ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П.КОРОЛЁВА»

А.Н. МУРАВЬЕВ

РАЗРАБОТКА ЦИФРОВЫХ СХЕМ НА БАЗЕ ПРОГРАММИРУЕМОЙ ЛОГИКИ

Утверждено Редакционно-издательским советом университета
в качестве учебного пособия

САМАРА
Издательство СГАУ
2010

УДК СГАУ: 621.381
ББК 32.97
М 73

Рецензенты: канд. техн. наук В. А. Глазунов
канд. техн. наук В. Т. Колесников

Муравьев А.Н.
М 73 **РАЗРАБОТКА ЦИФРОВЫХ СХЕМ НА БАЗЕ ПРОГРАММИРУЕМОЙ ЛОГИКИ:** учеб. пособие / *А.Н. Муравьев.* – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2010. – 68 с.

ISBN 978-5-7883-0734-3

Подробно изложен процесс разработки цифрового устройства на базе ПЛМ в пакете САПР ALTERA MAX PLUS. Представлены характеристики современных микросхем ПЛМ, их структурные схемы и области применения. Приведены способы описания проектов цифровых устройств на языке высокого уровня.

Учебное пособие рекомендуется для студентов, изучающих дисциплину «Системы автоматизированного проектирования радиоэлектронной аппаратуры» по специальности «Биотехнические и медицинские аппараты и системы» - 200401 и студентов, изучающих дисциплину «Прикладная информатика» по специальности «Радиотехника» - 210302.

УДК СГАУ: 621.381
ББК 32.97

ISBN 978-5-7883-0734-3

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современной электронике все большее значение приобретают интегральные микросхемы, пришедшие на смену дискретным радиодеталям, которые, однако, все еще встречаются в некоторых областях электроники, например там, где требуется особая точность. Кроме того, обойтись только стандартными интегральными микросхемами невозможно и использование некоторого количества дискретных элементов просто необходимо, чтобы собираемая схема полностью отвечала предъявляемым к ней требованиям.

Несмотря на огромный выбор специализированных изделий, представленных на рынке, разработчик не всегда может найти тот единственный компонент, который поможет решить все его проблемы. Решением является использование интегральных микросхем, внутренняя структура которых может многократно изменяться самим пользователем, причем сделать это очень легко.

Теперь можно самостоятельно "произвести" оригинальные радиоэлементы либо в единственном экземпляре, либо в виде небольшой партии, причем без огромных капиталовложений, необходимых для производства "заказных" микросхем.

FPGA (field programmable gate arrays), или *ПЛИС (программируемые логические интегральные схемы),* представляют собой *цифровые интегральные микросхемы (ИС),* состоящие из программируемых логических блоков и программируемых соединений между этими блоками. ПЛИС, как правило, использовались для создания прототипов заказных микросхем или для создания испытательных стендов, на которых проверяется физическая реализуемость новых алгоритмов. Однако благодаря низким затратам на производство и малым срокам выхода на рынок эти микросхемы всё чаше используются как конечный продукт. У некоторых крупных поставщиков ПЛИС есть устройства, которые составляют прямую конкуренцию заказным микросхемам.

Современные ПЛИС находят применение практически в любой сфере, включая устройства связи и программируемые радиостанции. ПЛИС применяют в радиолокации, обработке изображений и в других приложениях *цифровой обработки сигналов (ЦОС).* ПЛИС используют повсюду, в том числе и в *однокристальных системах,* содержащих программные и аппаратные модули. Если быть более точным, в настоящее время ПЛИС заполняют четыре крупных сегмента рынка: заказные интегральные схе-

мы, цифровая обработка сигналов, системы на основе встраиваемых микроконтроллеров и микросхемы, обеспечивающие физический уровень передачи данных. Кроме того, с появлением ПЛИС возник новый сектор рынка — *системы с перестраиваемой архитектурой,* или *reconfigurable computing (RC).*

В учебном пособии рассматриваются устройство современных ПЛИС, методика разработки цифровых схем в пакете САПР фирмы ALTERA, основы использования языков высокого уровня VHDL и AHDL для описания электронных узлов.

1. ПРОГРАММИРУЕМЫЕ ЛОГИЧЕСКИЕ МИКРОСХЕМЫ

В современной электронике все большее значение приобретают интегральные микросхемы. Наиболее заметной тенденцией последних лет является быстрый рост интеграции микросхем. Благодаря непрерывно совершенствующимся технологиям одна интегральная микросхема в настоящее время может содержать эквивалент микро-ЭВМ средней мощности. Сфера применения цифровых методов обработки информации (ЦОС) постоянно расширяется, они все чаще вытесняют использовавшиеся ранее аналоговые схемы. Новые интегральные микросхемы становятся все более специализированными. Несмотря на огромный выбор специализированных изделий, представленных на рынке, разработчик не всегда может найти тот единственный чудодейственный компонент, который поможет решить все его проблемы.

Понятие "программируемая логика" хорошо известно и представляет собой полную противоположность "логике жесткой". В устройствах на жесткой логике поведение схемы полностью определяется связями между определенным числом элементов. Модернизация или изменение схемы устройства достаточно трудоемки. В системах с программируемой логикой используется центральный процессор (микропроцессор, микроконтроллер), который связан с какими-либо периферийными устройствами и выполняет программу, содержащуюся в энергонезависимой памяти той или иной конструкции.

Существуют устройства, занимающие промежуточное положение между жесткой логикой и однокристальными микро-ЭВМ. Речь идет о программируемых логических интегральных схемах (ПЛИС), которые очень популярны у производителей. Это не программируемые запоминающие устройства (ПЗУ), а интегральные микросхемы, внутренняя структура которых может многократно изменяться самим пользователем, причем сделать это очень легко. Теперь можно самостоятельно "произвести" оригинальные радиоэлементы либо в единственном экземпляре, либо в виде небольшой партии, причем без огромных капиталовложений.

Последние годы характеризуются резким ростом плотности упаковки элементов на кристалле, многие ведущие производители либо начали серийное производство, либо анонсировали ПЛИС с эквивалентной ёмкостью более 1 млн. логических вентилей.

Приведём известную классификацию ПЛИС [1,2] по структурному признаку, так как она даёт наиболее полное представление о классе задач, пригодных для решения на той или иной ПЛИС. Следует заметить, что общепринятой оценкой логической ёмкости ПЛИС является число экви-

валентных вентилей, определяемое как среднее число вентилей “2И-НЕ”, необходимых для реализации эквивалентного проекта на ПЛИС и базовом матричном кристалле (БМК). Понятно, что эта оценка весьма условна, поскольку ПЛИС не содержат вентилей “2И-не” в чистом виде, однако для проведения сравнительного анализа различных архитектур она вполне пригодна. Основным критерием такой классификации является наличие, вид и способы коммутации элементов логических матриц. По этому признаку можно выделить несколько классов ПЛИС.

Программируемые логические матрицы — наиболее традиционный тип ПЛИС, имеющий программируемые матрицы “И” и “ИЛИ”. В зарубежной литературе соответствующими этому классу аббревиатурами являются FPLA (*Field Programmable Logic Array*) и FPLS (*Field Programmable Logic Sequensers*). Примерами таких ПЛИС могут служить отечественные схемы К556РТ1,РТ2,РТ21. Недостаток такой архитектуры — слабое использование ресурсов программируемой матрицы “ИЛИ”, поэтому дальнейшее развитие получили микросхемы, построенные по архитектуре программируемой матричной логики (ПМЛ) (PAL — *Programmable Array Logic*) — это ПЛИС, имеющие программируемую матрицу ”И” и фиксированную матрицу “ИЛИ”. К этому классу относится большинство современных ПЛИС небольшой степени интеграции. В качестве примеров можно привести отечественные ИС КМ1556ХП4, ХП6, ХП8, ХЛ8, ранние разработки (середина–конец 1980-х годов) ПЛИС фирм INTEL, ALTERA, AMD, LATTICE и др. Разновидностью этого класса являются ПЛИС, имеющие только одну (программируемую) матрицу “И”, например схема 85С508 фирмы INTEL. Следующий традиционный тип ПЛИС — программируемая макрологика. Они содержат единственную программируемую матрицу “И-НЕ” или “ИЛИ-НЕ”, но за счёт многочисленных инверсных обратных связей способны формировать сложные логические функции. К этому классу относятся, например, ПЛИС PLHS501 и PLHS502 фирмы SIGNETICS, имеющие матрицу “И-НЕ”, а также схема XL78C800 фирмы EXEL, основанная на матрице “ИЛИ-НЕ”.

Вышеперечисленные архитектуры ПЛИС, содержащие небольшое число ячеек, к настоящему времени морально устарели и применяются для реализации относительно простых устройств, для которых не существует готовых ИС средней степени интеграции. Естественно, для реализации, например, сложных алгоритмов ЦОС они непригодны.

ИС ПМЛ (PLD) имеют архитектуру, весьма удобную для реализации цифровых автоматов. Развитие этой архитектуры — программируемые коммутируемые матричные блоки (ПКМБ) — это ПЛИС, содержа-

щие несколько матричных логических блоков (МЛБ), объединённых коммутационной матрицей. Каждый МЛБ представляет собой структуру типа ПМЛ, то есть программируемую матрицу “И”, фиксированную матрицу “ИЛИ” и макроячейки. ПЛИС типа ПКМБ, как правило, имеют высокую степень интеграции (до 10000 эквивалентных вентилей, до 256 макроячеек). К этому классу относятся ПЛИС семейства MAX5000 и MAX7000 фирмы ALTERA, схемы XC7000 и XC9500 фирмы XILINX, а также большое число микросхем других производителей (Atmel, Vantis, Lucent и др.). В зарубежной литературе они получили название *Complex Programmable Logic Devices* (CPLD).

Другой тип архитектуры ПЛИС — программируемые вентильные матрицы (ПВМ), состоящие из логических блоков (ЛБ) и коммутирующих путей — программируемых матриц соединений. Логические блоки таких ПЛИС состоят из одного или нескольких относительно простых логических элементов, в основе которых лежат таблица перекодировки (ТП, *Look-up table* — LUT), программируемый мультиплексор, D-триггер, а также цепи управления. Таких простых элементов может быть достаточно много, например, у современных ПЛИС ёмкостью до 1 млн. вентилей число логических элементов достигает нескольких десятков тысяч. За счёт такого большого числа логических элементов они содержат значительное число триггеров, а также некоторые семейства ПЛИС имеют встроенные реконфигурируемые модули памяти (РМП, *embedded array block* — EAB), что делает ПЛИС данной архитектуры весьма удобным средством реализации алгоритмов цифровой обработки сигналов, основными операциями в которых являются перемножение, умножение на константу, суммирование и задержка сигнала. Вместе с тем, возможности комбинационной части таких ПЛИС ограничены, поэтому совместно с ПВМ применяют ПКМБ (CPLD) для реализации управляющих и интерфейсных схем. В зарубежной литературе такие ПЛИС получили название *Field Programmable Gate Array* (FPGA). К FPGA (ПВМ) классу относятся ПЛИС XC2000, XC3000, XC4000, Spartan, Virtex фирмы XILINX; ACT1, ACT2 фирмы ACTEL, а также семейства FLEX8000 фирмы ALTERA, некоторые ПЛИС Atmel и Vantis.

Типовая структура FPGA ПЛИС обычно содержит следующие узлы:

множество конфигурируемых логических блоков (*Configurable Logic Blocks* — CLBs), которые объединяются с помощью матрицы соединений;

элементы ввода/вывода (*input/output blocks* — IOBs), позволяющие реализовать двунаправленный ввод/вывод, третье состояние и т. п.

На рис.1 представлена блок-схема микросхемы FPGA семейства MAX7000 фирмы Altera, а на рис.2 изображена структура макроячейки этой микросхемы [3]. Особенностью современных ПЛИС является возможность тестирования узлов с помощью порта JTAG , а также наличие внутреннего генератора и схем управления последовательной конфигурацией.

Дальнейшее развитие архитектур идёт 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 для "систем на кристалле" (*system-on-chip*, SOC). В основе идеи SOC лежит интеграция всей электронной системы в одном кристалле (например, в случае ПК такой чип объединяет процессор, память и т. д.). Компоненты этих систем разрабатываются отдельно и хранятся в виде файлов параметризируемых модулей. Окончательная структура SOC-микросхемы выполняется на базе этих "виртуальных компонентов" с помощью программ систем автоматизации проектирования (САПР) электронных устройств — EDA (*Electronic Design Automation*). Благодаря стандартизации в одно целое, можно объединять "виртуальные компоненты" от разных разработчиков.

Как известно, при выборе элементной базы систем обработки сигналов обычно руководствуются следующими критериями отбора:

- быстродействие;
- логическая ёмкость, достаточная для реализации алгоритма;
- схемотехнические и конструктивные параметры ПЛИС, надёжность, рабочий диапазон температур, стойкость к ионизирующим излучениям и т. п.;
- стоимость владения средствами разработки, включающая как стоимость программного обеспечения, так наличие и стоимость аппаратных средств отладки;
- стоимость оборудования для программирования ПЛИС или конфигурационных ПЗУ;
- наличие методической и технической поддержки;
- наличие и надёжность российских поставщиков;
- стоимость микросхем.

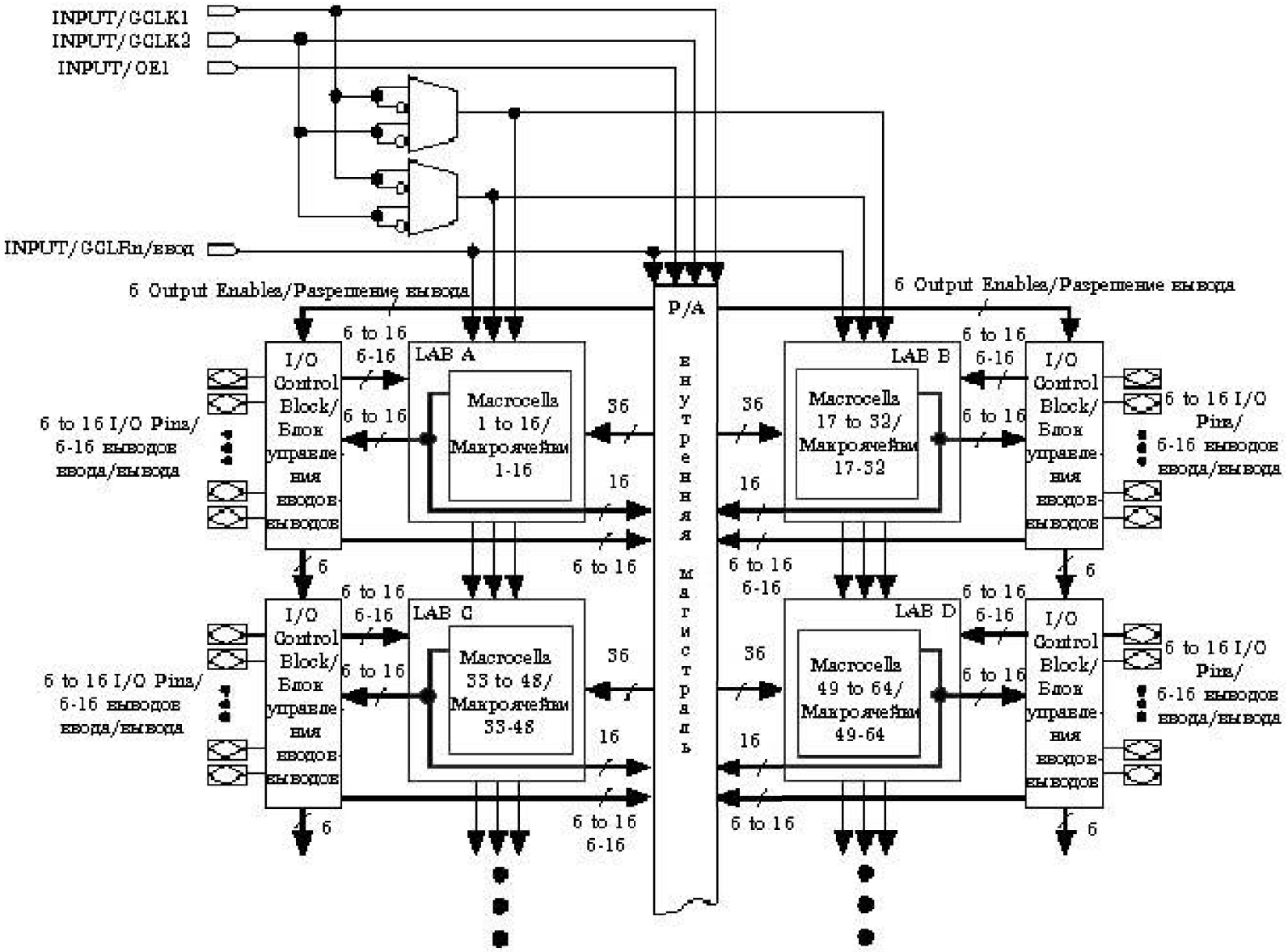

Рис. 1. Структура устройства семейства MAX7000 фирмы Altera

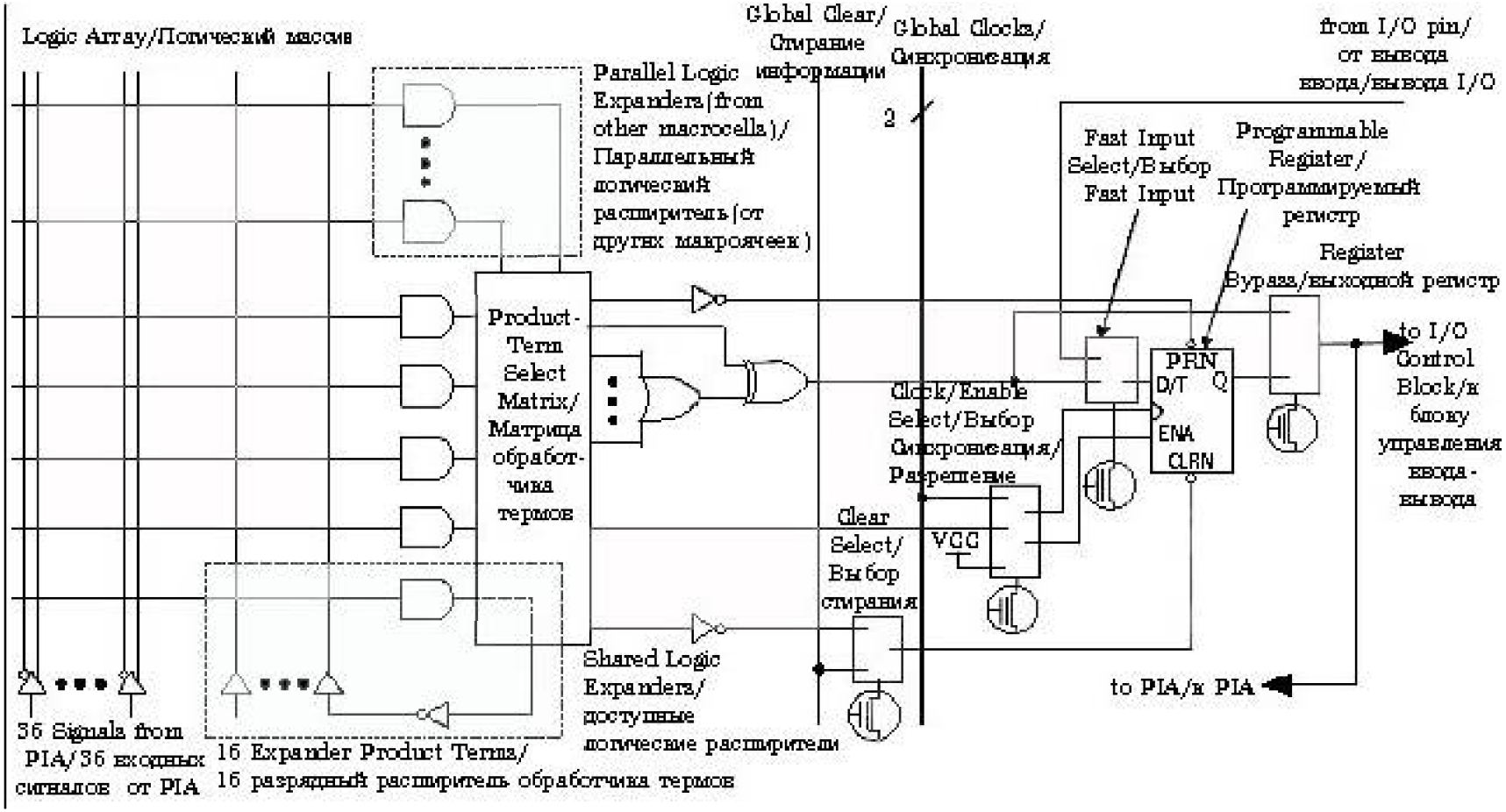

Рис. 2. Структура макроячейки микросхемы семейства MAX7000 фирмы Altera

Рассмотрим с этих позиций продукцию ведущих мировых производителей ПЛИС, доступную на российском рынке.

Фирма Altera Corporation, (101 Innovation Drive, San Jose, CA 95134, USA, http://www.altera.com/) была основана в июне 1983 года. В настоящее время High-end продуктом этой фирмы является семейство APEX20K, особенности архитектуры которого упоминались выше, а в табл. 1 приведены основные параметры ПЛИС этого семейства.

Таблица 1. ***Основные характеристики ПЛИС семейства APEX20K фирмы ALTERA***

Тип ПЛИС	**EP20K100**	**EP20K160**	**EP20K200**	**EP20K300**	**EP20K400**	**EP20K600**	**EP20K1000**
Максимальное число эквивалентных вентилей	263 000	404 000	526 000	728 000	1 052 000	1 537 000	2 670 000
Число логических элементов	4 160	6 400	8 320	11 520	16 640	24 320	42 240
Встроенные блоки памяти	26	40	52	72	104	152	264
Максимальный объем памяти, бит	53 248	81 920	106 496	147 456	212 992	311 296	540 672
Число макроячеек	416	640	832	1 152	1 664	2 432	4 224
Число выводов пользователя	252	320	382	420	502	620	780

Дополнительным фактором при выборе ПЛИС Altera является наличие достаточно развитых бесплатных версий САПР. В табл. 2 приведены основные характеристики пакета MAX+PLUS II BASELINE ver. 9.3 фирмы Altera, который можно бесплатно “скачать” с сайта http://www.altera.com/ или получить на CD Altera Digital Library, на котором содержится также и полный набор документации по архитектуре и применению ПЛИС.

Кроме того, ПЛИС фирмы Altera выпускаются с возможностью программирования в системе непосредственно на плате. Для программирования и загрузки конфигурации устройств фирмой Altera опубликована схема загрузочного кабеля ByteBlaster и ByteBlasteMV. Следует отметить, что новые конфигурационные ПЗУ EPC2 позволяют программировать с помощью этого устройства, тем самым отпадает нужда в программаторе,

что, естественно, снижает стоимость владения технологией. ПЛИС фирмы Altera выпускаются в корпусах, соответствующих коммерческому и индустриальному диапазону температур.

Таблица 2. ***Основные характеристики пакета MAX+PLUS II BASELINE ver. 9.3***

Поддерживаемые устройства	EPF10K10, EPF10K10A, EPF10K20, EPF10K30, EPF10K30A, EPF10K30E (до 30 000 эквивалентных вентилей), EPM9320, EPM9320A, EPF8452A, EPF8282A, MAX7000, FLEX6000, MAX5000, MAX3000A, Classic
Средства описания проекта	Схемный ввод, поддержка AHDL, средства интерфейса с САПР третьих фирм, топологический редактор, иерархическая структура проекта, наличие библиотеки параметризируемых модулей
Средства компиляции проекта	Логический синтез и трассировка, автоматическое обнаружение ошибок, поддержка мегафункций по программам MegaCore и AMPP
Средства верификации проекта	Временной анализ, функциональное и временное моделирование, анализ сигналов, возможность использования программ моделирования (симуляторов) третьих фирм

Компания Xilinx Inc. (2100 Logic Drive, San Jose, CA 95124-3400, USA, http://www.xilinx.com/) была основана в феврале 1984, её High-end продуктом являются ПЛИС семейства Virtex, основные характеристики которых представлены в табл. 3.

Таблица 3. ***Основные характеристики ПЛИС семейства Virtex фирмы XILINX***

Тип ПЛИС	**XCV50**	**XCV100**	**XCV150**	**XCV200**	**XCV300**	**XCV400**	**XCV600**	**XCV800**	**XCV1000**
Максимальное число эквивалентных вентилей	57 906	108 904	164 674	236 666	322 970	468 252	661 111	888 439	1124022
Число логических элементов	1 728	2 700	3 888	5 292	6 912	10 800	15 552	21 168	27 648
Максимальный объем памяти, бит	24 576	38 400	55 296	75 264	98 304	153 600	221 184	301 056	393 216
Число выводов пользователя	180	180	260	284	316	404	512	512	512

Архитектура семейства Virtex характеризуется широким разнообразием высокоскоростных трассировочных ресурсов, наличием выделенного блочного ОЗУ, развитой логикой ускоренного переноса. ПЛИС данной серии обеспечивают высокие скорости межкристального обмена — до 200 МГц (стандарт HSTL IV). Кристаллы серии Virtex за счёт развитой технологии производства и усовершенствованного процесса верификации имеют достаточно низкую стоимость (до 40% от эквивалентной стоимости серии XC4000XL).

Помимо семейства Virtex, Xilinx выпускает FPGA семейств XC3000A, XC4000E, Spartan, XC5200, а также CPLD XC9500 и малопотребляющую серию CoolPLD.

Существует бесплатная версия САПР — WebPACK, поддерживающая CPLD XC9500 и CoolPLD, ввод описания алгоритма с помощью языка описания аппаратуры VHDL. ПЛИС Xilinx выпускаются в коммерческом и индустриальном диапазоне температур с военной (Military) и космической (Space) приёмкой.

1.1. Пакеты САПР для разработки схем с программируемой логикой

Применение САПР требует эффективных, наглядных, управляемых и контролируемых средств описания проекта. Описать проектируемое устройство можно разными способами, причем обычно применяют способ, пригодный для описания проекта в целом. В настоящее время к наиболее распространенным универсальным способам описания, применимым для любого уровня иерархии проекта, относят *графический и текстовый.* Реже используются непосредственная разводка схем FPGA в редакторе топологии, описания в виде требуемых временных диаграмм и др. Каждый из способов описания проекта имеет свои достоинства и недостатки.

Графическое представление проекта создается в базисе допустимых для выбранной САПР библиотечных элементов, например, в базисе элементов стандартной серии ТТЛ(Ш). Главное достоинство графического способа – его традиционность и наглядность, связанные с привычностью разработчиков к восприятию изображений схем.

Современные языки описания аппаратуры (HDL, Hardware Description Languages) допускают описание проектируемого устройства как с точки зрения его *поведения,* так и с точки зрения его *структуры.*

При выборе вида описания схемы можно дать следующие рекомендации: ввод схем в виде элементов стандартной ТТЛ серии 54/74 оправдан, если имеется отработанный образец устройства, требующий миниатюризации, а также, если схема относительно проста, не содержит сложных алгоритмов работы и обратных связей (дополнительной проблемой может быть отсутствие VHDL–модели в библиотеке при копировании схем из других проектов); ввод схемы в виде VHDL –описания оправдан в случае применения в разрабатываемом устройстве сложных алгоритмов ЦОС, сложных комбинационных устройств, элементов ОЗУ, массивов, т.е. в случае, когда степень интеграции ПЛИС существенно возрастает. Язык описания VHDL является формальной записью, которая может быть использована на всех этапах разработки цифровых электронных систем. Это возможно вследствие того, что язык легко воспринимается как машиной, так и человеком. Он может использоваться на этапах проектирования, верификации, синтеза и тестирования аппаратуры так же, как и для передачи данных о проекте, модификации и сопровождения.

Рассмотрим основные этапы процесса проектирования ПЛИС, которые являются общими практически для любых пакетов САПР:

1. **Графический ввод схемы**. Ввод схемы возможен как в виде отдельных компонентов, так и иерархических блоков, внутренним содержанием которых может быть VHDL –код. Пакет САПР также обеспечивает ввод схемы на языке описания высокого уровня.
2. **Подготовка данных о проекте для реализации ПЛИС**. На этом этапе создается список соединений, простановка позиционных обозначений и проверка ошибок в электрической схеме.
3. **Функциональное моделирование схемы**. На начальном этапе проверяется логика работы схемы; задержки элементов не учитываются. Для моделирования обычно требуется создать или загрузить файл входных воздействий.
4. **Компиляция описания ПЛИС**. В результате синтеза составляется список соединений проекта на основании составленной принципиальной схемы и VHDL –описаний. В результате оптимизации производится упрощение исходной схемы для улучшения физической реализации в виде ПЛИС с учетом ряда ограничений.

5. **Размещение и трассировка ПЛИС**. В результате будет создан файл для программирования ПЛИС, а исходная схема будет размещена в выбранном кристалле. Для дальнейшего моделирования создается файл описания работы ПЛИС на языке **VHDL**. В этом файле прописываются реальные временные характеристики выбранной микросхемы.
6. **Временное моделирование** работы реальной ПЛИС с учетом реальных временных задержек.
7. **Создание графического символа разработанной ПЛИС** для дальнейшего моделирования с другими цифровыми схемами в данном проекте или других проектах.

Рассмотрим более подробно основные компоненты пакета САПР фирмы ALTERA MAX+PLUS II. Название системы MAX+PLUS II является аббревиатурой от **M**ultiple **A**rray Matri**X** **P**rogrammable **L**ogic **U**ser **S**ystem (Пользовательская система программирования логики упорядоченных структур). Система MAX+PLUS II обеспечивает многоплатформенную архитектурно независимую среду создания схем, имеет средства удобного ввода схем, быстрого прогона и непосредственного программирования устройств.

Полный спектр возможностей логического проектирования включает разнообразные средства описания проекта для создания проектов с иерархической структурой, мощный логический синтез, компиляцию с заданными временными параметрами, разделение на части, функциональное и временное тестирование (симуляцию), тестирование нескольких связанных устройств, анализ временных параметров системы, автоматическую локализацию ошибок, а также программирование и верификацию устройств. В системе MAX+PLUS II можно как читать, так и записывать файлы на языке AHDL и файлы трассировки в формате EDIF, файлы на языках описания аппаратуры Verilog HDL и VHDL, а также схемные файлы OrCAD. Кроме того, система MAX+PLUS II читает файлы трассировки, созданные с помощью ПО Xilinx, и записывает файлы задержек в формате SDF для удобства взаимодействия с пакетами, работающими с другими промышленными стандартами.

В полную систему MAX+PLUS II входят 11 полностью внедренных в систему приложений (рис. 3). Сложный иерархический проект образуют схемы (дизайны) и подсхемы (субдизайны).

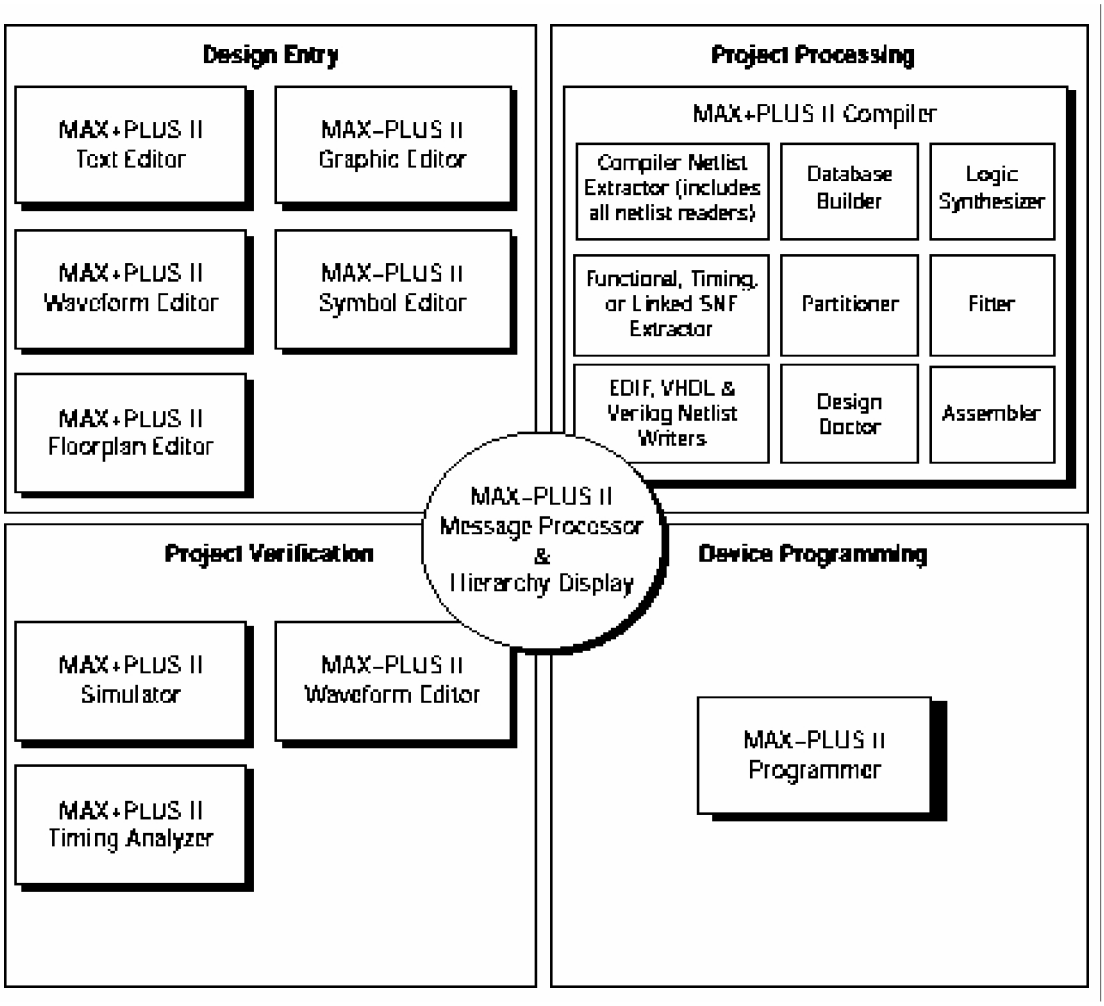

Рис. 3. Приложения в системе MAX+PLUS II

Для ввода описания проекта (Design Entry) возможно описание проекта в виде файла на языке описания аппаратуры, созданного либо во внешнем редакторе, либо в текстовом редакторе MAX+PLUS II (Text Editor), в виде схемы электрической принципиальной с помощью графического редактора Graphic Editor, в виде временной диаграммы, созданной в сигнальном редакторе Waveform Editor. Для удобства работы со сложными иерархическими проектами каждому субдизайну может быть сопоставлен символ, редактирование которого производится с помощью графического редактора Symbol Editor. Размещение узлов по логическим блокам и выводам ПЛИС выполняют с помощью поуровневого планировщика Floorplan Editor.

Компиляция проекта, включая извлечение списка соединений (Netlist Extractor), построение базы данных проекта (Data Base Builder), логический синтез (logic synthesis), извлечение временных, функциональных параметров проекта (SNF Extractor), разбиение на части (Partioner),

трассировку (Fitter) и формирование файла программирования или загрузки (Assembler), выполняется с помощью компилятора системы (Compiler).

Наиболее важные этапы компиляции:

Модуль логического синтезатора (**Logic Synthesizer)** применяет ряд алгоритмов, который уменьшает использование ресурсов и убирает дублирующую логику, обеспечивая тем самым максимально эффективное использование структуры логического элемента для архитектуры целевого семейства устройств.

Если проект не помещается при монтаже в одно устройство, модуль **Partitioner** (разделитель) разделяет базу данных, обновленную логическим синтезатором, на несколько ПЛИС одного и того же семейства, стараясь при этом разделить проект на минимально возможное число устройств. Разбиение проекта происходит по границам логических элементов, а число выводов, используемое для сообщения между устройствами, минимизируется.

Используя базу данных, обновленную модулем разбиения, модуль трассировки **(Fitter)** приводит в соответствие требования проекта с известными ресурсами одного или нескольких устройств. Он назначает каждой логической функции расположение реализующего ее логического элемента и выбирает соответствующие пути взаимных соединений и назначения выводов.

Functional SNF Extractor (экстрактор для функционального тестирования)создает файл для функционального тестирования (**.snf**). Этот функциональный файл SNF не содержит информации о временных параметрах.

Timing SNF Extractor (экстрактор для тестирования временных параметров) создает (если компиляция проекта прошла без ошибок) файл для тестирования временных параметров (**.snf**), который содержит данные о временных параметрах проекта. Этот файл используется для тестирования и анализа временных параметров.

Assembler (модуль ассемблера) - преобразует назначения логических элементов, выводов и устройства, сделанные модулем трассировки Fitter, в программный образ для устройства (устройств) в виде одного или нескольких двоичных объектных файлов, содержащих информацию для программатора, которая обрабатывается программатором системы MAX

PLUS II и программирующей аппаратурой фирмы Altera (или другим программатором).

Верификация проекта (Project verification) выполняется с помощью симулятора (simulator), результаты работы которого удобно просмотреть в сигнальном редакторе Waveform Editor, в нем же создаются тестовые воздействия.

В САПР MAX + PLUS II предусмотрено автоматическое вычисление трех основных классов временных параметров (**модуль Timing Analyzer**):

- минимальных и максимальных задержек между источниками (входными сигналами) и приемниками (выходными сигналами), информация о которых выдается в виде матрицы задержек;
- максимально возможной производительности устройства (пропускной способности) в виде максимальной частоты тактирования элементов памяти, используемых в проекте;
- времен предустановки и выдержки сигналов, гарантирующих надежную работу схем при фиксации сигналов в синхронных элементах памяти.

Программирование ПЛИС. Непосредственное программирование или загрузка конфигурации устройств с использованием соответствующего аппаратного обеспечения выполняется с использованием модуля программатора **(Programmer).**

2. СПОСОБЫ ИЕРАРХИЧЕСКОГО ЗАДАНИЯ ПРОЕКТА

В пакете **MAX+PLUS® II** предусмотрено несколько способов описания исходной схемы: графический ввод в виде стандартных микросхем популярных **TTL** серий **74LS** или встроенных цифровых примитивов, текстовое описание на языках **AHDL**, **VHDL**, **Verylog HDL**, описание работы схемы на уровне временных диаграмм, связывающих входные и выходные сигналы. В одном проекте возможно сочетание любых вышеуказанных способов, в результате чего разработчику очень легко манипулировать вариантами построения исходной схемы и гибко выбирать наиболее удобный метод задания проекта. Эта возможность обусловлена тем, что при любом методе ввода отдельных узлов схемы пакет **MAX+PLUS® II** после компиляции формирует файл графического символа разработанного блока и соответствующий ему код на языке **AHDL** (или **VHDL**). И соответственно из этих фрагментов текстового кода формируется итоговый файл (также, разумеется, на языке **AHDL** (или **VHDL**)), который является верхним на уровне иерархии и вбирает в себя все созданные ранее и отлаженные узлы большого проекта. Это позволяет вести разработку сложного проекта последовательно, поэтапно дополняя и усложняя схему отлаженными модулями, или разбить проект на отдельные простые части, каждую из которых можно вводить тем способом, который наиболее подходит, или готовить проект группой разработчиков, которые могут использовать удобные для себя способы описания схем.

Рассмотрим вышесказанное на конкретном примере и поясним назначение элементов входящих в иерархическую структуру проекта. Из рис.4 видно, что в папке рабочих документов **C:\max2work** пакета **MAX+PLUS® II** создана папка проекта **t1**, а в ней графический схемный файл **t1.gdf**, который является верхним или результирующим файлом в иерархии проекта и как контейнер вбирает в себя другие узлы.

Рекомендуется результирующему файлу давать то же имя, что и папка проекта.

Это позволяет отличить главный файл от вспомогательных в проекте, а также правильно выполнить дальнейшую компиляцию и верификацию проекта. Чтобы посмотреть иерархию проекта после загрузки или создания результирующего файла, нужно выполнить

команду **MAX+PLUS II/ Hierarchy Display** или нажать на кнопку меню. В открывшемся окне тонкими стрелками показаны иерархические связи между файлами в проекте. Двойное нажатие на иконку файла открывает соответствующий редактор для внесения изменений или просмотра. Толстыми стрелками показаны открытые окна соответствующих редакторов пакета **MAX+PLUS II**.

Из рис.4 видно, что результирующий файл **t1.gdf,** открытый в графическом редакторе, содержит собственные цифровые примитивы – это схемы **"Исключающего ИЛИ OR2"** и **"Инвертор NOT"**, а также подсхемы **lg, l2, s2,** представленные символами, потому что они разработаны как самостоятельные устройства и объединены в проекте **t1**.

Важно отметить, что файлы этих устройств также находятся в папке t1.

Устройство **lg.gdf** представлено графическим файлом схемы, содержащим цифровые примитивы (двухвходовая схема **"Логического И AND2"**) из встроенной базы данных элементов пакета **MAX+PLUS II**.

Устройство **l2.tdf** представлено текстовым файлом описания схемы на языке **AHDL**. Это код для комбинационной логики, выполняющей логическую функцию **И** с инверсией для входных сигналов **X3** и **X4** и выходного сигнала **Z2**.

Устройство **s2.wdf** представлено файлом временных диаграмм, в котором в графическом виде задана логика работы. Нетрудно видеть, что выходной сигнал **w** является инверсией входного сигнала **q**. Хотя логика работы этого блока определена на участке 100ns, компилятор считает, что эта логическая функция выполняется в любой момент времени.

Встроенные в проект подсхемы могут сами содержать в себе другие подсхемы. Пример такого вида иерархии показан на рис.5. Таким образом, просматривая иерархические связи в окне проекта, можно легко ориентироваться в сложной структуре разрабатываемой схемы и быстро модифицировать ее, выполняя замену блоков, сочетая различные способы описания.

Рассмотрим назначение других файлов в иерархическом дереве проекта **t1**, рис.5. Файл **t1.scf** содержит временные диаграммы тестирования работы проекта. Результаты моделирования с учетом временных задержек представлены на рис.6.

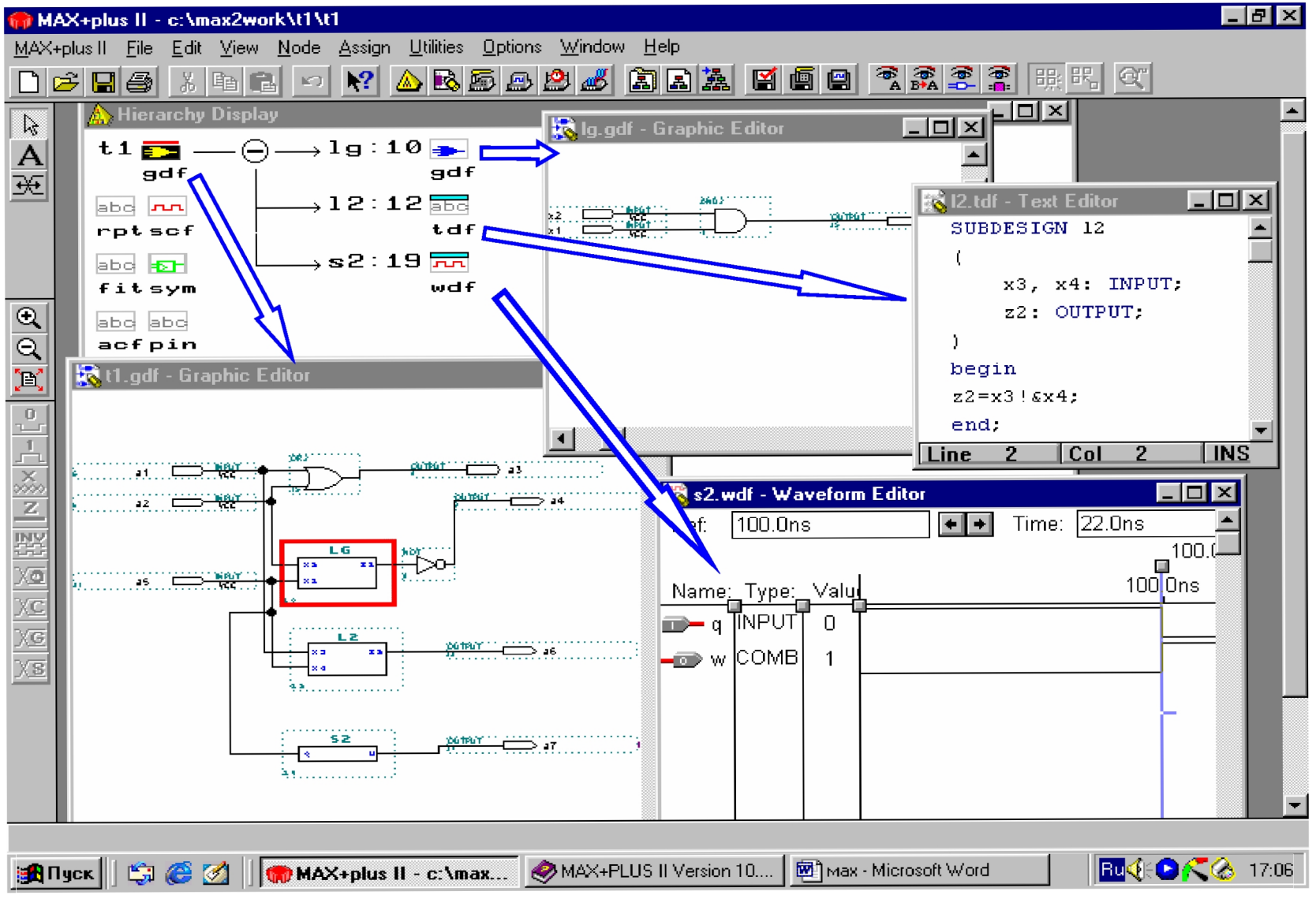

Рис. 4 . Окно иерархического проекта

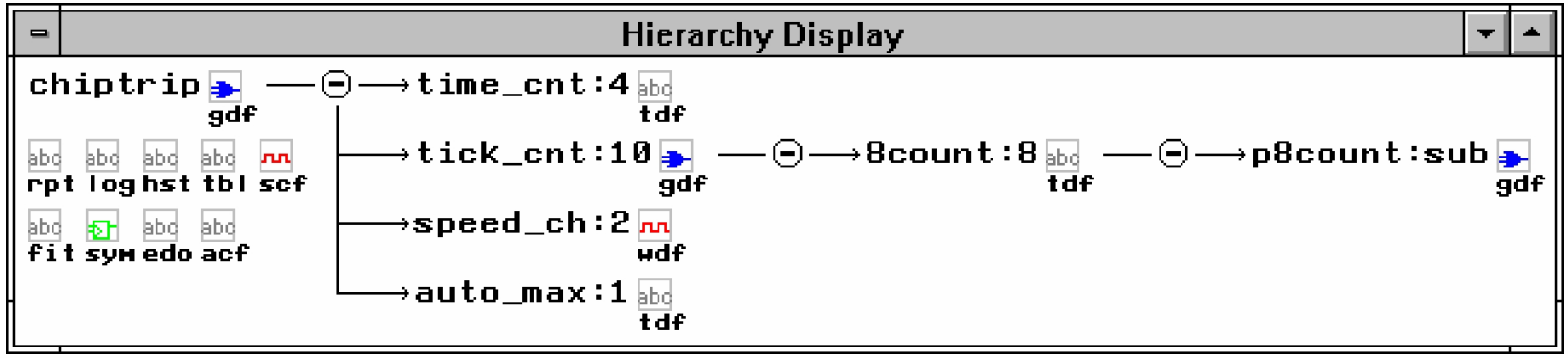

Рис. 5. Пример сложной иерархии схем в проекте

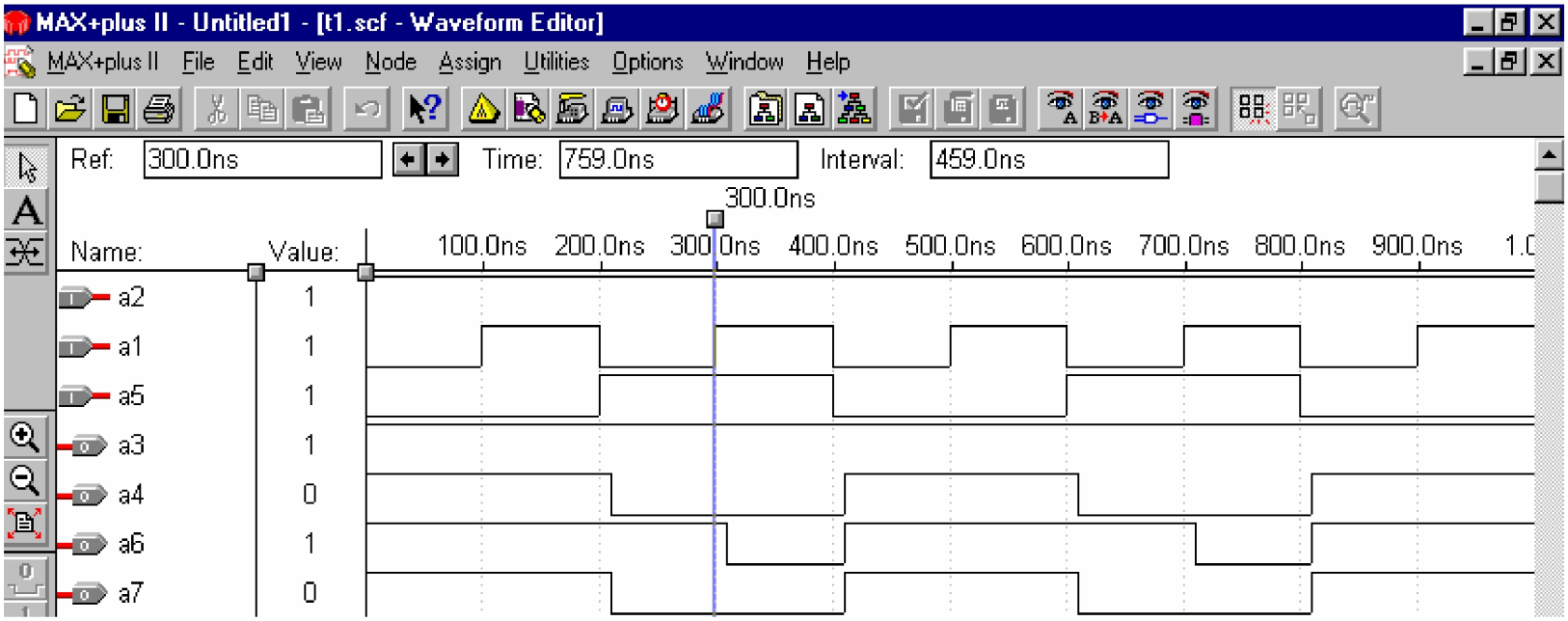

Рис. 6. Результаты временного моделирования

Файл **t1.sym** (рис.7) содержит графический символ проекта, который может быть использован как подсхема в других проектах. С ним будет связан код логики работы блока **t1** на языке описания **AHDL**, получаемый в ходе компиляции проекта. Файл **t1.acf** (см. рис.5) содержит информацию об управлении проектом при выполнении всех стадий проектирования (компиляция, моделирование верификация и т.д.) и их связи с изменением иерархического дерева.

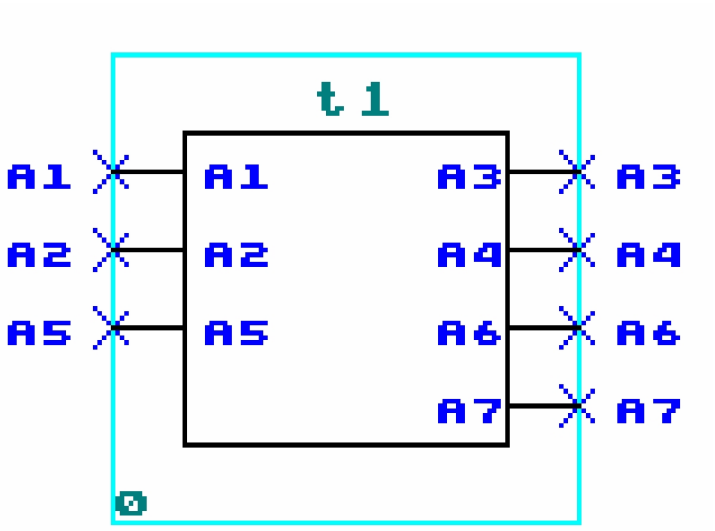

Рис. 7. Символ компонента

Файл **t1.fit** (рис. 8) содержит информацию о размещении разработанного проекта в выбранном устройстве ПЛМ.

Файл **t1.pin** (рис. 8) содержит информацию о расположении контактов микросхемы ПЛМ, выбранных компилятором в ходе проектирования. Файл **t1.rpt** (см. рис. 5) содержит информацию в виде отчета о всех стадиях процесса проектирования устройства. Теперь можно приступить к изучению процесса разработки схем с ПЛИС. Рекомендуется повторить действия, рассмотренные в следующем разделе.

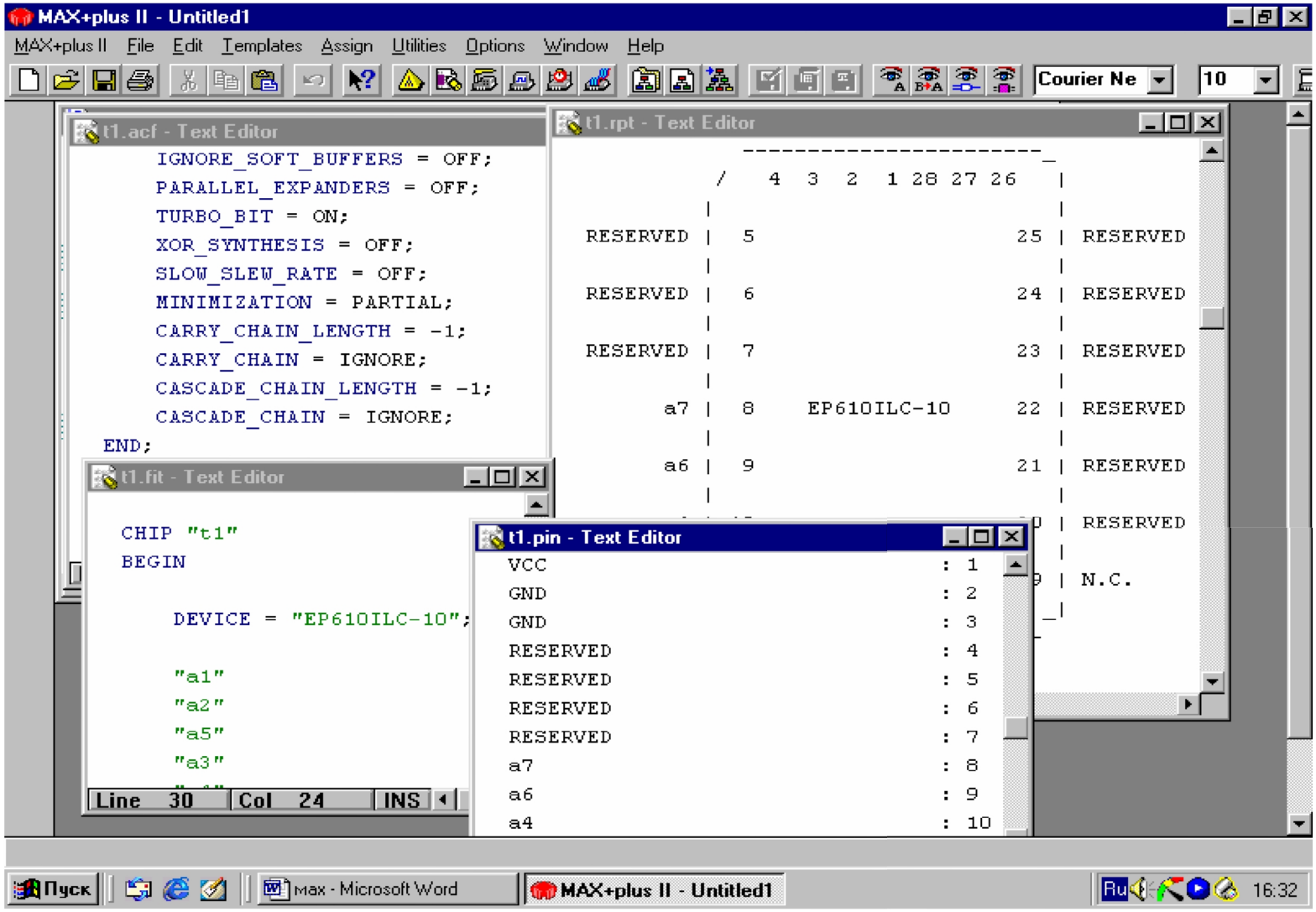

Рис. 8. Состав проекта

3. ПРИМЕР РАЗРАБОТКИ ПЛИС

Рассмотрим подробно ход проектирования устройств с программируемой логикой. Повторение указанных действий позволяет получить представление об основном цикле проектирования ПЛИС средствами пакета **MAX+PLUS II**. Повторив эти действия и поняв смысл проделанных операций, Вы можете приступить к разработке собственных ПЛИС. Для примера выбрана исходная схема средней сложности, содержащая как простые комбинационные схемы, так и последовательную логику (триггеры). Для тестирования схемы будут созданы входные сигналы разных типов. Таким образом, этот пример достаточно полно раскрывает особенности использования пакета **MAX+PLUS II**. Для описания проекта выбраны графический ввод схемы и описание на языке программирования **AHDL.** Вначале рассмотрим работу устройства.

3.1 Схема формирователя сигнала кода Баркера

Временная диаграмма 5-позиционного сигнала кода Баркера (в уровнях ТТЛ-схем) представлена на рис.9.

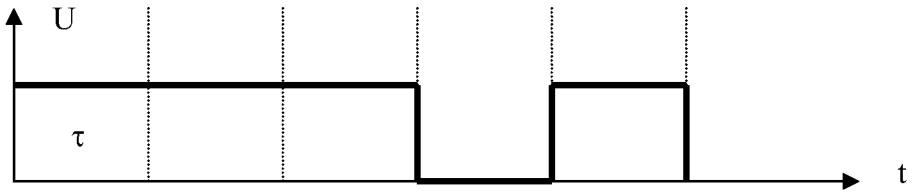

Рис. 9. Вид сигнала кода Баркера

Очевидно, что существует несколько способов формирования сигнала, представленного на рис.9. Самый простой метод для понимания принципа формирования кода Баркера представлен на рис.10. Одиночный импульс START поступает на вход регистра сдвига, образованного цепочкой последовательно включенных триггеров D-типа. Число триггеров равно числу позиций кода Баркера. Длительность входного импульса не должна превышать длительности одной позиции кода Баркера τ. Сдвиг импульса по цепочке обеспечивается сигналом синхронизации CLOCK, фронт которого последовательно будет переписывать входной сигнал из одного триггера в другой. Период повторения сигнала синхронизации равен длительности одной позиции кода Баркера. Логическая схема должна объединять сигналы с уровнем логической единицы с соответствующих выходов триггеров, чтобы обеспечить формирование требуемой последовательности.

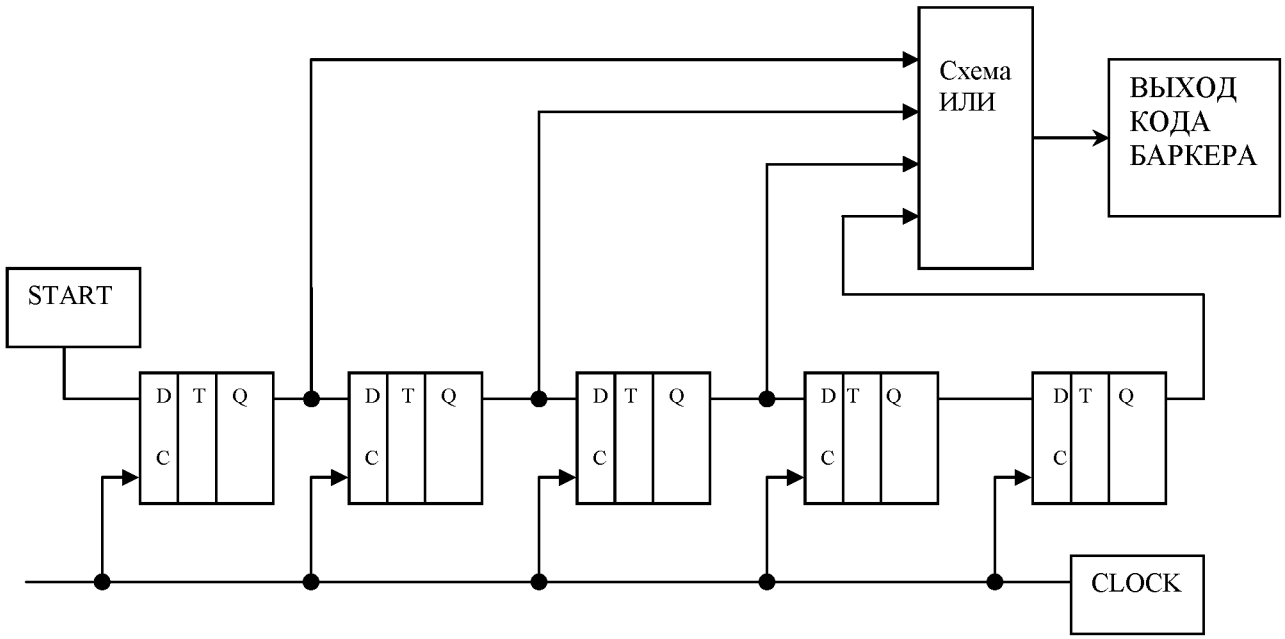

Рис. 10. Схема формирования сигнала кода Баркера при помощи D-триггеров

3.2. Ввод проекта в графическом редакторе

В рабочей папке проектов пакета **MAX+PLUS II** (обычно это **C:\max2work**) нужно создать папку **barker**. В ней будут храниться все файлы иерархического проекта. Часть схемы формирователя, содержащая триггеры, будет введена графическим файлом. Запускаем управляющую оболочку **MAX+PLUS II manager** и выбираем команду **File/New…**, в окне диалога **File Type** (рис. 11) указываем на файл графического редактора **Graphic Editor file** и его тип **.gdf** . После открытия окна графического редактора сохраняем будущий файл схемы под именем **registr.gdf** в папке **C:\max2work\ barker** (рис. 11) по команде **File/Save As…**. (Графический редактор также можно запустить из управляющей оболочки по команде **MAX+PLUS II/Graphic Editor**.)

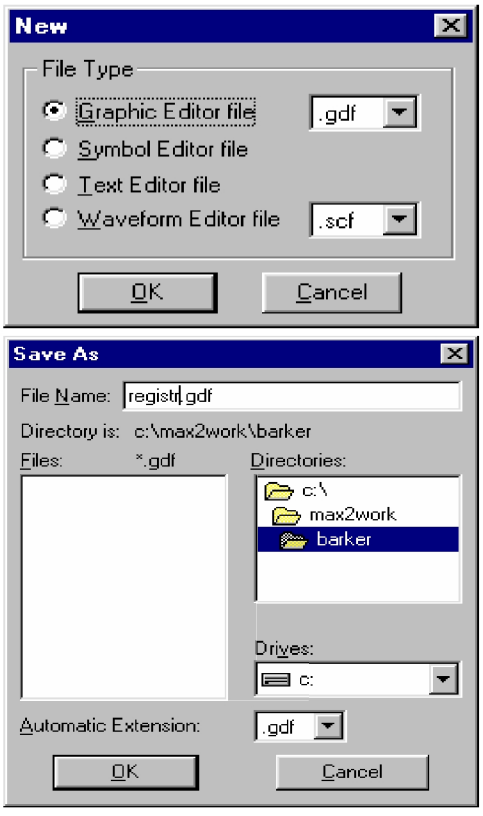

Рис. 11. Начальный диалог

Приступаем к размещению элементов. По команде **Symbol/Enter Symbol…** или двойным щелчком в пустом месте окна графического редактора открываем диалог ввода компонента **Enter Symbol** (рис. 12). В

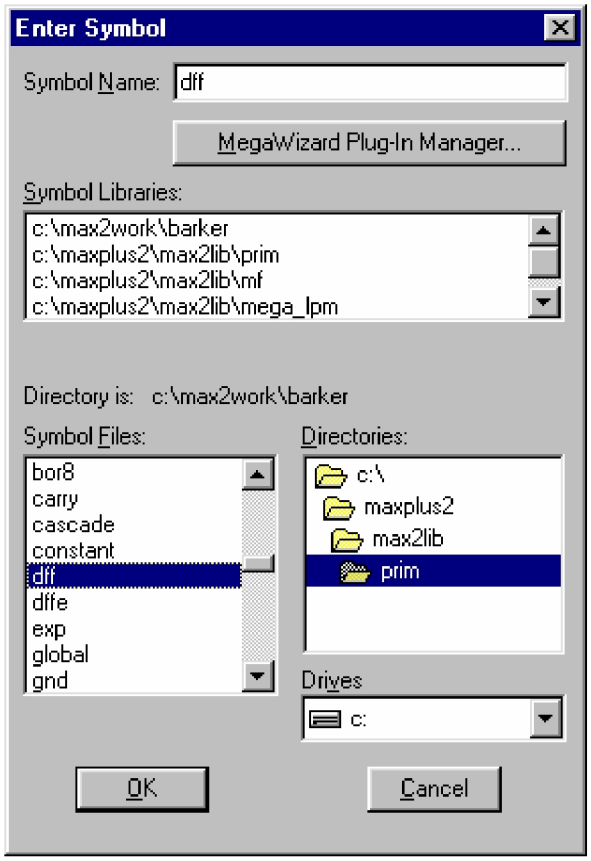

Рис. 12. Выбор элементов из библиотеки

окне библиотек **Symbol Libraries** выбираем библиотеку цифровых примитивов: **C:\maxplus2\max2lib\prim**, а в окне **Symbol Files** D- триггер **dff**.

Элемент размещаем на схеме, нажимая на **ОК** и используя стандартные команды копирования (**Edit/Copy**) –  вырезания (**Edit/Cut**) – вставки (**Edit/Paste**), формируем пять триггеров, расположенных один за другим.

Размещаем проводники. Для этого подводим перекрестье курсора к выводу одного элемента и проводим мышкой проводник до вывода другого элемента. Для удаления проводника его выделяют и из контекстного меню (правая клавиша мышки) выбирают **Delete**. Точка соединения появляется автоматически, если остановить движение мышки при пересечении проводников. Ошибки можно исправить, расставляя или убирая точки соединения при помощи  инструментов **Toggle Connection Dot**.

Размещаем контакты для входных и выходных сигналов. В данном случае потребуются выходы каждого триггера. Символы контактов находятся в той же библиотеке, что и символы триггеров - **C:\maxplus2\max2lib\prim**, их имена: **input** для входных сигналов и **output** для выходных.

Сохраняем схему по команде **File/Save**. Для продолжения разработки желательно сразу выбрать конкретный тип микросхемы ПЛМ, в которую будет разведена схема. Это назначение происходит по команде **Assign Device…** (рис.13). В окне выбора семейства **Device Famaly** укажем **MAX3000A**, а в окне выбора устройства в семействе укажем **AUTO**, чтобы компилятор сам выбрал подходящий вариант.

Разумеется, можно указать тот кристалл, которым располагает разработчик. Так как данный пример учебный, то не будем указывать другие ограничения. В реальной разработке можно ввести допустимые времена

задержек, максимальные рабочие частоты для всего проекта или его части, уровни логических сигналов и т.д. в соответствии с меню команды **Assign.**

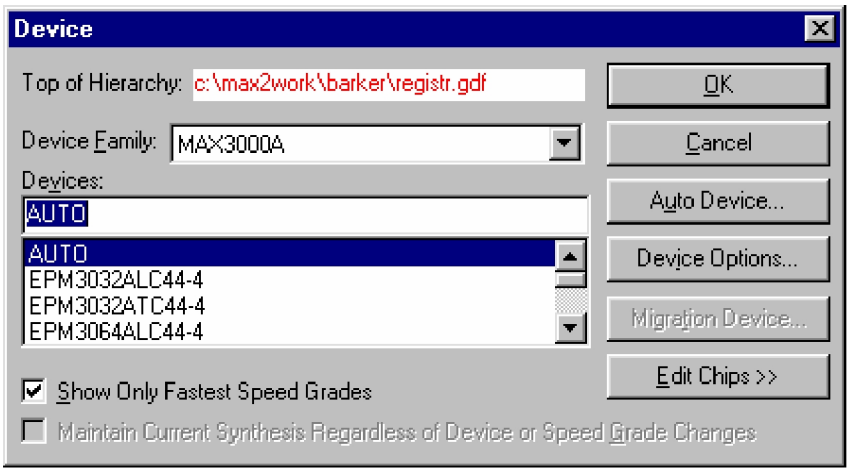

Рис. 13. Назначение типа микросхемы

В сложном иерархическом проекте может быть много описаний различных узлов (входных дизайнов). Для дальнейшей обработки нужно указать, какой элемент будет обрабатываться программами пакета **MAX+PLUS II**, например компилятором. Для этого **обязательно** нужно выполнить команду **File/Project/Set Project to Current File** для установки файла, который будет главным. Эту команду нужно выполнять всякий раз, когда происходит переход к обработке другого элемента проекта, иначе будет выполняться повторная обработка.

3.3. Функциональное моделирование

Функциональное моделирование позволяет проверить логику работы устройства без учета временных задержек в кристалле. Необходимо запустить компилятор командой **MAX+PLUS II/ Compiler** или нажать на кнопку пиктограмму панели инструментов. Обратите внимание, что меню команд изменилось – теперь это команды управления компилятором. Устанавливаем режим функционального моделирования командой **Processing/Functional SNF Extractor**. Для запуска компиляции нужно нажать на кнопку **START**. Процесс компиляции должен завершиться без ошибок (рис. 14). В противном случае нужно смотреть .log-файл сообщений об ошибках и исправить их. Окно компилятора можно пока закрыть.

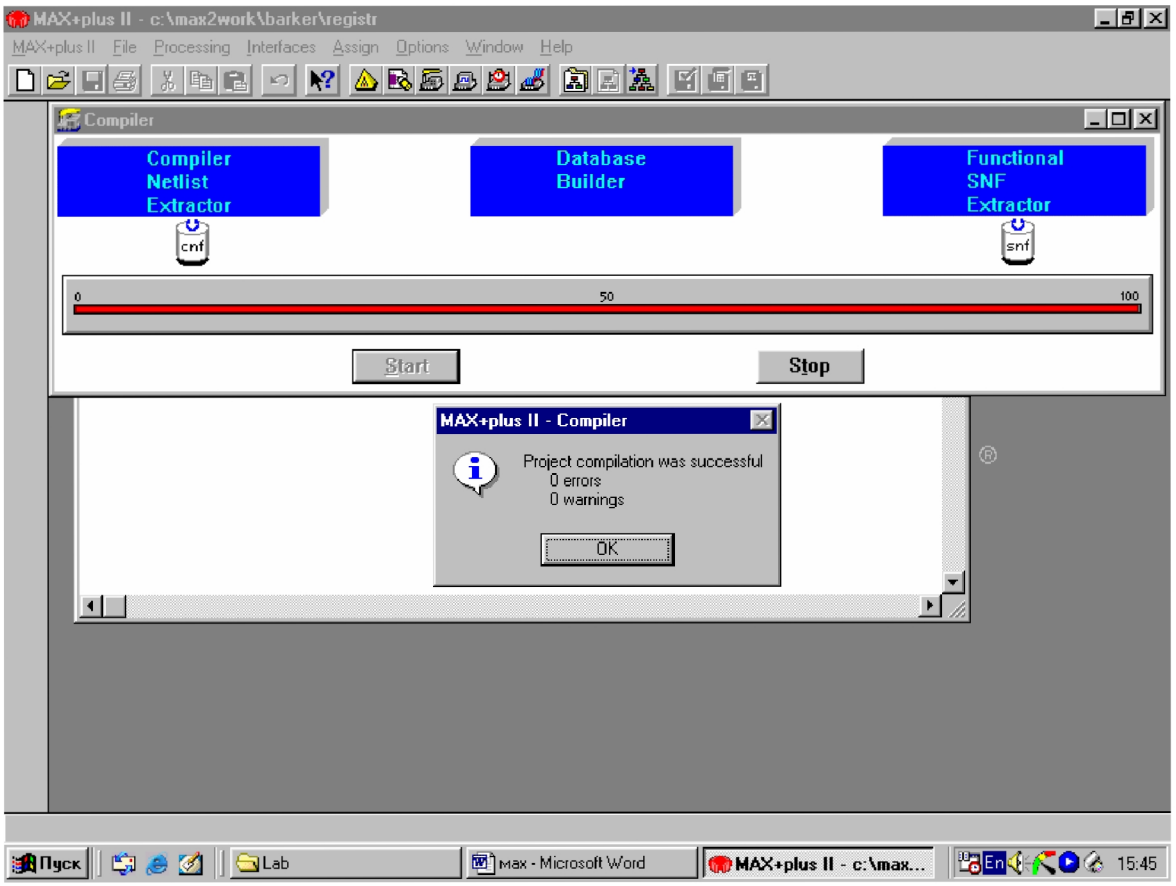

Рис. 14. Компилятор в режиме функционального моделирования

3.4. Верификация

Для проверки проекта нужно сформировать файл входных воздействий. Временные диаграммы, созданные в файле типа **.scf,** будут тестировать схему. Для запуска редактора временных диаграмм **Waveform Editor** можно использовать команду **File/New…** и выбрать в окне **File Type** тип **Waveform Editor file .scf**. После открытия окна редактора сохраняем файл в папке **barker** под **registr.scf** именем по команде **File/Save As…** (рис. 15). (Данный редактор также можно запустить из управляющей оболочки по команде **MAX+PLUS II/ Waveform Editor**).

В окне редактора диаграмм появляется ось времени, на которой будут сформированы графики входных сигналов. Этот же редактор будет использоваться для просмотра результатов моделирования. Поэтому расположим в его окне временные диаграммы всех сигналов схемы. Для этого в области редактора ниже заголовка правой клавишей мыши выберем команду **Enter Nodes from SNF…** (вызов из меню – **Node/ Enter Nodes from SNF…**). Данная команда автоматически переносит узлы из созданной схемы (неважно какого вида – графического или текстового) в редактор диаграмм.

Примечание. Если не выполнена команда **File/Project/Set Project to Current File**, то соответственно команда **Enter Nodes from SNF…** будет **недоступна**.

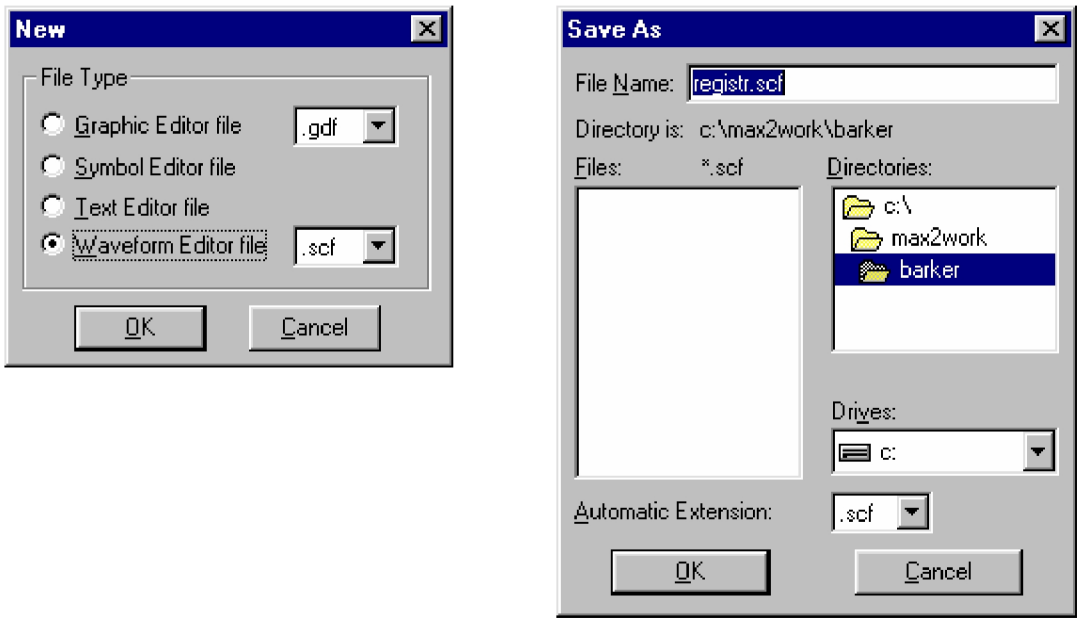

Рис. 15. Создание тестового файла

Для индивидуальной работы с узлами можно использовать команду **Node/Enter Node…**, а также узлы можно сортировать по имени **Node/Sort Names…**, редактировать **Node/Edit Node…**, объединять в группы **Node/Enter Group…** и разгруппировывать **Node/Ungroup**.

В окне **Enter Nodes from SNF** выбираем из списка **List** все узлы в окне **Available Nodes & Groups:** доступных узлов и кнопкой ⇒ переносим их в окно выбранных узлов **Selected Nodes & Groups:**. Остальные настройки обычно оставляют без изменений.

При нажатии на **OK** все узлы появляются в окне редактора диаграмм (рис. 16). Приступаем к редактированию сигналов в графическом режиме. По умолчанию начальные значения всех сигналов равны логическому нулю.

Сигнал **Start** должен иметь длительность более половины периода сигнала **Clock** , чтобы сработал первый триггер, но меньше длительности периода, чтобы не было повторного срабатывания триггера. Левой клавишей мыши выделяем область времени от 0 ns до примерно 60-70 ns (значения можно контролировать в окнах **Start**, **End**, и **Interval**). Нажимаем на кнопку установки выделенного фрагмента в логическую единицу (или правая клавиша мыши - команда **Overwrite/High [1]**) (рис. 17).

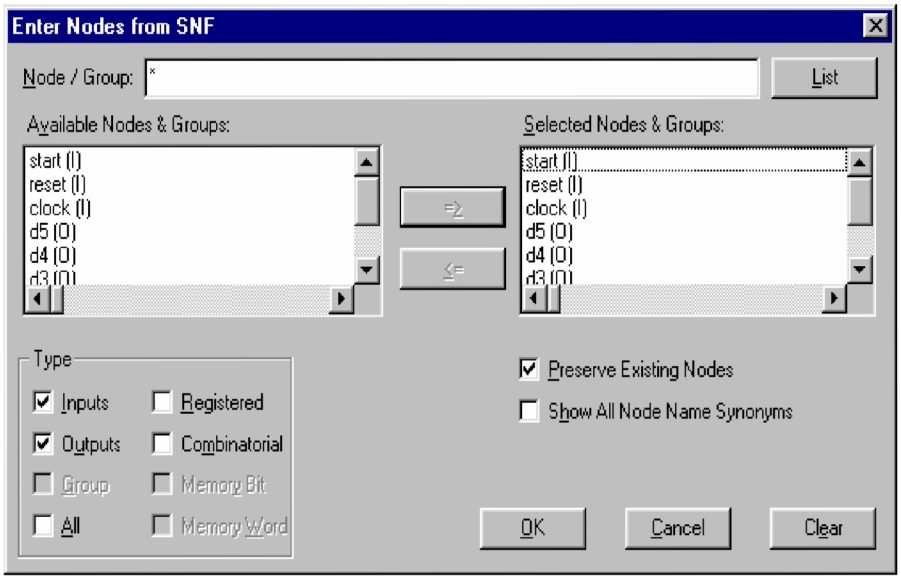

Рис. 16. Выбор узлов

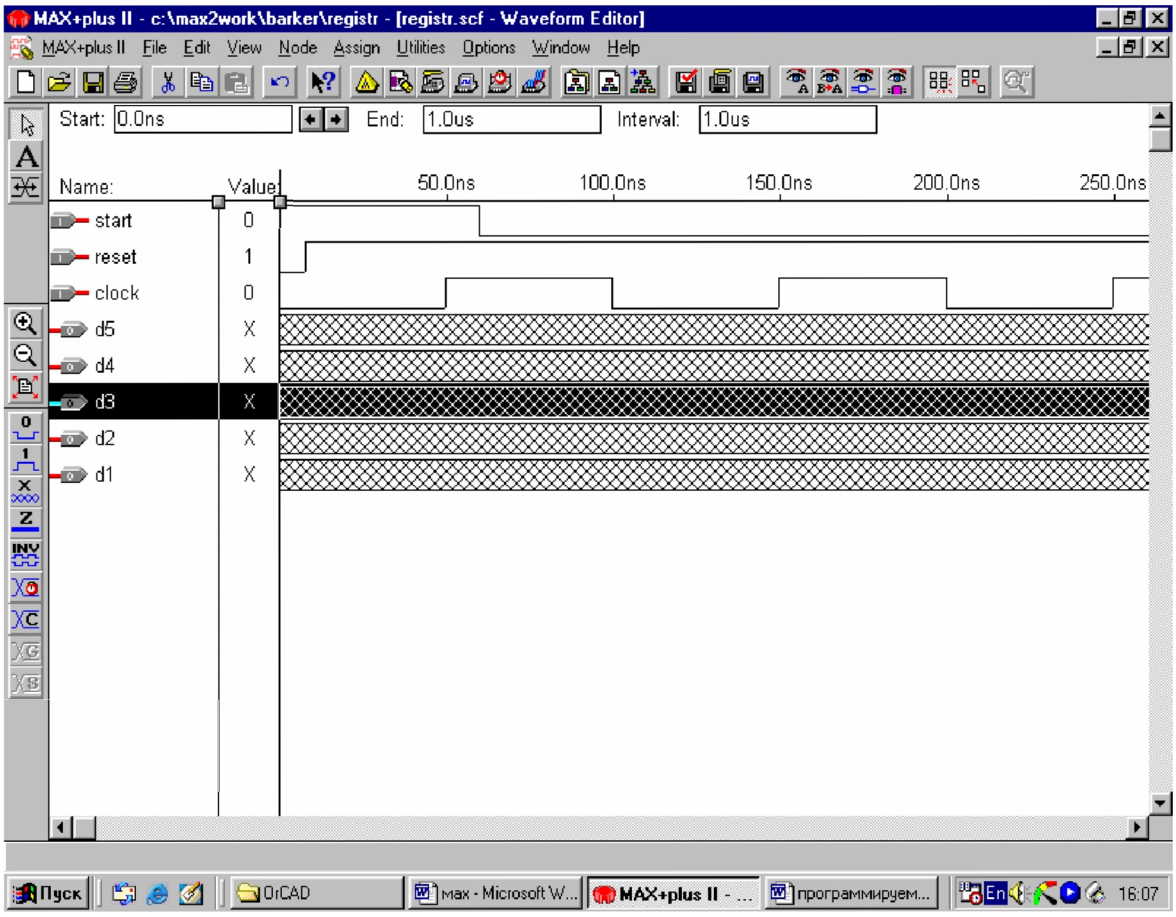

Рис. 17. Редактор временных диаграмм

Сигнал начальной установки **reset** должен быть равен 0 вначале, а затем постоянно 1. Сначала инвертируем его. Для этого выделим весь сигнал (левой клавишей мыши по имени сигнала) и перепишем его значение на 1.

Затем выделим участок от 0 ns до примерно 6-8 ns и перепишем его значение на логический 0 (правая клавиша мыши - команда **Overwrite/Low [0]**.

Сигнал синхронизации **Clock** - это периодическая последовательность прямоугольных импульсов с периодом 100 ns. Выделяем весь сигнал и заполняем значения сигнала **Clock** в окне **Overwrite Clock,** как показано на рис. 18 (правая клавиша мыши - команда **Overwrite/Clock …**). Начальное значение **Starting Value** выбрано равным 0, а период задан множителем **Multiplied By** равным 100. Сигнал задан в интервале от 0 до 1 μs. Сохраняем результаты проектирования и закрываем редактор диаграмм.

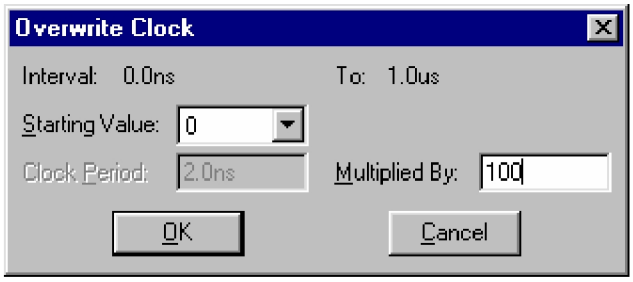

Рис. 18. Установка сигнала Clock

Для запуска симулятора функционального моделирования выполняем команду **MAX+PLUS II /Simulator**, или нажимаем кнопку пиктограммы на панели инструментов, или если открыто окно компилятора, то дважды щелкаем мышкой по объекту **Functional SNF Extractor**. В открывшемся окне **Simulator: Functional Simulation** (рис. 19) указан файл входных воздействий в строке **Simulation Input**, а время начала и конца анализа можно изменять в строках **Start Time** и **End Time** соответственно. Для запуска симулятора нужно нажать на кнопку **Start**, а по окончании убедиться в успешном завершении процесса моделирования.

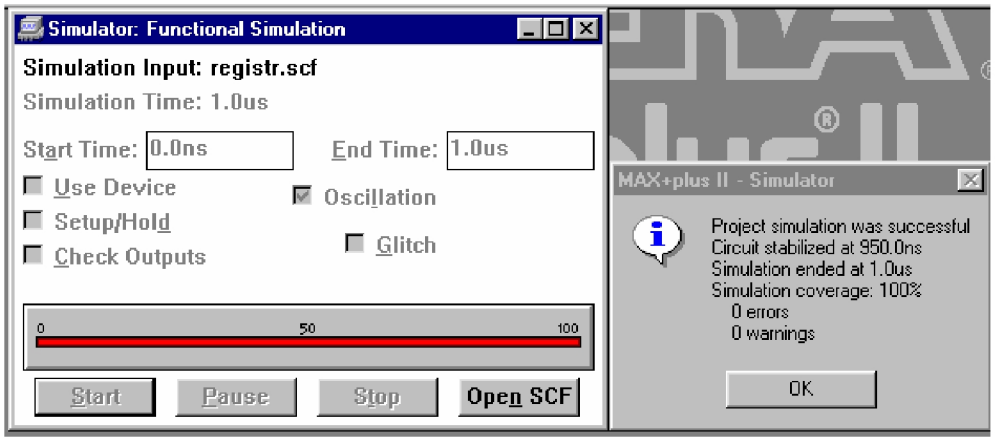

Рис. 19. Окно функционального моделирования

Для просмотра результата нужно нажать на кнопку **Open SCF**. Запускается редактор временных диаграмм **Waveform Editor,** и в его окне просматриваем входные и выходные сигналы. Для удобства просмотра их лучше сгруппировать (переместить мышкой) так, как показано на рис. 20. Здесь отчетливо виден эффект работы регистра сдвига по фронту сигнала синхронизации, что говорит о правильной разработке схемы.

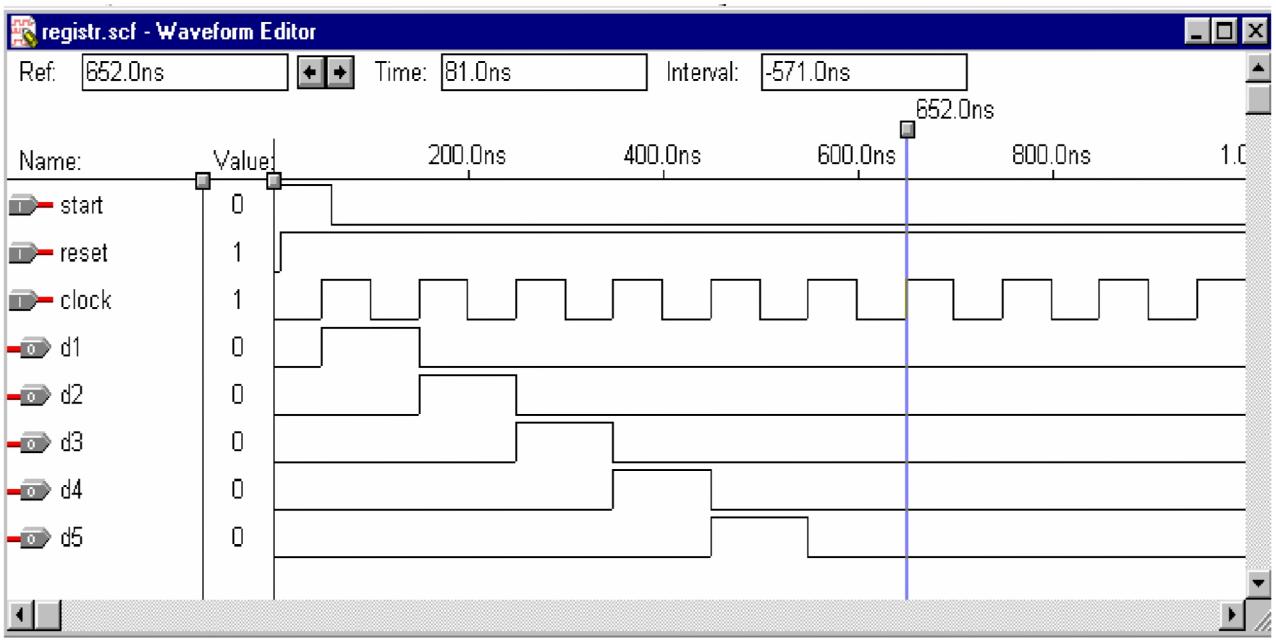

Рис. 20. Просмотр результатов моделирования

Масштаб изображения можно регулировать кнопками:  и  .

Теперь можно закрыть все окна, кроме окна графического редактора. Для использования данной схемы как подсхемы (субдизайн) в иерархическом проекте формирователя кода Баркера нужно создать графическое символьное обозначение, на которое будет идти ссылка в проекте. Данная операция выполняется по команде **File/Create Default Symbol**. Результат представлен на рис. 21. Символьный файл будет храниться в папке **barker** и доступен для дальнейшей разработки. Окно символьного редактора можно закрыть, при необходимости изменения сохранить в файле под именем **registr.sym**.

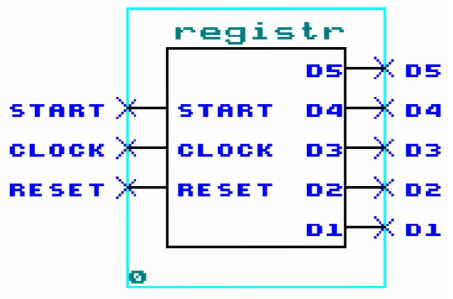

Рис. 21. Символ компонента

3.5. Ввод схемы в текстовом редакторе

Логическая схема, формирующая сигнал кода Баркера, будет введена в проект на языке описания **AHDL**. На основе этого примера можно создавать собственные несложные комбинационные схемы. Данный пример показывает, что нет ничего сложного в составлении описания схемы на языке **AHDL**, а при описании сложных устройств цифровой обработки рекомендуется не использовать схемные аналоги **TTL** –серий 74xx, так как их громоздкий **AHDL** код не будет способствовать оптимальной реализации.

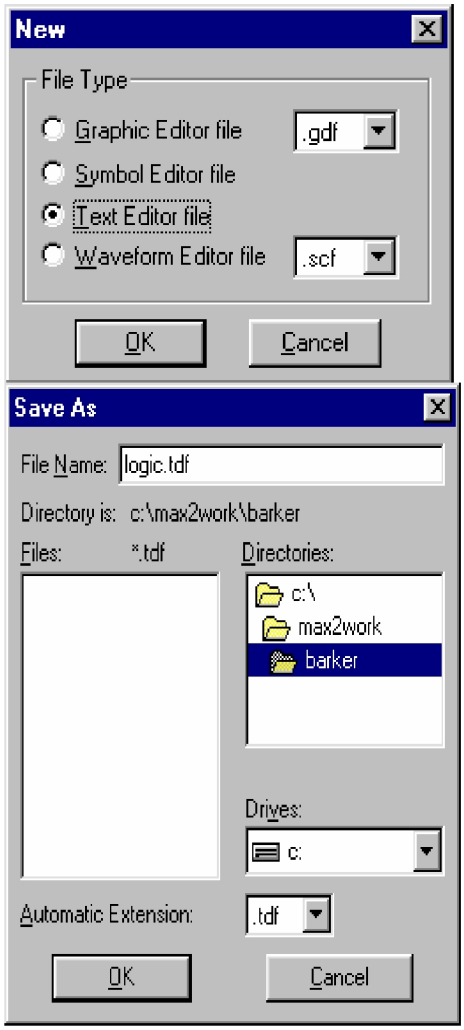

Рис. 22. Формирование текстового файла

Вначале можно закрыть окна всех редакторов, сохранив лишь окно управляющей оболочки **MAX+PLUS II manager.** Сформируем текстовый файл по команде **File/New…**, в окне диалога **File Type** (рис. 22) указываем на файл текстового редактора **Text Editor file**. После открытия окна текстового редактора сохраняем будущий файл схемы под именем **logic.tdf** в папке **C:\max2work\ barker** (рис. 22) по команде **File/Save As…**. (Текстовый редактор также можно запустить из управляющей оболочки по команде **MAX+PLUS II/Text Editor**).

В окне текстового редактора следует набрать листинг программы на языке AHDL по образцу, приведенному ниже:

```
SUBDESIGN LOGIC
    (
        x1,x2,x3,x4        : INPUT;
        z                  : OUTPUT;
    )
    BEGIN
    z = x1 # x2 # x3 # x4;
    END;
```

Дадим пояснения к программе. В первой строчке задано имя схемы (субдизайна), **оно должно совпадать с именем файла**. Затем перечисля-

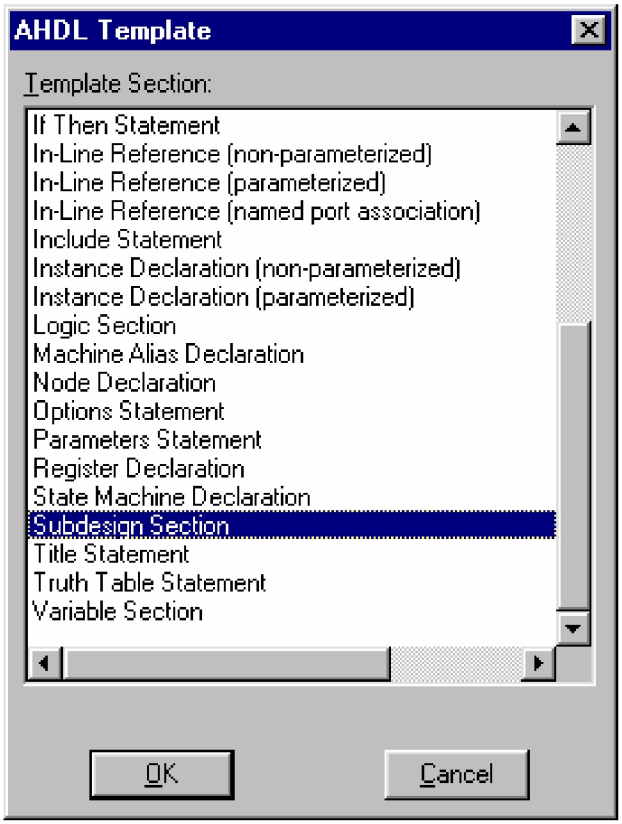

Рис. 23. Вставка AHDL-шаблона

ются входные и выходной сигналы схемы в соответствии с логикой работы. Логическая секция заключена между операторами **BEGIN** и **END**. В ней выполняется логическая функция **ИЛИ** над четырьмя входными сигналами **X1**, **X2**, **X3** и **X4**, а результат передается выходному сигналу **Z**. При наборе сложного кода рекомендуется использовать **AHDL** – шаблоны (рис. 23); они вызываются по команде **Templates/AHDL Templates…**. Вставленный в текст программы образец кода затем редактируется. В данном примере понадобятся следующие шаблоны: **Logic Section**, **Subdesign Section** и **Boolean Equation**.

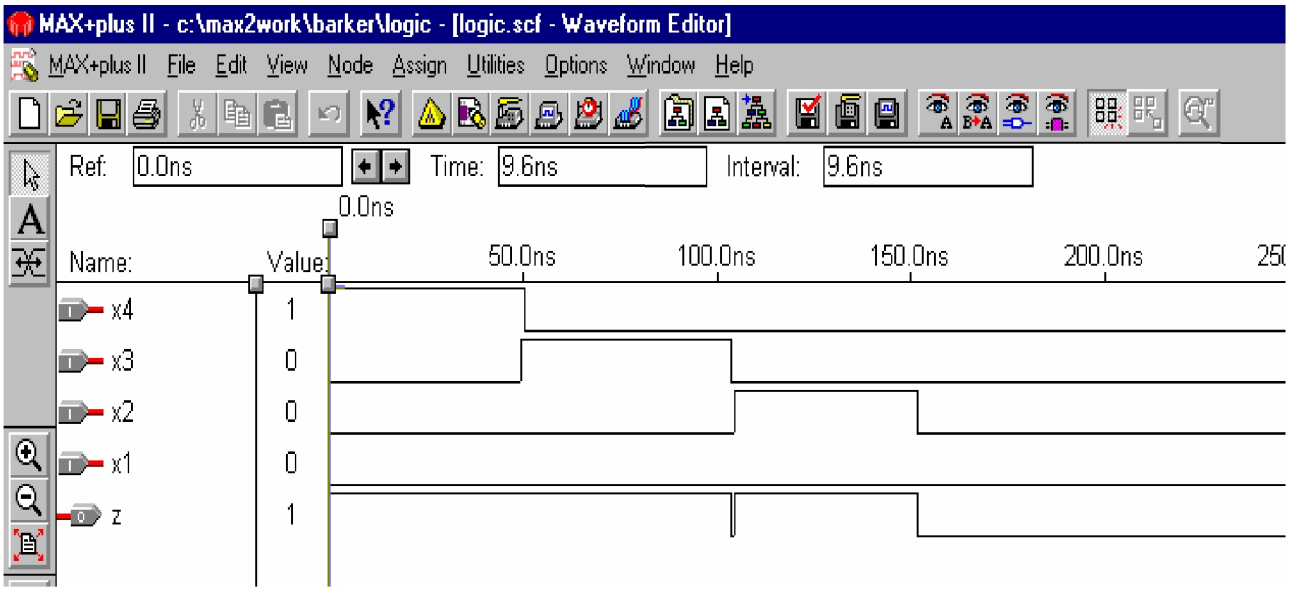

Рис. 24. Результаты моделирования блока Logic

Текстовый файл следует сохранить  , затем сделать главным в проекте (**File/Project/Set Project to Current File**) и откомпилировать  аналогично графическому файлу. Процесс должен пройти без ошибок! Схема довольно простая и не должна вызывать ошибок. Однако если желаете убедиться в правильности ее работы, то проведите верификацию

аналогично пункту **3.4**. Результат верификации показан на рис. 24. Входные сигналы созданы произвольно, чтобы можно было оценить различные состояния схемы.

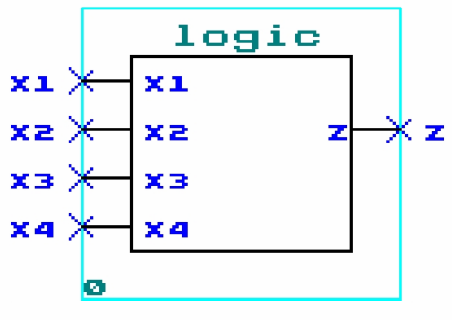

Рис. 25. Символ блока Logic

По команде **File/Create Default Symbol** создаем символ схемы для дальнейшего использования в проекте (рис. 25).

Закрываем все окна, кроме менеджера **MAX+PLUS II manager**, и сохраняем результаты.

3.6. Формирование полной схемы

Полная схема формирователя сигнала кода Баркера состоит из двух подсхем – регистра сдвига (файлы с именем **registr**) и логической схемы **4-ИЛИ** (файлы с именем **logic**). Объединим их в один иерархический проект. Для этого нужно создать новый графический файл с именем **barker.gdf** .

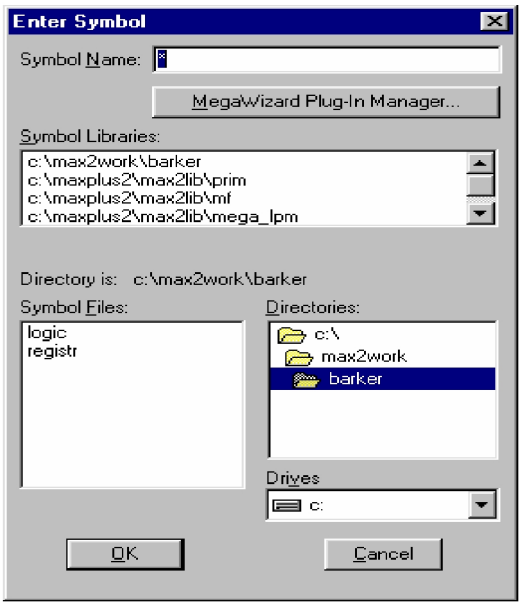

Рис. 26. Ввод символов проекта

В окне графического редактора нужно по команде **Symbol/Enter Symbol…** ввести символы **registr** и **logic** (рис. 26) и соединить их согласно схеме рис.10. Затем подсоединить входные и выходные контакты и дать им имена, как на рис. 27. Выходной контакт имеет имя **Barker**. Одинаковые имена для субдизайна и проекта верхнего уровня допустимы и это поможет упростить разработку тестовых сигналов. Установим файл схемы **barker.gdf** текущим в проекте по команде **File/Project/Set Project to Current File**.

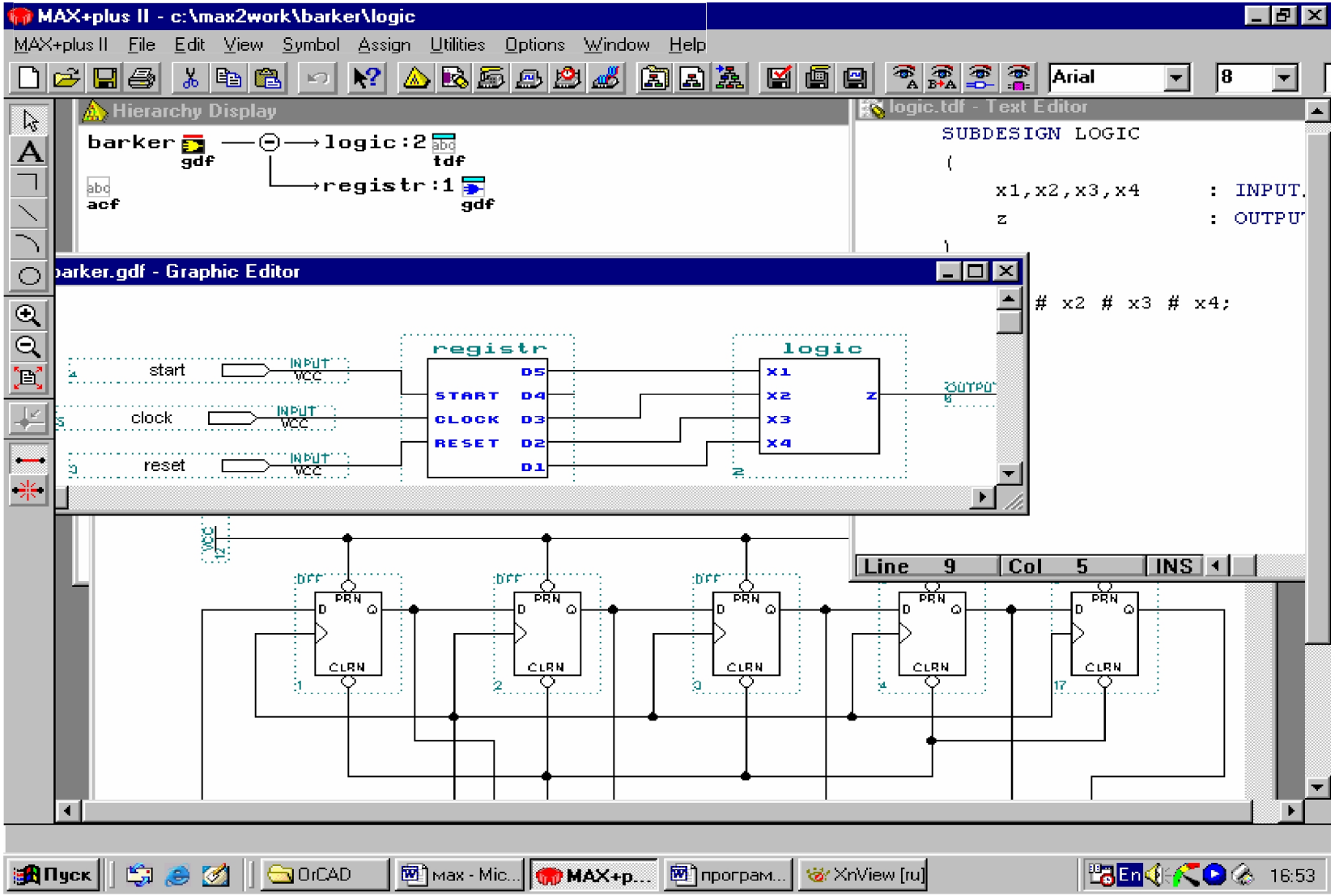

Рис. 27. Иерархия объектов проекта Barker

Рассмотрим иерархию блоков (субдизайнов) в проекте – рис.27. (Команда **MAX+PLUS II/ Hierarchy Display** или ). Из рисунка 27 видно, что итоговая схема **barker.gdf** проекта содержит два субдизайна: графический файл регистра сдвига **registr.gdf** и текстовый файл логической схемы **logic.tdf**. Между компонентами показаны иерархические связи, таким образом легко установить и понять структуру сложного проекта стороннему человеку.

Продолжим разработку проекта – проведем компиляцию и временное моделирование с учетом задержек сигналов внутри микросхемы ПЛИС. После сохранения результатов графической схемы проекта **barker.gdf** запускаем компилятор  . Устанавливаем режим компиляции для временного моделирования **Processing/Timing SNF Extractor** и режим проверки правильности введенных цепей схемы требованиям размещения проекта в выбранной микросхеме **Processing/Desing Doctor**. Процесс компиляции должен пройти без ошибок (рис.28). Компонентов компилятора, как видно из этого рисунка, стало больше, так как происходит реальная разработка с учетом временных задержек. Компилятор сообщает тип микросхемы ПЛМ, в которой будет размещен проект.

Если выбрать значок **rpt** в модуле **FITTER**, то откроется текстовый файл с информацией о размещении проекта в микросхеме ПЛИС, а также выходной AHDL-код, по которому будет запрограммирована микросхема ПЛИС.

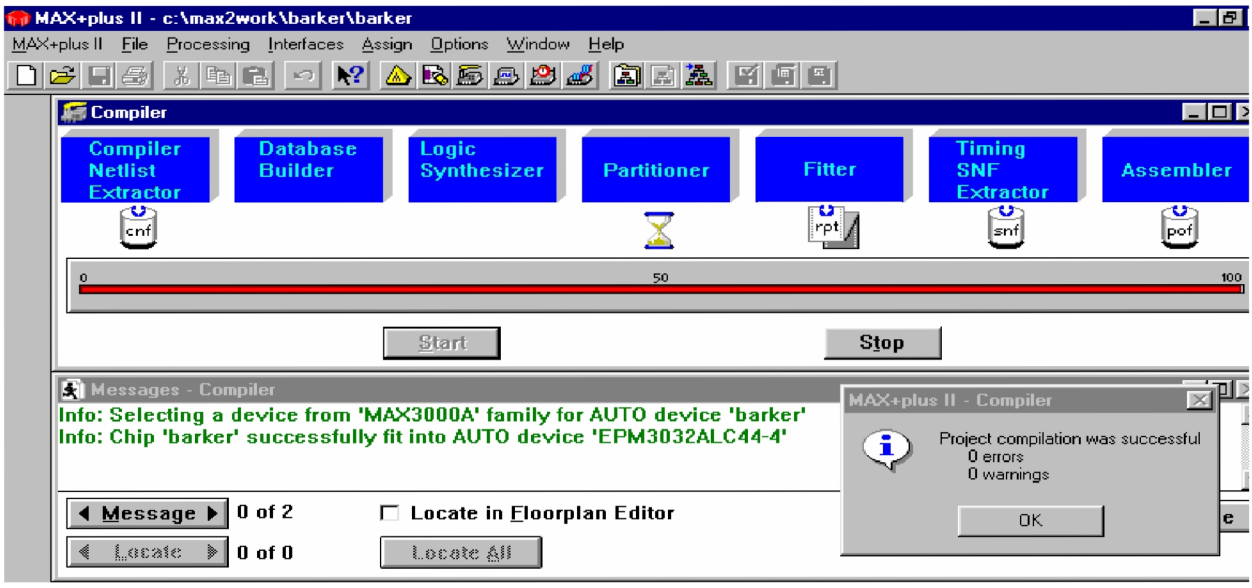

Рис. 28. Окно компилятора проекта barker

3.7. Временное моделирование и оценка результатов

Выбор значка **snf** в модуле **Timing SNF Extractor** приводит к запуску моделирования проекта с учетом временных задержек, что очень удобно в процессе многократной переделки схемы. Но вначале необходимо создать файл тестовых воздействий. Этот файл полностью совпадает с файлом **registr.scf,** созданным в п. 3.4 для анализа работы субдизайна **registr**, но теперь он должен иметь имя **barker.scf**, а также содержать выходной сигнал с именем **barker**.

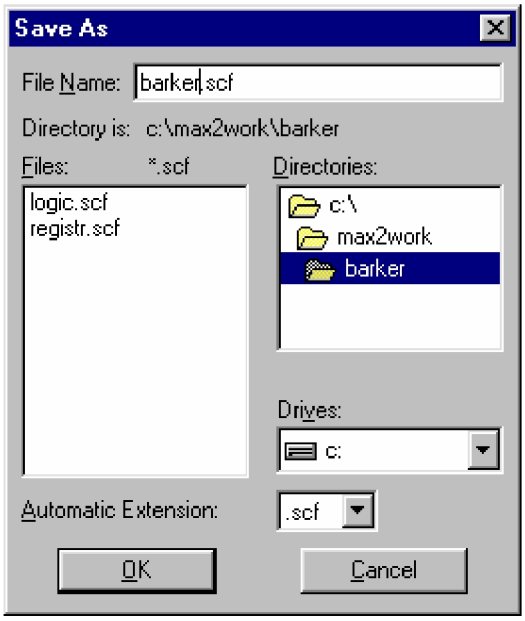

Рис. 29. Диалог создания файла barker.scf

Можно рекомендовать любой из двух способов создания файла тестовых сигналов:

1. Сформировать новый файл **barker.scf** по аналогии с п. 3.4.

2. Открыть существующий файл **registr .scf**, а затем сохранить под новым именем **barker.scf** (рис.29). В новом файле отредактировать выходные сигналы (т.к. входные совпадают) – удалить узлы **D1-D5** и ввести узел **barker** – и сохранить изменения.

Переходим к временному моделированию. Для запуска симулятора временного моделирования выполняем команду **MAX+PLUS II /Simulator**, или  нажимаем кнопку пиктограммы на панели инструментов, или если открыто окно компилятора, то дважды щелкаем мышкой по объекту **Timing SNF Extractor**.

В открывшемся окне **Simulator: Timing Simulation** указан файл входных воздействий **barker.scf** в строке **Simulation Input**, а время начала и конца анализа можно изменять в строках **Start Time** и **End Time** соответственно. Для запуска симулятора нужно нажать на кнопку **Start**, а по окончании убедиться в успешном завершении процесса моделирования.

Для просмотра результата нужно нажать на кнопку **Open SCF**. Запускается редактор временных диаграмм **Waveform Editor**, и в  его окне просматриваем входные и выходные сигналы (рис. 30). На диаграмме видно, что форма сигнала **barker** соответствует пятипозиционному коду Баркера, что говорит о правильной работе схемы проекта. Из рисунка видна задержка между выходным сигналом **barker** и сигналом синхронизации **clock,** которая составляет **6ns**.

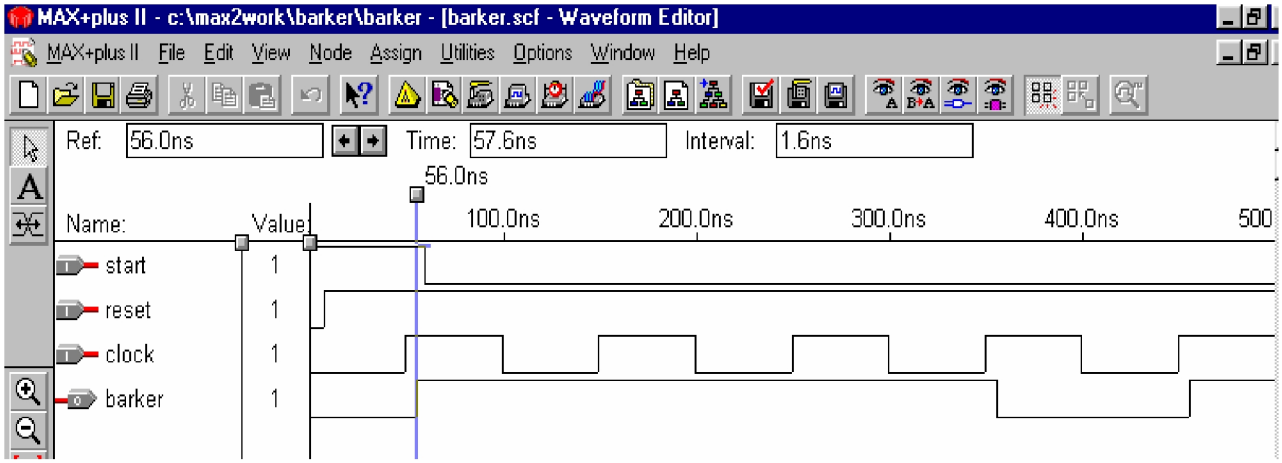

Рис. 30. Временные диаграммы проекта barker с учетом временных задержек

Если временные диаграммы имеют большую длительность, то анализ временных задержек будет затруднителен. Поэтому пакет **MAX+PLUS II** содержит модуль **Timing Analyzer (анализатор временных задержек),** в котором предусмотрен режим расчета задержек в схеме и отображения их в компактной матричной форме. Для запуска анализатора временных задержек выполняем команду **MAX+PLUS II / Timing Analyzer**, или нажимаем кнопку пиктограммы на панели инструментов.

Программа рассчитывает три вида временных задержек:

1. По команде **Analysis/Delay Matrix** рассчитываются временные задержки между группой входных сигналов **SOURCE** и группой связанных с ней выходных сигналов **DESTINATION**. Эта функция полезна для комбинационных логических схем, так как позволяет быстро просмотреть большой объем сигналов и выявить недостатки.

2. По команде **Analysis/Setup/Hold Matrix** рассчитываются времена установки/удержания сигналов на входе регистровых элементов ПЛИС относительно сигнала синхронизации. Эта функция позволяет выявить специфические недостатки в процессе разработки цифровых схем, называемые состязанием фронтов. Рассчитывается минимально необходимое время установки/удержания сигнала на информационном входе D- триггера, при котором будет не нарушено условие его работоспособности. Состязание фронтов возникает из-за задержек в распространении сигналов по путям комбинационных схем или других регистровых узлов ПЛИС. Вот почему логический синтезатор компилятора пакета **MAX+PLUS II** стремится минимизировать входные логические выражения, а после разработки любые задержки между входными и выходными

сигналами для комбинационной логики обычно равны минимальной величине, указываемой в паспортных данных микросхемы ПЛИС.

3. По команде **Analysis/Setup/Registered Performance** рассчитывается производительность регистровых операций и соответственно максимально возможная частота работы выбранного типа микросхемы ПЛИС. При нажатии на кнопку **List Paths** процессор сообщений дает детальную информацию о каждом регистровом узле ПЛИС.

Для начала анализа во всех трех режимах нужно нажать на кнопку **START**, при условии успешного окончания процесса анализа будет выведена матрица задержек.

Результаты анализа временных задержек для проекта **barker** приведены на рис. 31. Результат анализа **Delay Matrix** показывает взаимосвязь только межу сигналами **clock** и **barker**, что соответствует схеме. Максимальная задержка в данном случае составляет 6ns.

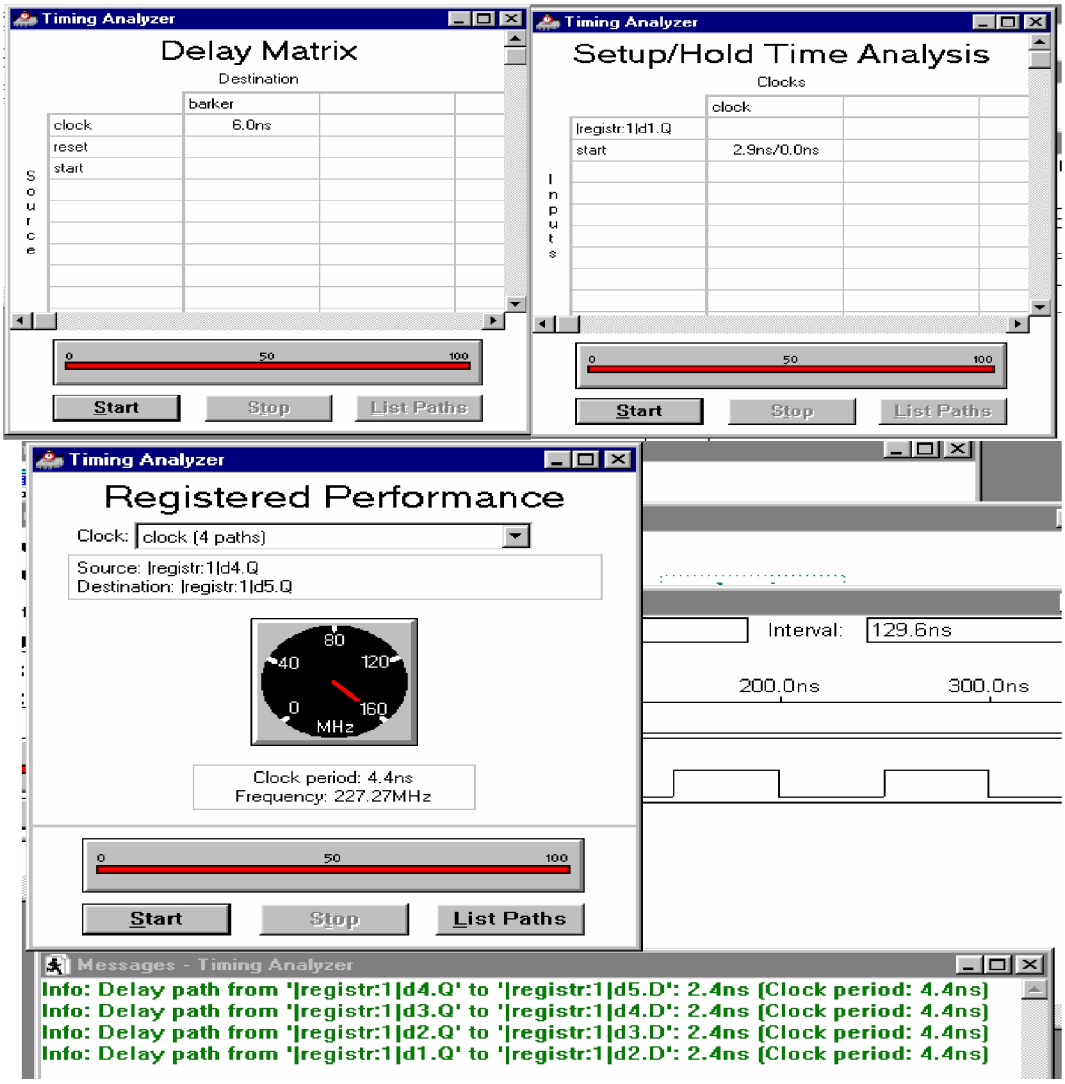

Рис. 31. Анализ временных задержек

Результат анализа **Setup/Hold Matrix** показывает, что минимальное время установки сигнала на входе **START** относительно сигнала синхронизации **CLOCK** составляет 2,9ns.

Результат анализа **Registered Performance** показывает, что максимальная частота синхронизации для данного проекта может достигать 227,27МГц, а минимальная задержка сигнала в каждом регистровом узле составляет 2,4ns.

Если полученные результаты анализа временных задержек не устраивают разработчика, то следует изменить конфигурацию схемы, или изменить режимы работы логического синтезатора в модуле компилятора, или выбрать более быстродействующую (а следовательно и более дорогую) микросхему ПЛИС.

На этом разработка проекта может считаться законченной. Дальнейшие действия обычно связаны со следующими этапами проектирования ПЛИС.

Проект можно запрограммировать в микросхему ПЛИС, используя модуль программатора **Programmer** пакета **MAX+PLUS II** и специальный кабель, подключаемый к порту принтера персонального компьютера.

Затем проводится тестирование (временное моделирование) с учетом реальной микросхемы при помощи модуля **Simulator** пакета **MAX+PLUS II**. Для этого в окне **Simulator: Timing Simulation** необходимо отметить пункт **Use Device**.

Если результаты разработки будут участвовать в других проектах, то по команде **File/Create Default Symbol** создается графический символ проекта в виде компонента схемы (рис.32), а материалы проекта используются в других проектах.

Рис. 32. Графический символ проекта **barker**

4. ИСПОЛЬЗОВАНИЕ ЯЗЫКА VHDL

Как уже отмечалось в предыдущих главах, возрастающая степень интеграции ПЛИС, новые концепции проектирования (система на кристалле) накладывают свой отпечаток на способы описания проекта на ПЛИС. Языки описания аппаратуры (Hardware Description Language) являются формальной записью, которая может быть использована на всех этапах разработки цифровых электронных систем. Это возможно вследствие того, что язык легко воспринимается как машиной, так и человеком. Он может использоваться на этапах проектирования, верификации, синтеза и тестирования аппаратуры так же, как и для передачи данных о проекте, модификации и сопровождении. Существует несколько разновидностей этих языков: **AHDL**, **VHDL**, **VerilogHDL**, **Abel** и др. Известны также случаи использования стандартных языков программирования, например Си, для описания структуры БИС.

Ряд языков описания аппаратуры (**AHDL**, **Abel**) предназначен для описания систем на ПЛИС, другие появились изначально как средство моделирования цифровых систем, а затем стали инструментом их описания.

Одним из наиболее универсальных языков описания аппаратуры является **VHDL**, первый стандарт которого был разработан в 1983–1987 годах при спонсорстве минобороны США. На этом языке возможно как поведенческое, так структурное и потоковое описание цифровых схем. **VHDL** поддерживает три различных стиля для описания аппаратных архитектур.

Первый из них — структурное описание (***structural description***), в котором архитектура представляется в виде иерархии связанных компонентов.

Второй — потоковое описание (***data-flow description***), в котором архитектура представляется в виде множества параллельных регистровых операций, каждая из которых управляется вентильными сигналами. Потоковое описание соответствует стилю описания, используемому в языках регистровых передач.

И, наконец, поведенческое описание (***behavioral description***), в котором преобразование описывается последовательными программными предложениями, которые похожи на имеющиеся в любом современном

языке программирования высокого уровня. Все три стиля могут совместно использоваться в одной архитектуре.

Структурное и потоковое описание используется в основном для проектирования цифровых схем, поведенческое — только для моделирования, так как содержит конструкции, которые невозможно реализовать в виде схемы. Наиболее важными в языке VHDL являются понятия параллелизма и иерархии.

ОБЪЕКТ ПРОЕКТА (***entity***) представляет собой описание компоненты проекта, имеющей чётко заданные входы и выходы и выполняющей чётко определённую функцию. Объект проекта может представлять всю проектируемую систему, некоторую подсистему, устройство, узел, стойку, плату, кристалл, макроячейку, логический элемент и т. п. В описании объекта проекта можно использовать компоненты, которые, в свою очередь, могут быть описаны как самостоятельные объекты проекта более низкого уровня. Таким образом, каждый компонент объекта проекта может быть связан с объектом проекта более низкого уровня. В результате такой декомпозиции пользователь строит иерархию объектов проекта, представляющую весь проект в целом и состоящую из нескольких уровней абстракций. Такая совокупность объектов проекта называется **ИЕРАРХИЕЙ ПРОЕКТА** (***design hierarchy***).

Каждый объект проекта состоит, как минимум, из двух различных типов описаний: описания интерфейса и одного или более архитектурных тел. Интерфейс описывается в **ОБЪЯВЛЕНИИ ОБЪЕКТА ПРОЕКТА** (***entity declaration***) и определяет только входы и выходы объекта проекта.

Для описания поведения объекта или его структуры служит **АРХИТЕКТУРНОЕ ТЕЛО** (***architecture body***). Чтобы задать, какие объекты проекта использованы для создания полного проекта, используется **ОБЪЯВЛЕНИЕ КОНФИГУРАЦИИ** (***configuration declaration***).

В языке VHDL предусмотрен механизм пакетов для часто используемых описаний, констант, типов, сигналов. Эти описания помещаются в **ОБЪЯВЛЕНИИ ПАКЕТА** (***package declaration***). Если пользователь использует нестандартные операции или функции, их интерфейсы описываются в объявлении пакета, а тела содержатся в **ТЕЛЕ ПАКЕТА** (**package body**).

Таким образом, при описании цифровых схем на языке **VHDL** возможно использование пяти различных типов описаний: объявление объ-

екта проекта, архитектурное тело, объявление конфигурации, объявление пакета и тело пакета. Каждое из описаний является самостоятельной конструкцией языка **VHDL**, может быть независимо проанализировано анализатором и поэтому получило название **"МОДУЛЬ ПРОЕКТА"** (***design unit***). Модули проекта, в свою очередь, можно разбить на две категории: ПЕРВИЧНЫЕ и ВТОРИЧНЫЕ. К первичным модулям относятся различного типа объявления. К вторичным — отдельно анализируемые тела первичных модулей. Один или несколько модулей проекта могут быть помещены в один файл, называемый **ФАЙЛОМ ПРОЕКТА** (***design file***). Каждый проанализированный модуль проекта помещается в **БИБЛИОТЕКУ ПРОЕКТА** (***design library***) и становится **БИБЛИОТЕЧНЫМ МОДУЛЕМ** (***library unit***). Данная реализация позволяет создать любое число библиотек проекта. Каждая библиотека проекта в языке **VHDL** имеет логическое имя (идентификатор). Фактическое имя файла, содержащего эту библиотеку, может совпадать или не совпадать с логическим именем библиотеки проекта. Для ассоциирования логического имени библиотеки с соответствующим ей фактическим именем предусмотрен специальный механизм установки внешних ссылок.

Объекты данных (***data object***) являются хранилищами для значений определённого типа. Следует заметить, что все типы в **VHDL** конструируются из элементов, представляющих собой скалярные типы. Значения всех объектов в создаваемой модели, взятые все вместе, отражают текущее состояние моделирования. Описание на **VHDL** содержит объявления, которые создают объекты данных четырёх классов: константы, переменные, сигналы и файлы.

Константы и переменные содержат одно значение данного типа. Значения переменных могут быть изменены назначением нового значения в предложении назначения переменной. Значение константы устанавливается до начала моделирования и не может после этого изменяться.

Сигнал имеет текущее значение подобно переменной. Он также имеет прошлую историю значений, на которые разработчик может пожелать сослаться, а также множество будущих значений, которые будут получены от формирователей сигналов. Новые значения для сигналов создаются предложениями назначения сигналов. Каждый объект в описании должен ассоциироваться с одним и только одним типом. Тип состоит из множества возможных значений и множества операций.

Имеются операции двух видов. Некоторые операции являются предопределёнными, это, к примеру, операторы “+”, “-“ для значений типа ***integer***. Другие операции явно кодируются в VHDL; например, может быть написана функция подпрограмма **Max**, которая возвращает наибольший из двух целых аргументов. Тип объекта представляет информацию, которая окончательно определяется в момент записи модели. Эта информация способствует обнаружению несоответствий в тексте без обращения к моделированию. Например, легко обнаружить и отметить попытку назначения булевого значения (**True** или **False**) целой переменной. Новое значение, которое должно быть создано предложением назначения, определяется выражением в правой части. Выражения используются также и в других контекстах: например, как условие в предложении ***if***. В состав выражения могут входить константы, переменные, сигналы, операторы и указатели функций. Когда имя объекта используется в выражении, при расчёте значения выражения учитывается его текущее значение. Рассмотрим некоторые примеры описания цифровых схем на **VHDL**.

4.1. Основные понятия языка VHDL

В пакетах САПР используется понятие пары **entity/architecture** (объект проекта/архитектура) языка **VHDL** для описания как моделей элементов, так и схем устройств. Это значительно упрощает понимание языка **VHDL**, особенно в стадии изучения. Однако, чтобы полностью использовать все возможности **VHDL**, следует придерживаться общих правил “хорошего тона” в программировании: **VHDL** - файл может содержать одну или несколько моделей. **VHDL** –модель, в противоположность **VHDL** -файлу, определяется как пара **entity/architecture**. **VHDL** –файл в общем случае представляет более сложное образование, содержащее, как правило, вышеуказанные структуры данных. Такое усложнение понятий – вполне закономерный процесс, связанный с необходимостью перехода от описания ПЛИС на схемном (компонентном) уровне к описанию на уровне алгоритмов, тем более что для реализации сложных методов ЦОС можно и не найти схемотехнических элементов-прототипов.

Секция **Объект проекта** задается ключевым словом **entity**, в ней оператором **port** указываются имена сигналов интерфейса схемы, их тип (**in** - вход, **out** - выход), тип данных, используемых портами (например вида **STD_LOGIC**).

Секция **Архитектура** задается ключевым словом **architecture**, в ней, собственно, и описывается содержимое схемы (оператор **structure**) или ее поведение (оператор **behavior**). Как было отмечено выше, тело **Архитектуры** может содержать весьма разнообразные объекты, которые нужно объявлять соответствующими ключевыми словами (**объявление конфигурации**). Например, если посмотреть **VHDL** –файл синтезированной ПЛИС кода Баркера, то он содержит объявление конфигурации как **structure**, так как в файле приводится только способ соединения отдельных моделей. Поведение каждой отдельной модели содержится в соответствующих файлах **VHDL** –моделей, с объявлением конфигурации оператором **behavior**. Ниже представлен пример описания схемы, выполняющей логическую операцию **И** над двумя входными переменными **in1** и **in2**, а результат передается выходной переменной **o** (ключевые слова языка **VHDL** выделены жирным шрифтом; в **VHDL** редакторе они выделяются цветом, что улучшает восприятие программы).

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity and2 is
  port (in1, in2 : std_logic;
        o : out std_logic);
end and2;
architecture behavior of and2 is
begin
  o <= in1 and in2;
end behavior;
```

Первые три строки указывают на дополнительные ресурсы (объявление пакетов (***package declaration***)), находящиеся в библиотеках (**package body) ieee.std_logic_1164** и **ieee.numeric_std**, которые значительно расширяют логические, арифметические и другие функции (например логического синтеза) исходного кода языка **VHDL**. Всегда желательно включать эти расширения в текст модели или схемы. Очевидно, что при необходимости и другие расширения языка **VHDL,** оформленные в виде подключаемых модулей библиотек, можно включать в свой про-

ект. Примеры использования различных расширений, их состав, место расположения на диске и способ подсоединения к проекту подробно рассмотрены в справочной системе пакета САПР.

Следующие четыре строки программы и есть интерфейс: описание соединений с внешними схемами. В данном случае модель имеет три порта: из них два - **in1** и **in2** - входные (если тип не указан, то по умолчанию порт считается входным); и один - **o** – выходной, его тип **out**. **VHDL** - порты эквивалентны выводам схемного компонента или иерархическим портам разрабатываемой схемы. Они описывают коммуникационные каналы модели.

В приведенной выше программе порты были определены типом **STD_LOGIC**, который обязательно должен быть объявлен для использования сигналов интерфейса. Тип данных выбирается разработчиком в зависимости от функций порта интерфейса из допустимых типов данных. Дополнительные модули библиотек (или расширения языка **VHDL**) могут включать собственные типы данных, существенно расширяющие возможности стандартного набора типов данных. Рекомендуется ознакомиться с типами данных расширений языка, которые обычно приведены в **VHDL**-файле модуля расширения.

Секция архитектуры объявляется ключевым словом **architecture**. Она связана с моделью ***and2***, в которой приводится описание поведения (ключевое слово ***behavior***) с помощью логических функций. В данном случае выполняется операция логического «И» над двумя входными сигналами. Приведенный выше пример показывает, что изучение языка **VHDL** не столь затруднительно и наиболее приемлемый способ состоит в рассмотрении способов реализации базовых логических схем комбинаторной и последовательной логики, основных операторов, процедур и функций на реальных примерах.

4.2. Особенности реализации VHDL в пакетах САПР

Язык **VHDL** используется на всех этапах разработки цифровых схем: на этапе описания схемы, в виде моделей цифровых компонентов, на этапе синтеза структуры схемы и определения ее временных параметров (величин задержек распространения сигналов), на этапе создания списка соединений (***netlist***), на этапе синтеза и оптимизации, для управле-

ния процессом размещения и трассировки в кристалле ПЛИС, на этапе создания тестовых векторов для проверки правильности работы, на этапе создания символа нового компонента для дальнейшего использования разработки и т.д. Это приводит к тому, что в проекте создается большое число **VHDL**-файлов. Графически **VHDL**-файлы появляются в виде пиктограмм, вид которых зависит от определенного типа, который, в свою очередь зависит от того, какое место занимает этот **VHDL**-файл в ходе процесса работы пакета САПР над проектом.

Все рассмотренные типы файлов можно открыть для просмотра и изменения в **VHDL**-редакторе. Его использование предпочтительнее по сравнению с обычным текстовым редактором. Рассмотрим основные свойства **VHDL**-редактора. На рис. 33 представлен фрагмент **VHDL**-кода для схемы формирователя, разработанной в пункте 3. Пояснения на рис. 33 показывают основные компоненты **VHDL**-редактора. Нажатие на правую кнопку мыши вызывает контекстно-зависимое меню.

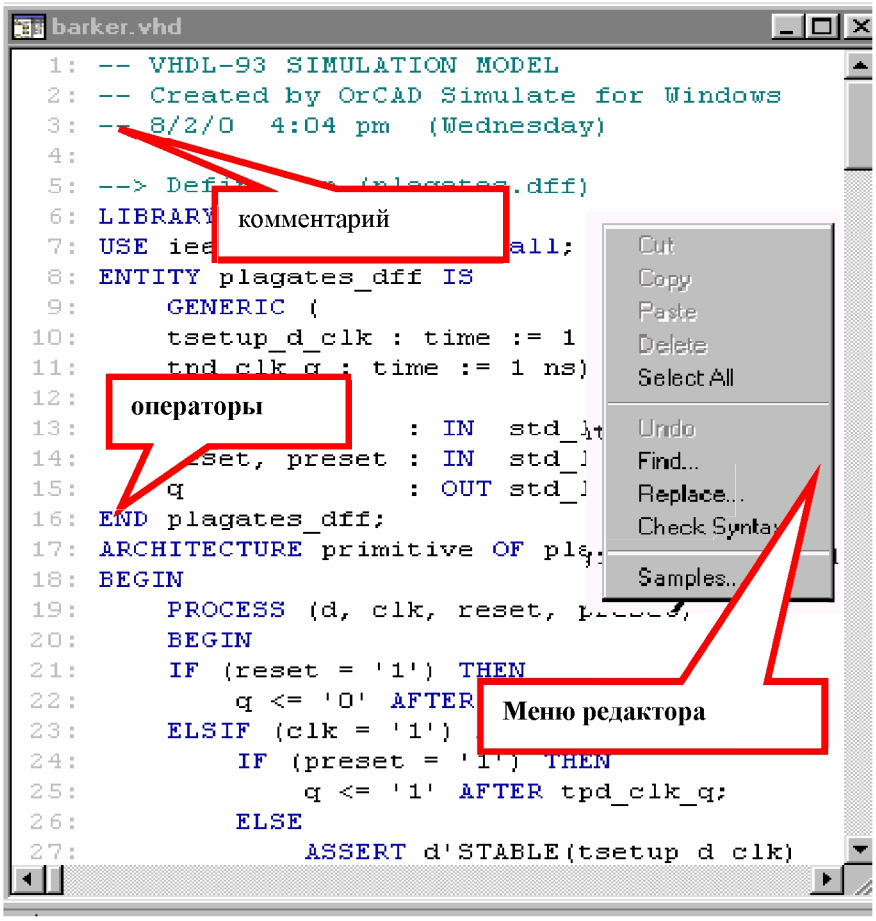

Рис. 33. Окно **VHDL**-редактора

Отметим наиболее важные команды этого меню. Для проверки синтаксиса языка **VHDL** используем команду **Syntax Check**. Проверка начинается от текущего месторасположения курсора. При возникновении ошибки курсор будет указывать на строку с ошибкой. Следует помнить, что ошибки в логике программы эта команда не проверяет. Для поиска логических ошибок следует использовать средства отладки пакета MAX PLUS.

Чтобы свести к минимуму синтаксические ошибки целесообразно использовать команду из меню **VHDL**-редактора. В открывающемся окне **VHDL Samples** представлены все основные образцы команд языка **VHDL**. В нижней части даны образцы заготовок, которые нужно скопировать в свой проект и затем отредактировать. Образцы собраны в файле STANDARD.VHX, в который можно добавить собственные компоненты. На рис.34 представлен образец **entity** (Объект проекта).

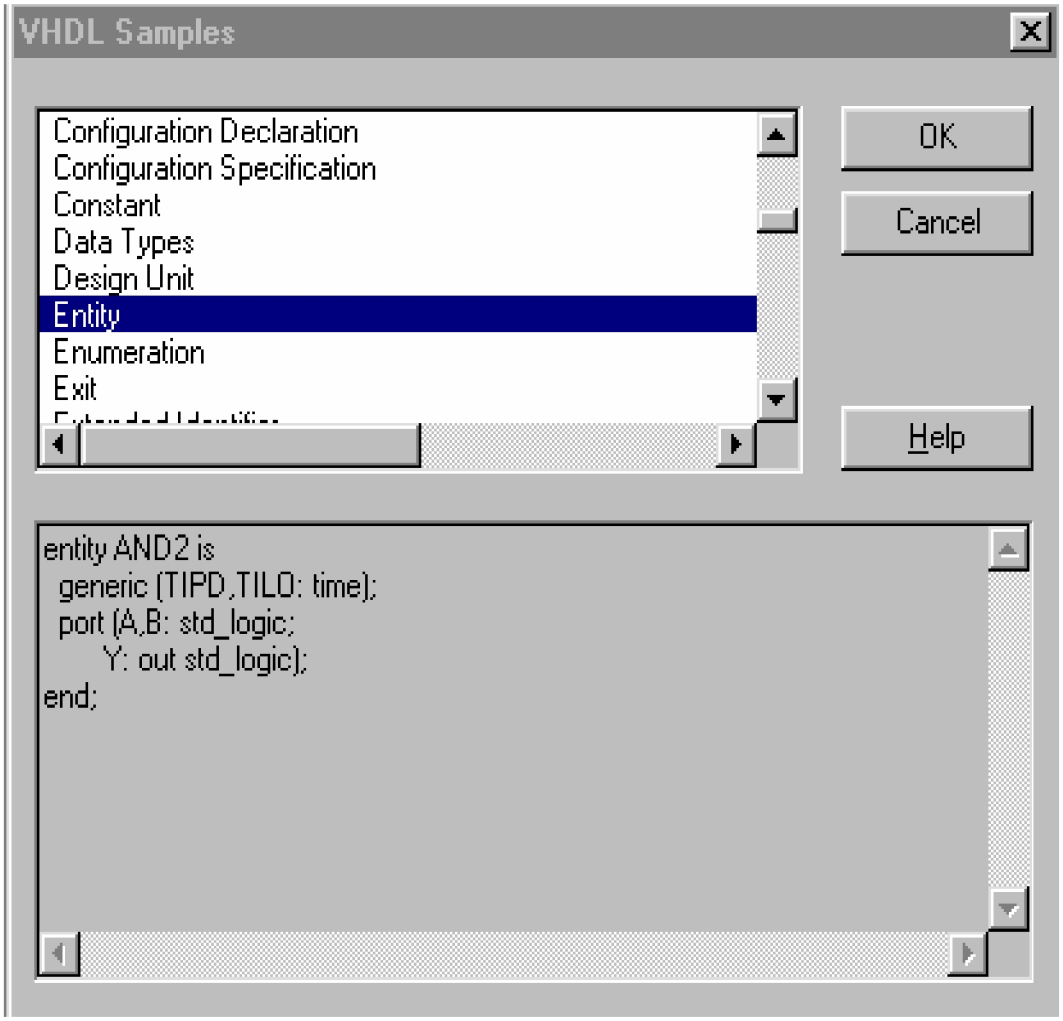

Рис. 34. Вызов образцов конструкций **VHDL**

С помощью пакета САПР можно создать **VHDL**-код для иерархического блока, размещаемого на странице схемы в **Capture**. Размещенный в блоке **VHDL**-код будет определять логику работы. Созданная таким образом **VHDL**-модель характерна для разработки проектов методом нисходящего проектирования. Однажды отработанную модель можно использовать в различных проектах.

Подведем некоторые итоги. Язык **VHDL** является формальным описанием, предназначенный для использования на всех стадиях разработки проектов электронных систем. В силу того что он хорошо воспринимается и человеком и компьютером, он поддерживает разработку, верификацию, синтез и тестирование аппаратных средств, передачу данных на аппаратном уровне, а также обслуживание, модификацию и документирование компонентов на базе ПЛИС.

Благодаря своей гибкости, **VHDL** может быть использован различными способами. Так как язык описывает поведение логических элементов, то он может быть использован для моделирования моделей и составляющих сложного проекта. Например, исходный код для модели простого вентиля «И» с реальной задержкой выходного сигнала на 2нс может быть следующим:

o <= in1 and in2 after 2 ns.

На рис.35 представлены временные диаграммы моделирования работы, подтверждающие правильность логики функционирования модели. Тот же самый исходный код может быть использован для синтеза или компиляции **VHDL**-логики, чтобы учесть в преобразованиях специфические свойства аппаратного элемента.

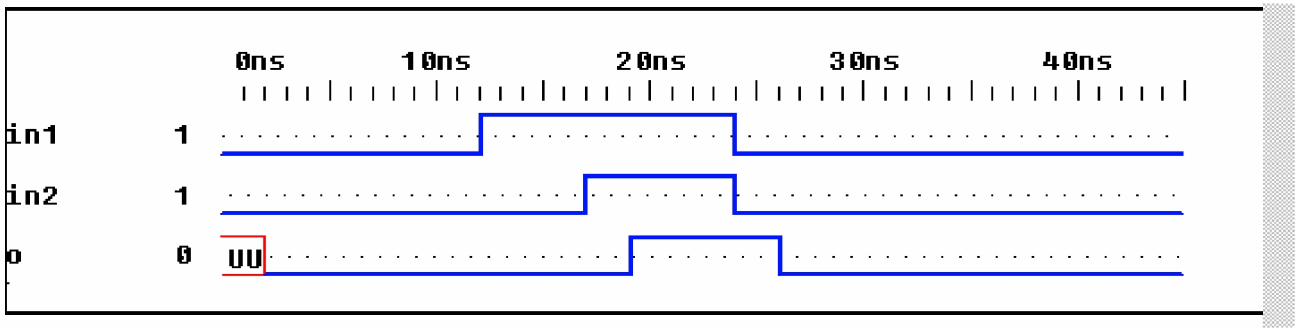

Рис. 35. Результаты моделирования работы вентиля «И»

Синтезатор **MAX PLUS** генерирует список узлов, как показано во фрагменте ниже, в котором определяется аппаратный компонент.

SYM, U1, AND, LIBVER=2.0.0
PIN, I0, I, IN1
PIN, I1, I, IN2
PIN, O, O, _O_
END

Компилятор **MAX PLUS** воспринимает **VHDL** –коды и схемы, а затем создает список узлов на уровне вентилей из входящих компонентов, подобно примеру, представленному выше.

4.3. Типы данных

Язык программирования **VHDL** относится к языкам с жестко установленными ("strongly typed") правилами объявления и использования типов данных. Это означает, что данные должны быть объявлены до того, как они будут использованы, а действия с неправильными типами будут ограничены. В списке, приведенном ниже, указаны основные типы данных, входящие в стандартный набор пакета ***std_logic*** конструкций **VHDL**, а также в расширения пакетов ***std_logic_1164*** и ***Numeric_std***.

Стандартный тип (Standard types)	Классификатор
boolean, bit character	Перечисление (Enumeration)
integer	Целое число
natural, positive	Подмножество целого
string	Массив знаков (Array of character)
bit_vector	Массив бит (Array of bit)

Тип Std_logic_1164	
std_ulogic	Перечисление (Enumeration)
std_logic	Подмножество типа std_ulogic
std_ulogic_vector	Массив типа std_ulogic
std_logic_vector	Массив типа std_logic

Тип Numeric_std

signed	Произвольный массив бит
unsigned	Произвольный массив бит

Вышеуказанные типы данных являются стандартными для описания поведения электронных схем и в соответствии со стандартом **VHDL** воспринимаются большинством средств САПР. Данные, определенные как ***std_logic,*** имеют 9 стандартных уровней, которые обозначаются следующим образом:

Состояние	*Значение*
U (Uninitialized)	**НЕ ОПРЕДЕЛЕН**
X (Forcing unknown)	**НЕ ИЗВЕСТЕН**
0 (Forcing logic 0)	**ЛОГИЧЕСКИЙ «0»**
1 (Forcing logic 1)	**ЛОГИЧЕСКАЯ «1»**
Z (High impedance)	**СОСТОЯНИЕ ВЫСОКОГО СОПРОТИВЛЕНИЯ**
W (Weak unknown)	**СЛАБЫЙ НЕОПРЕДЕЛЕННЫЙ СИГНАЛ**
L (Weak logic 0)	**СЛАБЫЙ СИГНАЛ ЛОГИЧЕСКОГО «0»**
(Характерен для состояния «открытый эмиттер» ЭСЛ логики)	
H (Weak logic 1)	**СЛАБЫЙ СИГНАЛ ЛОГИЧЕСКОЙ «1»**
(Характерен для состояния «открытый коллектор» ТТЛ логики)	
- (Don't care)	**НЕ ИМЕЕТ ЗНАЧЕНИЯ**

Следует отметить, что различные библиотеки расширений (пакеты) могут содержать собственные типы данных, которые нужно объявлять. Состав типов данных находится в файле расширения (пакета) и может быть просмотрен любым текстовым редактором.

Язык программирования содержит хорошо знакомые компоненты присущие, любому другому языку программирования: операторы, пере-

менные, функции, процедуры, процессы, подпрограммы. Их изучение лучше всего рассматривать на конкретных примерах, постепенно продвигаясь от простого к сложному.

4.4. Комбинационная логика

Как известно, логическая схема называется комбинационной, если состояния выходов являются только функциями входов независимо от момента времени. Комбинационная логика в языке **VHDL** реализована булевыми выражениями и уравнениями, таблицами истинности и большим количеством «макрофункций» через расширения стандартных библиотек. В число примеров комбинаторных логических функций входят дешифраторы, мультиплексоры и сумматоры.

Булевы выражения — это множества узлов, чисел, констант и других булевых выражений, выделяемых операторами, компараторами (операциями сравнения) и, возможно, сгруппированных в заключающих круглых скобках. Булево уравнение устанавливает равенство между узлом или группой и булевым выражением. Приведем список операторов языка **VHDL**, сгруппированных в четыре основных класса – логические, операции сравнения, арифметические, операции конкатенации (слияния символьных строк):

Категория	Оператор	Функция
Логические	**And**	логическое И
Boolean	**Or**	логическое ИЛИ
	Nand	логическое И-НЕ
	Nor	логическое ИЛИ-НЕ
	Xor	исключающее ИЛИ
Операции сравнения	=	равенство
Comparison	/=	неравенство
	<	меньше чем
	<=	меньше или равно

	>	больше
	>=	больше или равно
Арифметические	**abs**	абсолютное значение
Arithmetic	+	сложение
	–	вычитание
	*	умножение
	/	деление
	mod	целое (ариф. модуль)
	rem	остаток от деления
Конкатенация	**&**	операция конкатенации
Concatenation		

В представленном ниже примере даны строки булевых выражений, их коды и выполняемые функции.

Булево выражение	Эквивалент на **VHDL**	Действие
Y1 = (A1)(B1)	***y1 <= a1 and b1;***	***AND***
Y2 = A2 + B2	***y2 <= a2 or b2;***	***OR***
Y3 = A3 B3	***y3 <= a3 xor b3;***	***EXCLUSIVE OR***
Y4 = (A4)'	***y4 <= not(a4);***	***NOT***
Y5 = ((A5)(B5))'	***y5 <= a5 nand b5;***	***NAND***
Y6 = ((A6)+(B6))'	***y6 <= a6 nor b6;***	***NOR***

В схемах сигналы представлены цепями и шинами, а элементарные вентили – графическими символами. В **VHDL** сигналы объявляются, а логика работы вентилей вводится операторами языка **VHDL**, такими как **AND** и **OR** . Если предположить, что представленные выше выражения определяют логику работы некоторой цифровой системы, то можно создать следующий исходный код на языке **VHDL**. На рис. 36 он представлен в окне **VHDL-** редактора. Постарайтесь самостоятельно разобраться в структуре этой программы и понять роль ключевых слов языка **VHDL**.

design1.vhd

```
 86:
 87:  -- primer of Combinational logic
 88: library ieee;
 89: use ieee.std_logic_1164.all;
 90: use ieee.numeric_std.all;
 91: entity eqs is
 92:   port (a1, a2, a3, a4, a5, a6,
 93:         b1, b2, b3, b5, b6 : std_logic;
 94:         y1, y2, y3, y4, y5, y6 : out std_logic);
 95: end eqs;
 96: architecture behavior of eqs is
 97: begin
 98:   y1 <= a1 and b1;
 99:   y2 <= a2 or b2;
100:   y3 <= a3 xor b3;
101:   y4 <= not(a4);
102:   y5 <= a5 nand b5;
103:   y6 <= a6 nor b6;
104: end behavior;
105:
```

Рис. 36. Пример исходного **VHDL**-кода

4.5. Мультиплексоры и селекторы

Мультиплексоры и селекторы используются для обработки данных, связанных с выбором значений по некоторому кодовому признаку. Например, при поступлении некоторого кодового слова на управляющий вход мультиплексора соответствующий ему входной информационный канал будет перенаправлен на выход. Таким образом, при помощи кода можно изменять информационные потоки. При помощи оператора **case** языка **VHDL** можно построить как мультиплексоры, так и селекторы. В следующем примере создана схема селектора:

```
 case test_vector is
    when “000" => o <= bus(0) ;
    when “001" | ”010" | “100" => o <= bus(1) ;
    when “011" | ”101" | “110" => o <= bus(2) ;
    when “111" => o <= bus(3) ;
end case ;
```

Если значение селектора – индекс, который будет выбираться из массива, то селектор будет похож на мультиплексор. Данный способ предусматривает использование оператора **case**, однако можно использо-

вать переменную индексированного массива. Например, если целое значение определяет индекс массива, то переменная индексированного массива будет выполнять функцию мультиплексирования:

```
signal vec : std_logic_vector (0 to 15) ;
signal o : std_logic ;
signal i : integer range 0 to 15 ;
 ...
o <= vec(i) ;
```

Здесь сигнал **o** выбирается битом **i** из вектора **vec**. Это эквивалентно более сложному стилю записи с использованием оператора **case**:

```
case i is
    when 0 => o <= vec(0) ;
    when 1 => o <= vec(1) ;
    when 2 => o <= vec(2) ;
    when 3 => o <= vec(3) ;
...
end case ;
```

В результате логического синтеза будет создан мультиплексор, аналогичный по функциям предыдущему, в котором использован массив с переменным индексом. Компилятор модуля полностью поддерживает все типы массивов с переменным индексом, включая индексы, значения которых являются перечисляемыми типами, аналогично целым числам, и индексы, значения которых являются выражениями, аналогично простым идентификаторам.

5. ЯЗЫК ОПИСАНИЯ АППАРАТУРЫ AHDL

Язык описания аппаратуры **AHDL** разработан фирмой **Altera** и предназначен для описания комбинационных и последовательностных логических устройств, групповых операций, цифровых автоматов (***state machine***) и таблиц истинности с учётом архитектурных особенностей ПЛИС фирмы **Altera**. Он полностью интегрируется с системой автоматизированного проектирования ПЛИС **MAX+PlusII**. Файлы описания аппаратуры, написанные на языке **AHDL**, имеют расширение "***.TDF**" (***Text design file***). Для создания TDF-файла можно использовать как текстовый редактор системы **MAX+PlusII**, так и любой другой. Проект, выполненный в виде TDF-файла, компилируется, отлаживается и используется для формирования файла программирования или загрузки ПЛИС.

Операторы и элементы языка **AHDL** являются достаточно мощным и универсальным средством описания алгоритмов функционирования цифровых устройств, удобным в использовании. Язык описания аппаратуры **AHDL** даёт возможность создавать иерархические проекты в рамках одного этого языка или использовать TDF-файлы, разработанные на языке **AHDL**, наряду с другими типами файлов. Для создания проектов на **AHDL** можно, конечно, пользоваться любым текстовым редактором, но текстовый редактор системы **MAX+PlusII** предоставляет ряд дополнительных возможностей для ввода, компиляции и отладки проектов.

Проекты, созданные на языке **AHDL**, легко внедряются в иерархическую структуру. Система **MAX+PlusII** позволяет автоматически создать символ компонента, алгоритм функционирования которого описывается TDF-файлом, и затем вставить его в файл схемного описания (GDF-файл). Подобным же образом можно вводить в любой TDF-файл собственные функции разработчика и около 300 макрофункций, разработанных фирмой **Altera**. Для всех функций, включённых в макробиблиотеку системы **MAX+PlusII**, фирма **Altera** поставляет файлы с расширением "***.inc**", которые используются в операторе включения **INCLUDE**.

При распределении ресурсов устройств разработчик может пользоваться командами текстового редактора или операторами языка **AHDL** для того, чтобы сделать назначения ресурсов и устройств. Кроме того, разработчик может только проверить синтаксис или выполнить полную компиляцию для отладки и запуска проекта.

5.1. Комбинационная логика

Как известно, логическая схема называется комбинационной, если состояния выходов являются только функциями входов независимо от момента времени. Комбинационная логика в языке **AHDL** реализована булевыми выражениями и уравнениями, таблицами истинности и большим количеством макрофункций. В число примеров комбинаторных логических функций входят дешифраторы, мультиплексоры и сумматоры.

Булевы выражения — это множества узлов, чисел, констант и других булевых выражений, выделяемых операторами, компараторами и, возможно, сгруппированных в заключающих круглых скобках. Булево уравнение устанавливает равенство между узлом или группой и булевым выражением. В качестве примера приведён файл ***boole1.tdf***, в котором даны два простых булевых выражения, представляющих два логических элемента.

```
  SUBDESIGN boole1
    (
    a0, a1, b : INPUT;
    out1, out2 : OUTPUT;
    )
BEGIN
    out1 = a1 & !a0;
    out2 = out1 # b;
END;
```

Здесь выход **out1** получается в результате логической операции **И**, применённой ко входу **a1** и инвертированному входу **a0**, а выход **out2** получается в результате применения логической операции **ИЛИ** к выходу **out**1 и входу **b**. Поскольку эти уравнения обрабатываются одновременно, последовательность их следования в файле не важна.

Дополнительно в качестве примера рассмотрим файл ***decoder.tdf*** , реализующий функции дешифратора, преобразующего код из двухразрядного в четырёхразрядный. В результате его работы два двухразрядных двоичных входа преобразуются в один “горячий код”, называемый так потому, что четыре его допустимых значения содержат по одной единице: 0001, 0010, 0100, 1000.

```
SUBDESIGN decoder
        (
            code[1..0] : INPUT;
            out[3..0] : OUTPUT;
        )
    BEGIN
        CASE  code[] IS
WHEN 0 => out[] = B"0001";
WHEN 1 => out[] = B"0010";
WHEN 2 => out[] = B"0100";
WHEN 3 => out[] = B"1000";
          END CASE;
     END;
```

Здесь группа входа **code [1..2]** может принимать значения **0, 1, 2, 3**. В зависимости от реального кода активизируется соответствующая ветвь оператора, и только она одна, в данный момент времени. Например, если вход **code []** равен **1**, на выходе **out** устанавливается значение **B"0010".**

5.2 Последовательностная логика в AHDL

Логическая схема называется последовательностной, если выходы в заданный момент времени являются функцией входов не только в тот же момент, но и во все предыдущие моменты времени. Таким образом, в последовательностную схему должны входить некоторые элементы памяти (триггеры). В языке **AHDL** последовательностная логика реализована цифровыми автоматами с памятью (**state machines**), регистрами и триггерами. При этом средства описания цифровых автоматов занимают особое место. Кроме того, к последовательностным логическим схемам относятся различные счётчики и контроллеры.

Регистры используются для хранения значений данных и промежуточных результатов счётчика, тактирование осуществляется синхросигналом. Регистр создаётся его объявлением в секции **VARIABLE**.

Ниже приводится файл **bur_reg.tdf**, содержащий байтовый регистр, который фиксирует значения на входах **d** по выходам **q** на фронте синхроимпульса, когда уровень загрузки высокий.

```
SUBDESIGN bur_reg
        (
            clk, load, d[7..0]  :  INPUT;
            q[7..0]  : OUTPUT;
         )
    VARIABLE
        ff[7..0]  :  DFFE;
    BEGIN
        ff[].clk = clk;
        ff[].ena = load;
        ff[].d = d[];
        q[] = ff[].q
    END;
```

Как видно из файла, регистр объявлен в секции **VARIABLE** как **D**-триггер с разрешением (**DFFE**). В первом булевом уравнении в логической секции происходит соединение входа тактового сигнала подпроекта к портам тактового сигнала триггеров **ff[7...0]**. Во втором уравнении отпирающий тактовый сигнал соединяется с загрузкой. В третьем уравнении входы данных подпроекта соединяются с портами данных триггеров **ff[7...0]**. В четвёртом уравнении выходы подпроекта соединяются с выходами триггеров. Все четыре уравнения оцениваются одновременно.

Можно также в секции **VARIABLE** объявить T-, JK- и SR-триггеры и затем использовать их в логической секции. При использовании T-триггеров придётся в третьем уравнении изменить порт **d** на **t**. При использовании JK- и SR-триггеров вместо третьего уравнения придётся записать два уравнения, в которых происходит соединение портов **j** и **k** или **s** и **r** с сигналами. При загрузке регистра по заданному фронту глобального тактового сигнала фирма Altera рекомендует использовать вход разрешения одного из регистров: **DFFE**, **TFFE**, **JKFFE** или **SRFFE**, чтобы контролировать загрузку регистра.

Счётчиками называются последовательностные логические схемы для счёта тактовых импульсов. В некоторых счётчиках реализован счёт вперед и назад (реверсивные счётчики), в некоторые счётчики можно загружать данные, а также обнулять их. Счётчики обычно определяют как **D**-триггеры (**DFF** и **DFFE**) и используют операторы **IF**.

Ниже приведён файл **ahdlcnt.tdf**, который реализует 16-бит загружаемый счётчик со сбросом.

```
SUBDESIGN ahdlcnt
        (
                clk, load, ena, clr, d[15..0]  :  INPUT;
    q[15..0]  : OUTPUT;
         )
    VARIABLE
                count[15..0]  :  DFFE;
    BEGIN
                count[].clk = clk;
                count[].clrn = !clr;
                IF load THEN
                        count[].d = d[];
                ELSIF ena THEN
count[].d = count[].q + 1;
                ELSE
count[].d = count.q;
                END IF;
                q[] = count[];
    END;
```

В данном файле в секции **VARIABLE** объявлены 16 **D**-триггеров и им присвоены имена от **count0** до **count15**. В операторе **IF** определяется значение, загружаемое в триггеры по фронту синхросигнала (например, если загрузка запускается **VCC**, то триггерам присваивается значение **d[]**).

ЗАКЛЮЧЕНИЕ

Разработка цифровых схем с применением ПЛИС является основным методом проектирования сложных цифровых устройств. Степень интеграции и диапазон частот кристаллов ПЛИС позволяет формировать сложные проекты блоков радиоаппаратуры на базе современных методов цифровой обработки сигналов. Для массового выпуска устройств проект ПЛИС может быть преобразован в вариант изготовления микросхем на базе полузаказных матриц или базовых матричных кристаллов, что снизит стоимость. Если требуется вносить изменения в проект в процессе эксплуатации, то программируемые элементы ПЛИС позволят изменять внутреннюю структуру микросхемы непосредственно в системе без демонтажа корпуса. Использование формальных языков описания аппаратуры высокого уровня позволяет повысить эффективность процесса разработки цифровых устройств. Практические рекомендации, приведенные в данном пособии, помогут организовать работу с пакетами САПР для разработки цифровых схем на базе программируемой логики.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Дайте определение ПЛИС, укажите назначение ПЛИС.
2. Укажите типы программируемых элементов.
3. Назовите основные параметры ПЛИС (уровень интеграции / быстродействие).
4. Укажите способы конфигурирования ПЛИС.
5. Архитектура FPGA. Логический блок FPGA (укажите назначение элементов).
6. Архитектура FPGA. Блок ввода/вывода FPGA (укажите назначение элементов).
7. Архитектура CPLD (укажите назначение элементов).
8. Логический блок (макроячейка) CPLD (укажите назначение элементов).
9. Блок ввода/вывода CPLD (укажите назначение элементов).
10. Назначение интерфейса JTAG, задачи периферийного сканирования.
11. В чем смысл СБИС типа «система на кристалле» (SOS)?
12. Способы описания проекта на БИС ПЛИС (на примере пакета MAX PLUS).
13. Назначение системы проектирования MAX PLUS.
14. Схема процесса проектирования (этапы проектирования) БИС ПЛИС.
15. Компиляция проекта в системе проектирования MAX PLUS.
16. Верификация проекта в системе проектирования MAX PLUS.
17. Состав пакета проектирования MAX PLUS.
18. Назовите области применения БИС ПЛИС.
19. Какие языки описания высокого уровня аппаратуры Вы знаете?
20. Составьте описание структуры схемы (рис. 37) на языке VHDL.
21. Составьте описание структуры схемы (рис. 37) на языке AHDL.

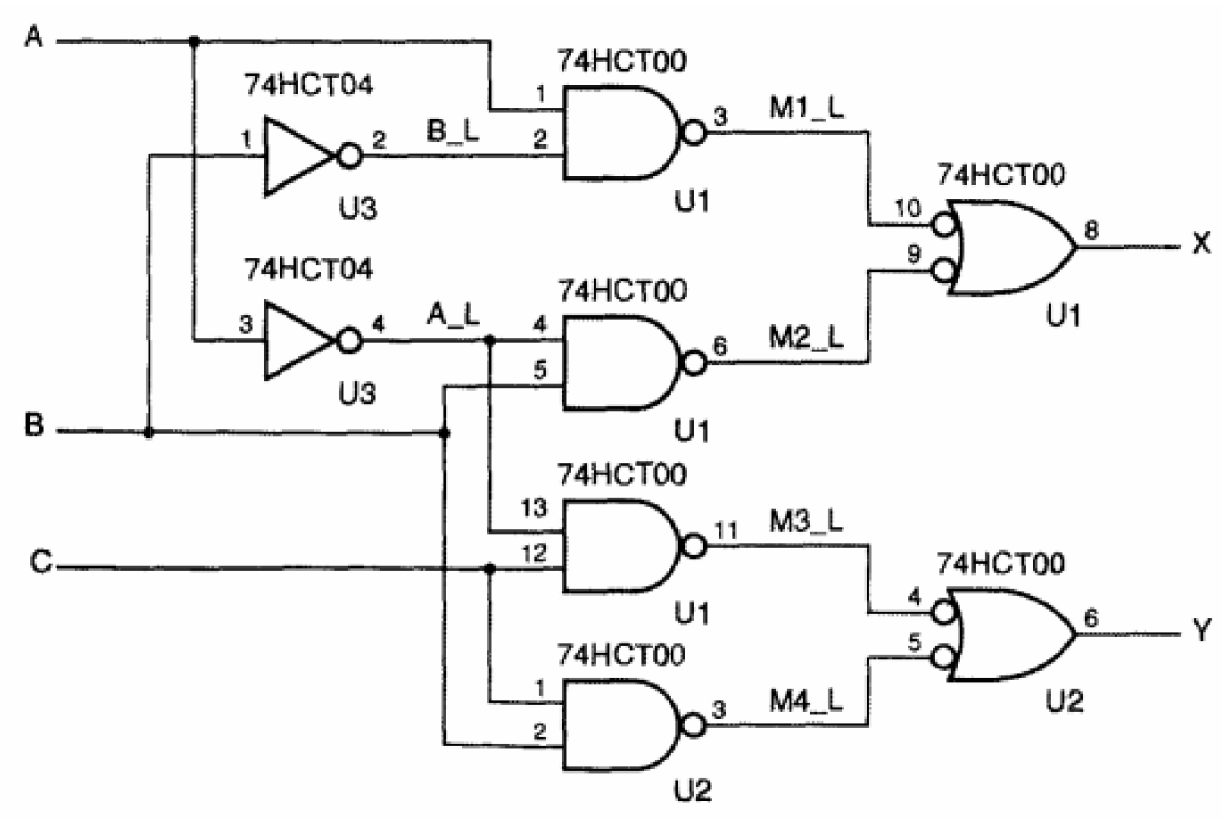

Рис. 37. Комбинационная схема

СПИСОК ЛИТЕРАТУРЫ

1. Стешенко, В.Б. EDA. Практика применения САПР в проектировании радиоэлектронных устройств / В.Б. Стешенко. - М.: Солон-Пресс, 2004. – 300 с.

2. Системы автоматизированного проектирования фирмы ALTERA: учебник для вузов / Д.А. Комолов, Р.А. Мяльк, А.А. Зобенко [и др.] -М.: ИП РадиоСофт, 2002. -355 с.

3. Угрюмов, Е.П. Цифровая схемотехника / Е.П. Угрюмов. – СПб.: БХВ-Петербург, 2004. - 528 с.

4. Грушвицкий, Р. И. Проектирование систем на микросхемах программируемой логики / Р. И. Грушвицкий, А. Х. Мурсаев, Е. П.Угрюмов - Спб.: БХВ-Петербург, 2002. -608 с.

5. Максфилд, К. Проектирование на ПЛИС. Курс молодого бойца / К. Максфилд. — М.: Издательский дом «Додека-XXI», 2007. — 408 с. (Серия «Программируемые системы»).

6. Антонов, А.П. Язык описания цифровых устройств AlteraHDL: практический курс / А.П. Антонов. — М.: ИП РадиоСофт, 2001. — 224 с.

7. Поляков, А. К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры / А. К. Поляков. — М.: Солон-Пресс, 2003. — 320 с. (Серия «Системы проектирования»).

8. Суворова, Е. А. Проектирование цифровых систем на VHDL / Е. А. Суворова, Ю. Е.Шейнин. — СПб.: БХВ-Петербург, 2003. - 576 с.

9. Бибило, П. Н. Синтез логических схем с использованием языка VHDL / П. Н. Бибило. — М.: Солон-Пресс, 2002.- 384 с.

10. Кнышев, Д. А. ПЛИС фирмы XILINX / Д.А. Кнышев, М.О. Кузьмин. - М.: ДОДЕКА, 2001. - 238 с.

Учебное издание

Муравьев Александр Николаевич

РАЗРАБОТКА ЦИФРОВЫХ СХЕМ
НА БАЗЕ ПРОГРАММИРУЕМОЙ ЛОГИКИ

Учебное пособие

Редактор Т. К. К р е т и н и н а
Доверстка Т. Е. П о л о в н е в а

Подписано в печать 15.02.2010. Формат 60x84 1/16
Бумага офсетная. Печать офсетная.
Печ. л. 4,25.
Тираж 100 экз. Заказ . Арт.С -17/2010.

Самарский государственный
аэрокосмический университет.
443086 Самара, Московское шоссе, 34.

Изд-во Самарского государственного
аэрокосмического университета.
443086 Самара, Московское шоссе, 34.