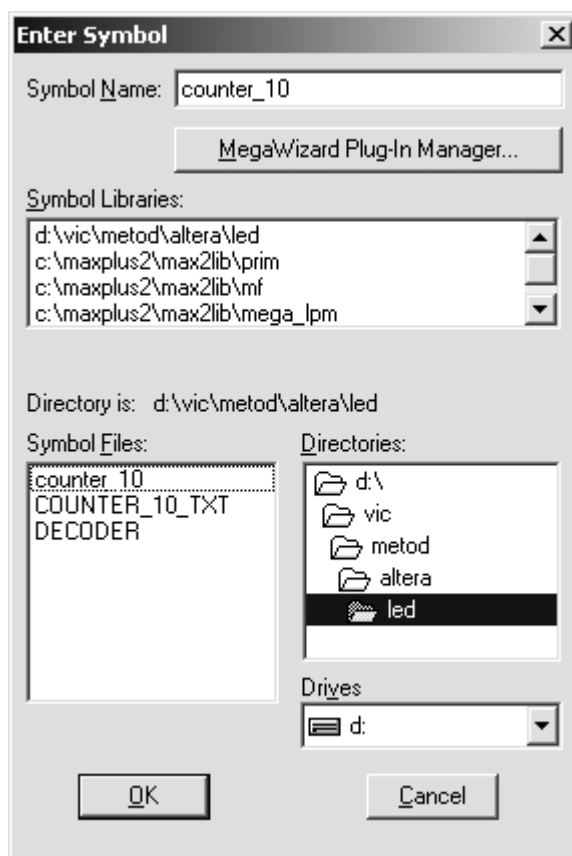


Рис. 3.17

Создадим графический файл проекта верхнего уровня led.gdf, который будет состоять из графических символов, представляющих созданные ранее проекты более низкого уровня: counter\_10.gdf, counter\_10\_txt.vhd и decoder.vhd.

Создайте новый gdf -файл и сохраните под именем led.gdf в папке ...\\led.

1. Задайте имя проекта led (команда File|Project|Set Project to Current File, либо Ctrl+Shift+J, либо нажав кнопку Changes the project name to the name of the current file.. на горизонтальной панели инструментов главного окна MAX PLUS II).

2. Разместите в окне редактора требуемые графические символы компонентов для этой схемы – команда Symbol|Enter Symbol (эту команду можно активизировать двойным щелчком левой кнопки мыши по полю редактора либо из всплывающего меню, щелкнув правой кнопкой мыши по полю редактора).

Название символа:	Файл проекта
a) Counter_10	counter_10.gdf
b) Counter_10_txt	counter_10_txt.vhd
c) Decoder	decoder.vhd
d) Decoder	decoder.vhd
e) базовый элемент GLOBAL	...\\maxplus\\maxlib\\prim
f) 2 контакта INPUT	...\\maxplus\\maxlib\\prim
g) 3 контакта OUTPUT	...\\maxplus\\maxlib\\prim

3. Задайте имена контактов:

“clk” и “En” – для контактов INPUT;

“L[6..0]”, “LL[6..0]”, “Car” – для контактов OUTPUT.

Чтобы задать имя контакта, достаточно дважды щелкнуть левой кнопкой мыши по имени, которое система присваивает ему по умолчанию, и изменить его на требуемое имя. Это же можно сделать, щелкнув правой кнопкой мыши по контакту и выбрав во всплывающем меню команду Edit Pin Name.

4. Нарисуйте линии цепей и линии шин для соединения символов, как показано на рис. 3.18. Для ввода точек соединения проводников или шин переместите указатель выделения или любой другой указатель на пересечение двух линий, щелкните по нему правой кнопкой мыши и выберите команду Toggle Connection Dot из всплывающего меню Edit.

*Примечание. Две цепи или более можно соединить между собой, если этим цепям присвоить одинаковые имена.*

На рис. 3.18 показан файл led.gdf, который создан аналогично тому, как создавался файл counter\_10.gdf. На нижней части рис. 3.18 приведены результаты моделирования проекта led.gdf.

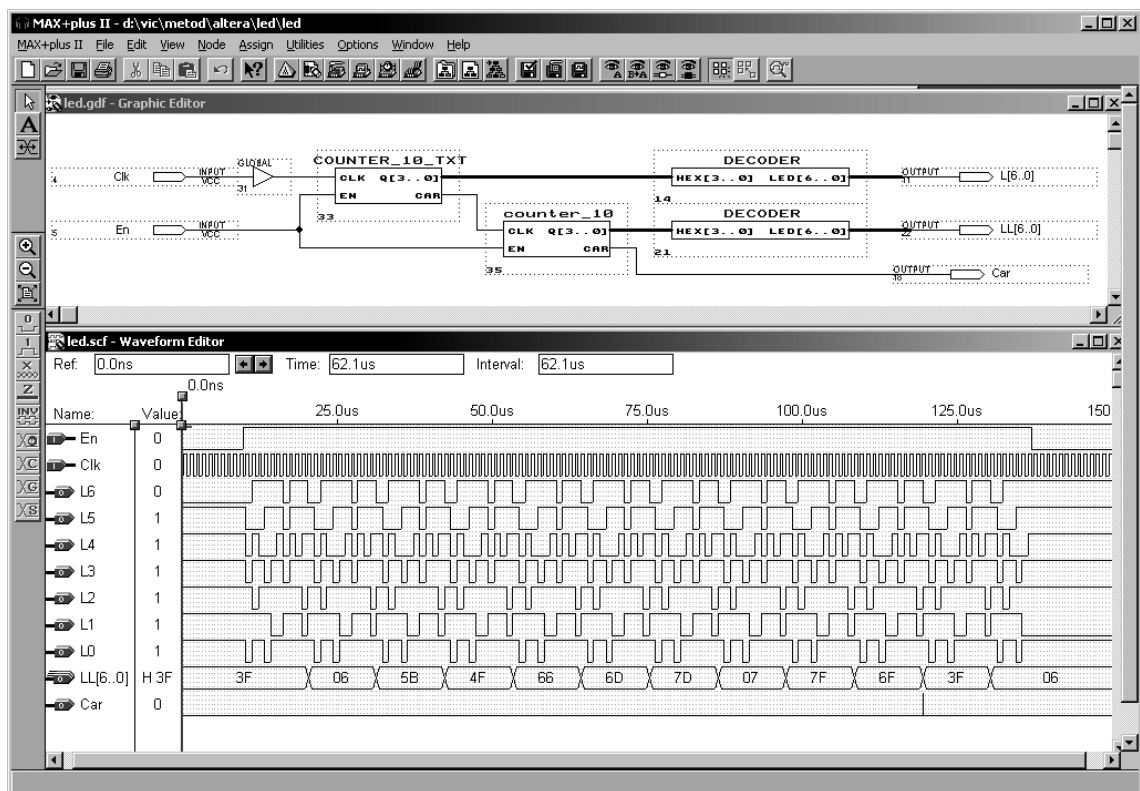


Рис. 3.18

3.4.2. Компиляция проекта. Этот этап включает следующие шаги:

- 1) открытие окна компилятора;
- 2) выбор семейства микросхем;
- 3) запуск компилятора;

## 4) поиск и исправление ошибок.

Чтобы открыть окно компилятора (рис. 3.19), введите команду Max Puls | Compiler (альтернативные варианты File|Project|Save&Compile или Ctrl+L).

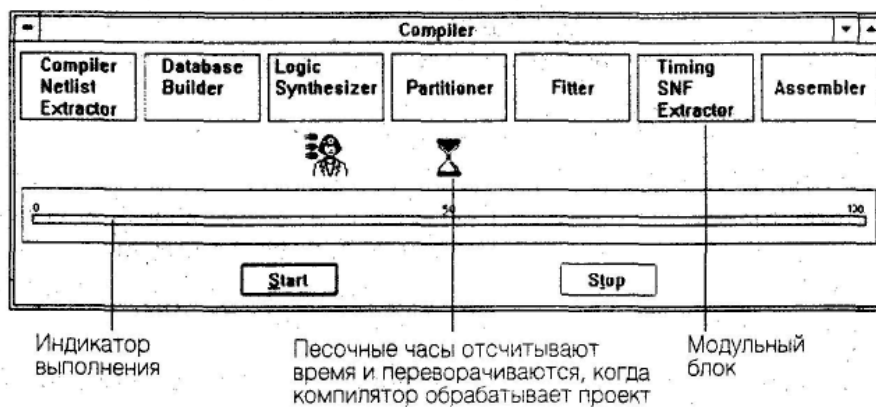


Рис. 3.19

*Примечание. Обратите внимание, что при активизации компилятора в главном меню программы появился новый пункт Processing. Он предназначен для настроек режима работы компилятора.*

Для проведения функциональной верификации проекта (т.е. правильности логики его работы) введите команду Processing|Functional SNF Extractor. В этом случае компилятор игнорирует временные соотношения между сигналами, и будут работать только три первые модуля компилятора (рис. 3.20).

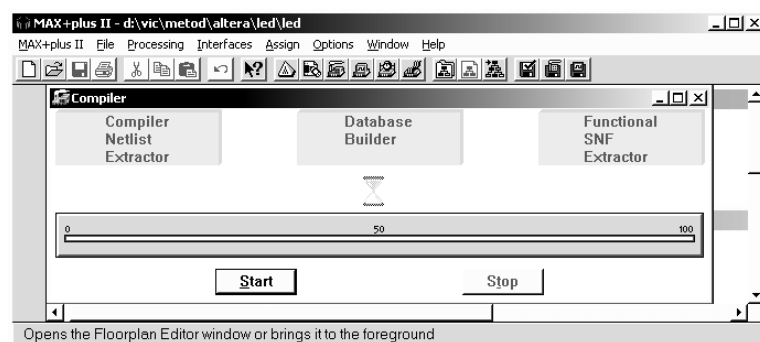


Рис. 3.20

Для запуска компилятора нажмите кнопку Start. По мере того, как компилятор обрабатывает проект, все информационные сообщения, сообщения об ошибках и предупреждения будут появляться в

окне процессора сообщений, которое открывается автоматически (рис. 3.21).

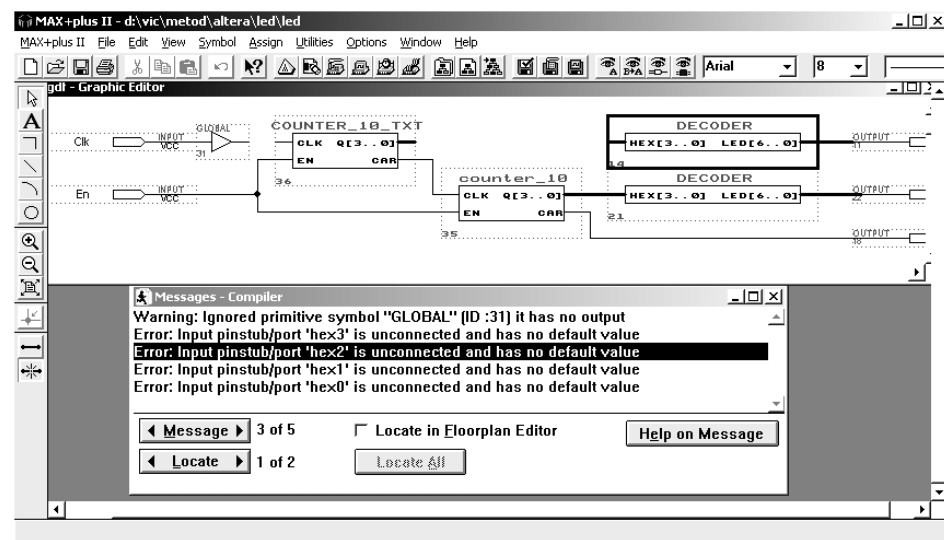


Рис. 3.21

Здесь приведены результаты компиляции проекта led, в который намеренно были внесены ошибки: разорвана связь между выходом счетчика Counter\_10\_txt и входом декодера, а также между входом Clk этого счетчика и компонентом Global. В окне процессора сообщений (Messages – Compiler) появилось одно предупреждение (Warning) и четыре сообщения об ошибках. Генерируемые процессором сообщения можно разделить на три типа:

- а) Error – сообщение о критической ошибке– при появлении первой ошибки компиляция прекращается и выдается такое сообщение;
- б) Warning – предупреждение о некритической ошибке или сделанных системой назначениях;
- г) Info – информационные сообщения.

Сообщение об ошибке и некоторые информационные сообщения могут быть локализованы, т.е. можно найти место в файлах проекта, которое вызвало это сообщение. Для локализации места ошибки в проекте необходимо щелкнуть левой кнопкой мыши по соответствующему сообщению и затем нажать кнопку **Locate**: система автоматически откроет требуемый файл и выделит то место в нем, которое привело к появлению данной ошибки. Устранив ошибки и выполнив повторную компиляцию, убедимся, что формальные

ошибки в проекте отсутствуют (рис. 3.22).

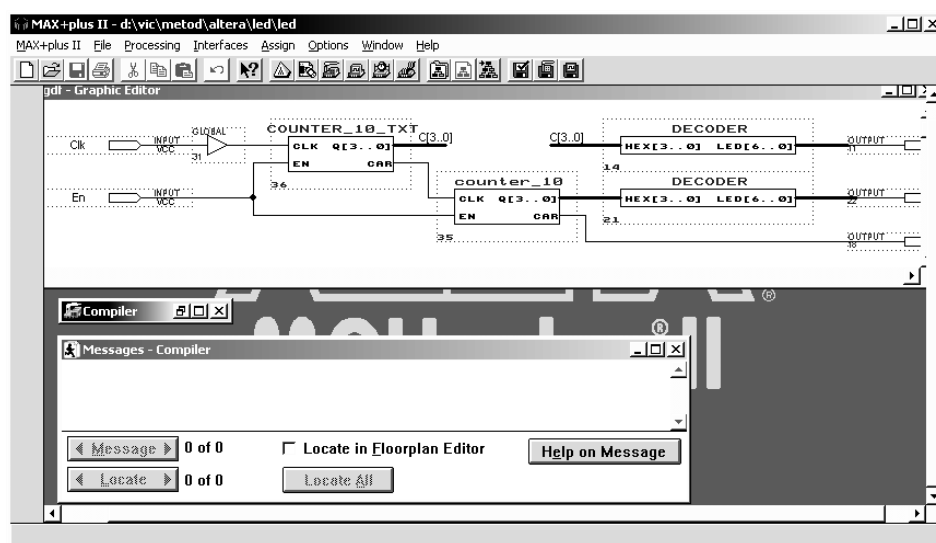


Рис. 3.22

3.4.3. Для проверки правильности логики работы проектируемого устройства необходимо выполнить моделирование его работы. Для выполнения этого этапа проектирования необходимо сформировать файл временных диаграмм, из которого будет считываться информация о входных сигналах, и куда будут выводиться выходные сигналы. Технология создания таких файлов была рассмотрена выше (см. п.п. 3.8.2). Поэтому ограничимся тем, что кратко укажем основные операции и параметры системы, устанавливаемые при выполнении этого этапа.

Командой MAX plus II| Wave Form Editor активизируем графический редактор цифровых сигналов и сохраняем открывшийся файл под именем led.scf. Затем командой Node|Enter Node from SNF (альтернативный вариант вызова – из всплывающего меню, которое активизируется щелчком правой кнопки мыши по полю редактора) вызывается одноименное окно, в котором с помощью кнопок List, => и <= формируется список входных и выходных сигналов. Установить шаг сетки (команда Options|Grid Size) равным 500 ns, а время моделирования (команда File|End Time) равным 150 us. Далее сигнал En установить равным 1 на интервале (10÷140) us, а на входе Clk сформировать последовательность тактовых импульсов с периодом, равным удвоенному шагу сетки (рис. 3.23).



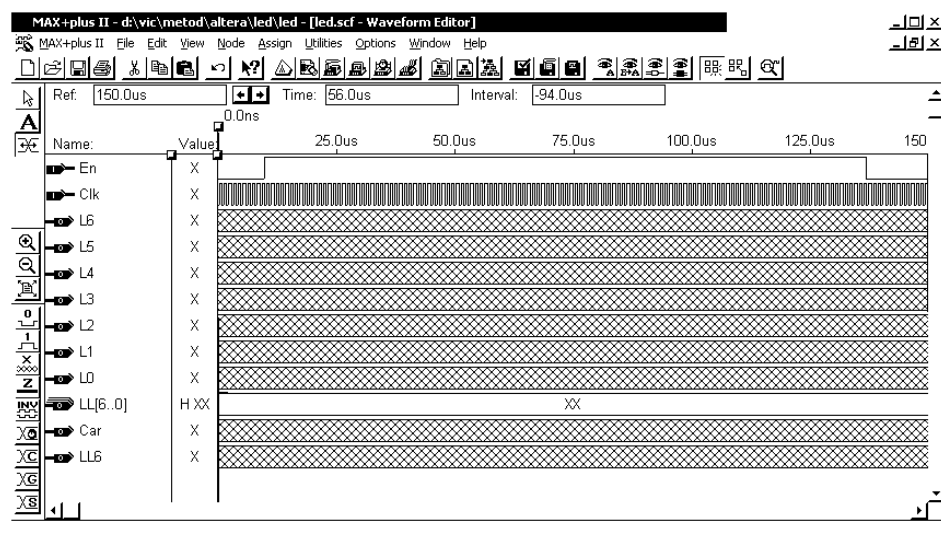


Рис. 3.23

Для моделирования работы спроектированного устройства необходимо вызвать программу Simulator, нажав соответствующую кнопку на панели инструментов либо через главное меню (команда MAX plus II| Simulator). Затем нажать кнопку Start в открывшемся окне, дождаться сообщения о завершении моделирования и нажатием кнопки Open SCF вывести на экран окно с результатами моделирования (рис. 3.24).

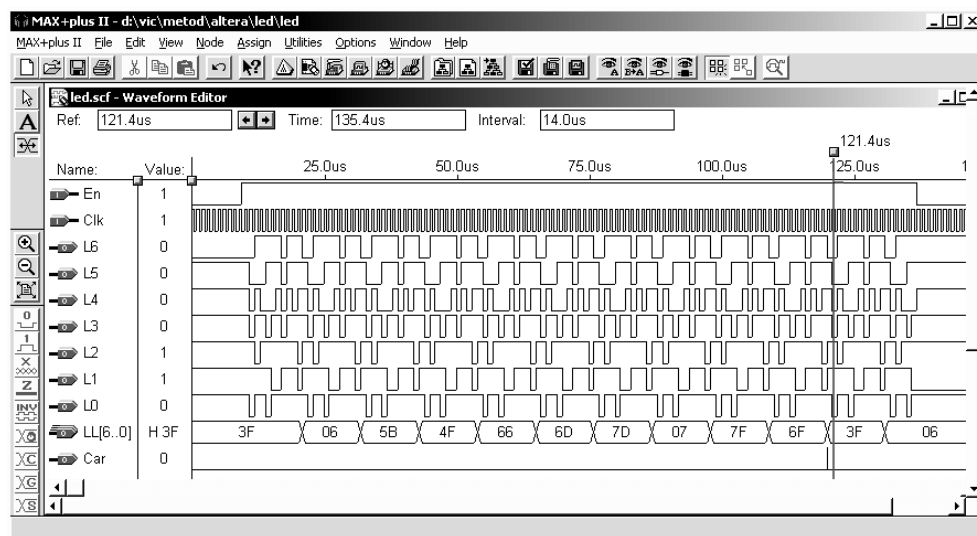


Рис. 3.24

Для определения точного времени каждого изменения логического уровня выполните следующие действия.

1. Нажмите левую кнопку мыши на значке опорного временного маркера и переместите его в нужную точку временной диаграммы.

2. Щелкните левой кнопкой мыши по стрелкам перемещения опорного временного маркера вправо или влево для его перемещения к первому изменению логического уровня.

2. Щелкая левой кнопкой мыши по стрелкам перемещения опорного временного маркера вправо или влево, вы можете перемещаться к последующим или предыдущим изменениям логических уровней сигналов.

Точное время, соответствующее положению опорного временного маркера, отображается сверху над ним и в поле Ref.

Временной анализ результатов моделирования проектируемого устройства – двухступенчатого десятичного счетчика – показывает, что он работает неправильно. В самом деле, поскольку тактовый интервал сигнала Clk составляет 1 us, то через 100 us после начала счета счетчики должны устанавливаться в исходное состояние и вырабатываться сигнал переноса Car. Однако в моделируемом устройстве это происходит только через 110 us. Это произошло по той причине, что неправильно работает компонент проекта Counter\_10\_txt: он сбрасывается в исходное состояние не 10-м, а 11-м тактовым импульсом. Анализ файла проекта (рис. 3.12) показывает, что при его разработке допущена ошибка в сроке проверки переполнения счетчика и формирования сигнала переноса (рис. 3.25).

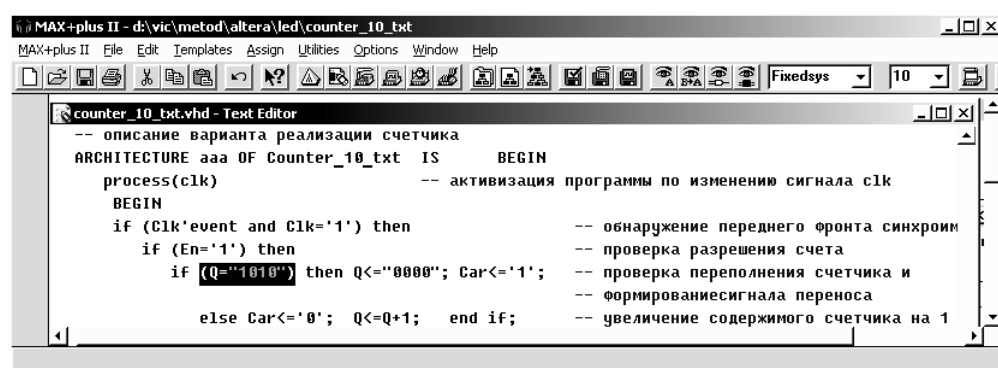


Рис. 3.25

Содержимое счетчика Q сравнивается с “1010” – запись числа 10 в двоичном коде, тогда как сравнение необходимо проводить с числом “1001” – 9 в двоичном коде.

Исправив эту ошибку в файле Counter\_10\_txt.vhd, откомпилировав заново проект и выполнив моделирование, получим временные диаграммы, показанные на рис. 3.26. Анализ временных диаграмм показывает, что теперь устройство работает правильно.

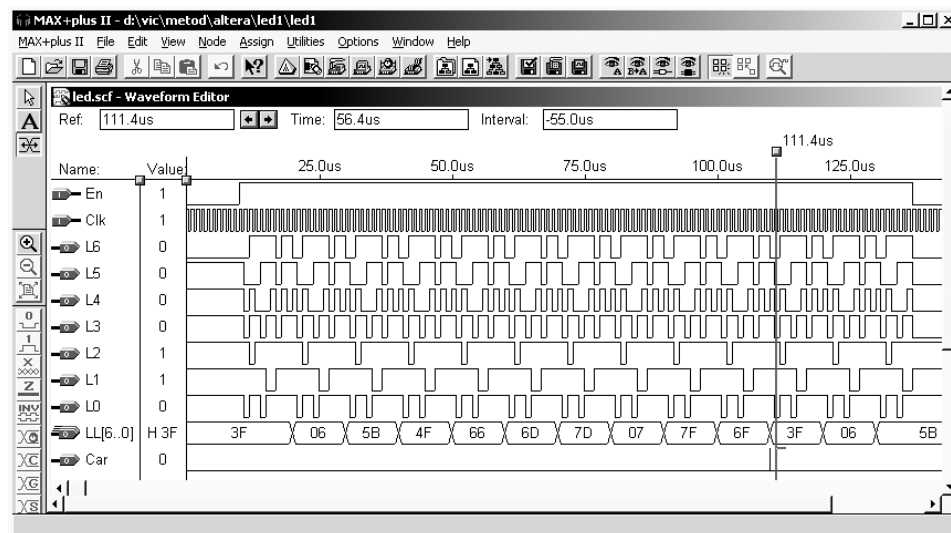


Рис. 3.26

3.4.4. Заключительным шагом проектирования на уровне функционального синтеза устройства является выбор микросхемы ПЛИС, на которой оно будет реализовано. Одним из важнейших критериев, по которому выбирается тип ПЛИС, является выполнение временных соотношений между входными и выходными сигналами, а также сигналами синхронизации. Для оценки временных соотношений между входными и выходными сигналами необходимо указать, какие требования предъявляются к рассматриваемым временным соотношениям, и перекомпилировать проект.

Чтобы указать предъявляемые к устройству временные требования, введите команду Assign | Global Project Timing Requirements и в открывшемся диалоговом окне (рис. 3.27), введите значения допустимых задержек: *tpd* (вход – нерегистровый выход, т.е. время распространения сигнала со входа через комбинационную логику на выход), *tsu* (синхросигнал – время окончания переходных процессов), *tco* (синхросигнал – выходной сигнал), *fmax* – максимальная

частота синхронизации.

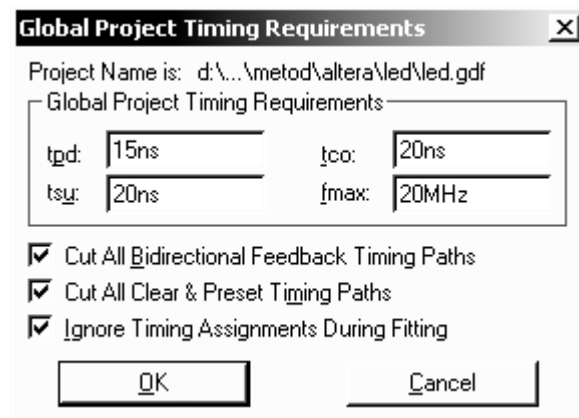


Рис. 3.27

Перед началом компиляции необходимо указать компилятору семейство микросхем или конкретную микросхему, на которых будет реализовано проектируемое устройство. Тип микросхемы в пределах семейства компилятор может выбрать автоматически либо его также необходимо указать.

Для выбора микросхем введите команду Assign| Device. Откроется диалоговое окно **Device** (микросхема) (рис. 3.28):

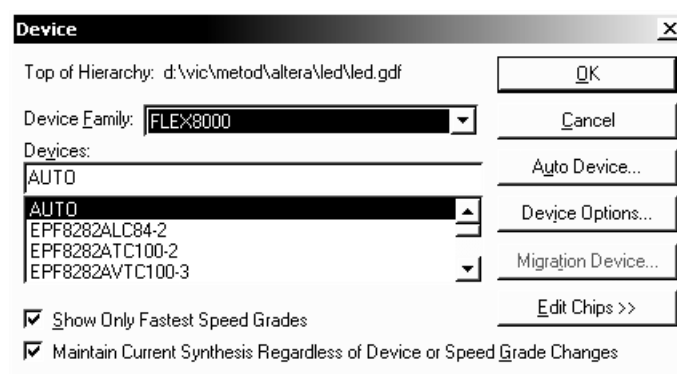


Рис. 3.28

Если требуемое семейство не выбрано, то выберите его (например, FLEX8000) в окошке раскрывающегося списка *Device Family*, затем укажите тип микросхемы в раскрывающемся списке *Devices*. Если выбор конкретного типа микросхемы не важен, выберите AUTO.

Активизируем компилятор (командой Max Puls | Compiler или любым из альтернативных способов), введем команду Processing | Timing SNF Extractor и запустим компилятор, нажав кнопку Start в окне

компилятора. После окончания компиляции откроем окно процессора сообщений (рис. 3.29).

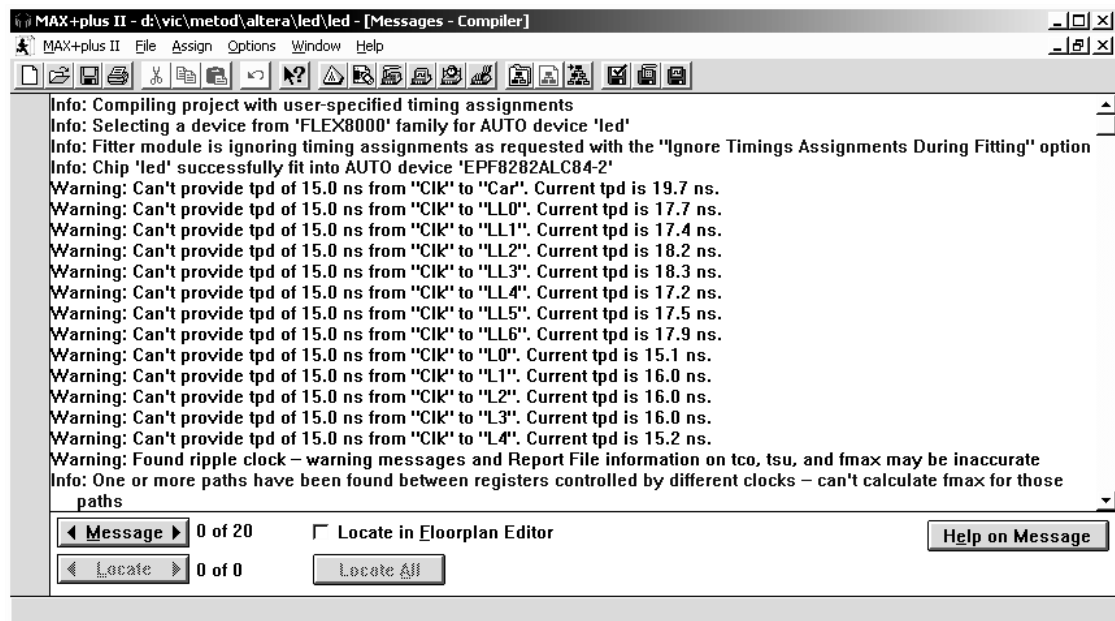


Рис. 3.29

Как следует из результатов компиляции, выбранный системой тип ПЛИС EPF8282ALC84-2 не позволяет выполнить предъявленные к проекту требования по временному параметру tpd: для выходов LL[6..0], для выходов L[4..0] и для выхода Car. Можно поменять семейство микросхем на FLEX6000 и перекомпилировать проект. Это несколько улучшит ситуацию (рис.3.30), но не позволит полностью выполнить предъявленные требования.

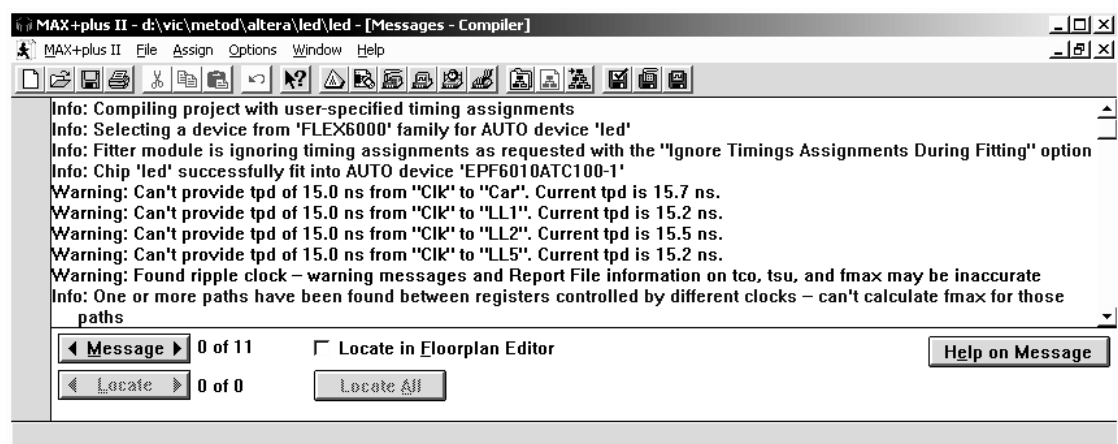


Рис. 3.30

Изменив требуемое значение tpd с 15ns на 16ns и перекомпили-

ровав проект, увидим, что в этом варианте предъявляемые временные требования к устройству выполняются.

Этой операцией заканчивается функциональное проектирование цифрового устройства и далее можно переходить к конструкторским этапам проекта: размещению узлов схемы в логических блоках микросхемы, назначению выводов и трассировке (конфигурированию) ПЛИС.

## **ЗАКЛЮЧЕНИЕ**

В первой части пособия дана краткая характеристика современных ПЛИС и рассмотрены общие принципы проектирования цифровых схем на них. На примере системы MAX+Plus II проиллюстрирована технология создания комбинированного проекта иерархической структуры. За рамками пособия заключительные этапы проектирования цифровых устройств на ПЛИС – анализ временных соотношений между сигналами, конструкторско-технологические аспекты (предпочтения при размещении отдельных компонентов проектируемого устройства в теле кристалла, распределение сигналов по выводам БИС и т.д.). Тем не менее, приведенный материал позволяет составить достаточно полное представление о технологии и принципах проектирования цифровых устройств на ПЛИС. Во второй части пособия будет рассмотрена технология работы с более удобными и мощными системами Quartus и Active HDL. Пособие может оказаться полезным не только для освоения технологии проектирования цифровых устройств на ПЛИС, но и при изучении студентами дисциплин по цифровой схемотехнике и по освоению HDL-языков.

## **Библиографический список**

1. Стешенко В.Б. ПЛИС фирмы «Altera»: элементная база, система проектирования и языки описания аппаратуры. – М.: «Додэка», 2002.

2. Стешенко В.Б. ПЛИС фирмы «Altera»: проектирование устройств обработки сигналов. – М.: «Додэка», 2000.
3. Кузелип М. О., Кнышев Д. А., Зотов В. Ю. Современные семейства ПЛИС фирмы Xilinx: Справочное пособие. – М.: Горячая линия – Телеком, 2004.
4. Поляков А. К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. – М.: СОЛОН-Пресс, 2003. – 320 с: ил. – (Серия "Системы проектирования").
5. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL. – М.: Горячая линия – Телеком, 2005.
6. Грушвицкий Р. И., Мурсаев А. Х., Угрюмов Е. П. Проектирование систем на микросхемах программируемой логики. – СПб.: БХВ-Петербург, 2002. – 608 с: ил.
7. Суворова Е. А., Шейнин Ю- Е. Проектирование цифровых систем на VHDL. – СПб.: БХВ-Петербург, 2003.
8. Разевиг В. Д. Система проектирования цифровых устройств OrCad. – М.: Солон-Р, 2000.
9. Соловьев В. В. Проектирование цифровых систем на основе программируемых логических интегральных схем. – М.: Горячая линия – Телеком, 2007.
10. ByteBlasterMV Parallel Port Download Cable, Data Sheet, «Altera corporation», ver.1, April 1998.

**Шеболков Виктор Васильевич**

**Проектирование цифровых устройств на ПЛИС  
в САПР MAX Plus II**

**Учебное пособие**

Ответственный за выпуск Шеболков В.В.

Редактор

Корректор

ЛР №020565 от 23.06. 1997г. Подписано к печати\_\_\_\_\_г.

Бумага офсетная. Печать офсетная.

Формат 60x84<sub>1/16</sub>

Усл.п.л. – 4,0 Уч.-изд.л. –

Заказ №\_\_\_\_\_ Тираж экз.

”С”

---

Издательство Технологического института  
Южного федерального университета  
ГСП 17А, Таганрог, 28, Некрасовский, 44  
Типография Технологического института  
Южного федерального университета  
ГСП 17А, Таганрог, 28, Энгельса, 1