ГЛАВА 5

ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ НА БИС ПРОГРАММИРУЕМОЙ ЛОГИКИ

5.1. Основные семейства ПЛИС фирмы Altera

Программируемые логические интегральные схемы (ПЛИС, англ. programmable logic devices – PLD) применялись в начале в составе микропроцессорных систем для замены стандартной логики, выполнявшей вспомогательные функции. Микропроцессорная техника в это время доминировала, поскольку для нее был найден ясный инженерный подход к проектированию систем. Разработчики быстро освоили магистрально-модульную структуру аппаратных средств на основе ведущего микропроцессора и подчиненных ему интерфейсных БИС. Создание прикладных программ осуществлялось известными методами и средствами, использовался ранее накопленный багаж знаний. Затруднения имели место на этапе комплексной отладки аппаратуры и программного обеспечения в реальном масштабе времени, но они были преодолены созданием метода внутрисхемной эмуляции и комплексов инструментальных средств на основе эмуляторов. Появление персональных компьютеров дало дополнительное ус-

корение процессу обучения специалистов интегрированным методам проектирования аппаратуры и программ.

Основные трудности при развитии идеи ПЛИС (как идеи свободного проектирования и изготовления разработчиком произвольного цифрового устройства) заключались в невозможности использования простого инженерного подхода и необходимости создания новых математических методов синтеза цифровых структур в некотором элементном базисе на основе описания целевой логической функции. Требовалось во взаимосвязи решить следующие задачи:

- определить элементный базис ПЛИС – достаточно развитый, чтобы реализовать необходимые функции целевых устройств, и достаточно простой, чтобы время расчетов на инструментальном компьютере при синтезе не было чрезмерным;
- разработать математические методы синтеза устройств (в выбранном базисе), декомпозиции, компиляции, межэлементной трассировки, функционального моделирования и временного анализа;
- создать интегрированную систему проектирования цифровых устройств на ПЛИС.

Вначале появились микросхемы типа PLA (*программируемые логические матрицы – ПЛМ*, из отечественных К556РТ1), их программирование осуществлялось в кодах через заполнение таблицы истинности. Позже появились микросхемы типа PAL и стали применяться языки программирования ассемблерного типа, как например PALASM. В настоящее время БИС программируемой логики имеют степень интеграции до нескольких миллионов эквивалентных вентилей, быстродействие (ввода/вывода) до 400МГц. В качестве средств описания проектов применяются языки высокого уровня типа HDL (Hardware Description Language), например AlteraHDL, VHDL, Verilog HDL.

Новый импульс развитию ПЛИС сообщило развитие коммуникационных технологий. Именно здесь, на больших потоках и предельных скоростях обработки информации стали проявляться принципиальные ограничения микропроцессоров, связанные с избыточностью их архитектуры. ПЛИС позволяет реализовать специализированную структуру, поэтому при одинаковой тактовой частоте реализованное устройство имеет существенное преимущество в быстродействии. С этой особенностью связана и другая быстро развивающаяся область применения ПЛИС – реализация функций цифровой обработки сигналов.

Из наиболее известных производителей ПЛИС следует отметить фирму Altera. Небольшая, вначале, компания удачно решила перечисленные выше задачи путем постепенного согласованного усложнения элементной базы и средств проектирования. Ее успехи ко второй половине 90-х годов вывели ее в число основных производителей микросхем ПЛИС.

Основные семейства ПЛИС фирмы Altera приведены ниже в таблице.

	Семейства						
Параметры	**MAX 3000**	**FLEX 6000**	**MAX 7000**	**FLEX 8000**	**MAX 9000**	**FLEX 10K**	**APEX 20K**
Число экв. вентилей	600-5000	10000-24000	600-5000	6000-12000	2500-16000	10000-250000	113000-2392000
Внутренняя память, бит	-	-	-	-	-	6144 - 40960	24576-442368
Выводов пользователя	34-158	102-218	36-164	68-208	68-208	150-470	128-808
Технология	EEPROM	SRAM	EEPROM	EEPROM	SRAM	SRAM	SRAM

Семейство MAX3000 включает микросхемы со следующими характеристиками:

Тип микросхемы	**Корпуса**	**Входы**	**I/O**	**Триггеры**	**Ячейки**	**Fmax, MHz**	**Ориентировочная цена,$**
EPM 3032A	L44, T44	4	30	32	32	192	2
EPM 3064A	L44, T44 T100	4	30 62	64	64	192	2,5
EPM 3128A	T100, T144	4	76 92	128	128	182	10
EPM 3256A	T144 P208	4	112 154	256	256	156	21

В таблицах этого параграфа использованы следующие обозначения:

Входы – количество входных линий с предопределенными функциями (Clock, Res, OE).

I/O – количество программируемых линий портов типа вход/выход.

Триггеры – количество триггеров, включая внутренние и периферийные.

Fmax – максимальная частота работы счетчиков, реализованных на ПЛИС.

Обозначения корпусов:

L44 – PLCC44	T44 – TQFP44	P208 – PQFP208	G192 – PGA192	B100 – BGA100
L84 – PLCC84	T00 – TQFP100	P240 – PQFP240	G232 – PGA232	B144 – BGA144
	T144 – TQFP144	R240 – RQFP240	G503 – PGA503	B256 – BGA256
	T160 – TQFP160	R304 – RQFP304		B324 – BGA324
				B356 – BGA356

Технология изготовления EEPROM обеспечивает сохранение конфигурации при отключении питания. Число логических эквивалентных вентилей ПЛИС этой серии находится в диапазоне 600–5000, количество программируемых пользователем выводов 34–158, общее количество выводов 44–208. Микросхемы этой серии могут быть запрограммированы с помощью программатора, в этом случае можно использовать все линии I/O.

Кроме того, все ПЛИС этой серии имеют возможность внутрисистемного программирования (in-system programmability) через порт типа JTAG с использованием устройств типа BitBlaster, ByteBlaster и MasterBlaster, тогда 4 линии порта *JTAG* резервируются для этой цели. Выводы имеют возможность эмуляции режимов открытого коллектора и третьего (высокоимпедансного) состояния, существует возможность управления скоростью нарастания выходных сигналов. Может быть использовано не более 6 управляющих сигналов разрешения выхода.

Семейство FLEX6000 включает следующие микросхемы:

Тип микросхемы	Корпуса	Входы	I/O	Триггеры	Ячейки	Объем кода, Кбайт	Fmax, MHz	Ориентировочная цена,$
EPF 6010A	T100, T144 B256	4	77 98 135	880	880	32		18
EPF 6016A	T100 T144 P208, B256	4	77 113 167	1320	1320	32		25
EPF 6024A	T144 P208 P240	4	113 167 195	1960	1960	49		37

В этой таблице параметр «Объем кода» указывает размер массива данных конфигурации для загрузки в ПЛИС.

ПЛИС этого семейства выполнены по технологии SRAM, поэтому конфигурация теряется после отключения питания и ее нужно восстанавливать. Загрузка может быть:

1. активная, когда ПЛИС сама читает данные из внешнего ПЗУ:
   - с последовательным доступом (*Active Serial*), используются ПЗУ серий EPCxx (Altera), однократные, или AT17Cxx (Atmel), стираемые;
   - с параллельным доступом (*Active Parallel*), используются стандартные ПЗУ серий 27xxx, 28xxx, 29xxx;
2. пассивная, когда данные загружаются ведущим процессором:
   - с последовательным доступом (*Passive Serial*), возможна загрузка через ByteBlaster от компьютера или с помощью специальной программы-загрузчика;
   - с параллельным доступом (*Passive Parallel*), загрузка производится с помощью специальной программы-загрузчика.

Число логических эквивалентных вентилей ПЛИС этой серии находится в диапазоне 10000–24000, количество программируемых пользователем выводов 102–218, общее количество выводов 100–256. Видно, что ис-

пользуются корпуса с большим количеством выводов, предъявляющие очень высокие требования к технологии печатных плат и монтажа. ПЛИС этой серии, кроме EPF6016 (Vcc = 5,0 В), имеют напряжение питания 3,3 В. Все ПЛИС этой серии имеют возможность программирования с использованием устройств типа BitBlaster, ByteBlasterMV и MasterBlaster через выделенные линии. Каждый из пользовательских выводов может быть индивидуально переведен в третье состояние. Имеется возможность управления скоростью нарастания выходных сигналов.

Семейство MAX7000 включает микросхемы MAX7000A и MAX7000AE с напряжением питания 3,3 В, MAX7000B с напряжением питания 2,5 В и MAX7000S с напряжением питания 5,0 В, все имеют возможность внутрисистемного программирования. Микросхемы MAX7000AE представляют собой улучшенный вариант и допускают «горячее включение».

Тип микросхемы	Корпуса	Входы	I/O	Триггеры	Ячейки	Fmax, MHz	Ориентировочная цена,$
EPM 7032	L44, T44	4	32	32	32	192	4.5
EPM 7064	L44, T44 T100, B100	4	32 64	64	64	192	6
EPM 7128	L84 T100,B100 T144,B256	4	64 80 96	128	128	182	14
EPM 7256	T100,B100 T144 P208,B256	4	80 116 160	256	256	164	34
EPM 7512AE	T144 P208 B256	4	116 172 208	512	512	119	
EPM 7160S	L84 T100 T160	4	60 80 96	160	160	149	34
EPM 7192S	T160	4	116	192	192	125	38

Каждая линия включает микросхемы типа EPM7032, EPM7064, EPM7128, EPM7256. Низковольтные линии включают микросхемы EPM7160S, EPM7192S, EPM7512A,B. Технология изготовления EEPROM обеспечивает сохранение конфигурации при отключении питания. Число логических эквивалентных вентилей ПЛИС этой серии находится в диапазоне 600–5000, количество программируемых пользователем выводов 36–164, общее количество выводов 44–208. Выводы имеют возможность эмуляции режимов открытого коллектора и третьего (высокоимпедансного) состояния, существует возможность управления скоростью нарастания выходных сигналов. Может быть использовано не более 6 управляющих сигналов разрешения выхода.

Семейство FLEX8000 включает микросхемы:

Тип микросхемы	Корпуса	Входы	I/O	Триггеры	Ячейки	Объем кода, Кбайт	Fmax, MHz	Ориентировочная цена, $
EPF8282A	L84 T100	4	64 74	282	208	5	133	9,5
EPF 8452A	L84,T100 P160	4	64 116	452	336	8	133	16
EPF 8636A	L84 P160 G192,P208	4	64 114 132	636	504	12	133	22
EPF 8820A	T144 P160 G192,P208	4	108 114 148	820	672	16	133	27
EPF 81188A	P208 G232,R240	4	144 180	1188	1008	24	133	34
EPF 81500A	R240 G280,R304	4	177 204	1500	1296	31	133	46

ПЛИС этого семейства выполнены по технологии SRAM, после отключения питания необходима перезагрузка. Число логических эквивалентных вентилей ПЛИС этой серии находится в диапазоне 2500–16000, количество программируемых пользователем выводов 68–208, общее количество выводов 84–304. Корпуса используются как обычные PLCC, так и очень сложные, штырьковые многорядные и с шариковыми выводами. Все ПЛИС этой серии имеют возможность тестирования через порт типа JTAG, программируются с использованием устройств типа BitBlaster, ByteBlasterMV и MasterBlaster через выделенные линии.

Семейство MAX9000 включает микросхемы:

Тип микросхемы	Корпуса	Входы	I/O	Триггеры	Ячейки	Fmax, MHz	Ориентировочная цена,$
9320	L84 P208 B356	4	56 128 164	484	320	144	59
9400	L84 P208 R240	4	55 135 155	580	400	118	74
9480	P208 R240	4	142 171	676	480	144	121
9560	P208 R240 B356	4	149 187 212	772	560	144	146

ПЛИС этой серии имеют организацию матрицы, аналогичную сериям FLEX6000 и FLEX8000, но изготовлены по технологии EEPROM, что обеспечивает сохранение конфигурации при отключении питания. Число

логических эквивалентных вентилей ПЛИС этой серии находится в диапазоне 600–5000, количество программируемых пользователем выводов 168–216, общее количество выводов 84–356. Из-за использования сложнейших корпусов ПЛИС этой серии программируются только внутрисистемно при помощи устройств типа BitBlaster и ByteBlasterMV.

Семейство FLEX10K включает микросхемы:

Тип микросхемы	Корпуса	Входы	I/O	Ячейки	Объем кода, Кбайт	RAM, бит	Fmax, MHz	Ориентировочная цена, $
10K10	L84 T144 P208	4	55 98 130	576	15	6144	204	22
10K20	T144 P208 R240	4	98 143 185	1152	29	12288	204	39
10K30	P208 R240 B356	4	143 185 242	1728	46-58	12288	204	50
10K40	P208 R240	4	143 185	2304	61	16384	204	105
10K50	R240 B356 G403	4	185 270 306	2880	76-96	2480	204	91
10K70	R240 G503	4	185 354	3744	109	18432	204	222
10K100	G503	4	402	4992	146	24576	204	

Архитектура этого семейства отличается от всех рассмотренных ранее наличием нескольких встроенных блоков памяти размером 2048 бит каждый (4096 бит в FLEX10KE). В семействе имеются линии с напряжениями питания 5 В, 3,3 В и 2,5 В.

Семейство APEX20K включает БИС очень высокой степени интеграции и предназначено для построения вычислительных систем на одном кристалле.

Микросхемы имеют корпуса с общим количеством выводов от 144 до 984. Возможно до восьми глобальных синхросигналов. Каждый из пользовательских выводов может быть индивидуально переведен в третье состояние.

Напряжение питания может быть равным 1,8 В и 2,5 В, но при этом обеспечивается совместимость по входам и выходам только со схемами с аналогичными логическими уровнями.

Для совместимости с уровнями 3 В- и 5 В-схем на второй вывод питания должно быть подано напряжение 3,3 В. Это называется MultiVoltI/O интерфейс.

Тип микросхемы	Корпуса	Входы	I/O	Ячейки	Объем кода, Кбайт	RAM, бит	Fmax, MHz	Ориентировочная цена, $
EP20K30E	T144 B144 P208 B324	6	86 87 122 122	1200	42	24576	370	22
EP20K60E	T144 B144 P208 P240 B324	6	86 87 142 145 190	2560	79	32768	370	40
EP20K100E	T144 B144 P208 P240 B324	6	86 87 142 176 241	4160	124	53248	370	54
EP20K100	T144 P208 P240 B324	6	95 153 183 246	4160	121	53248	370	54
EP20K160E	T144 P208 P240 B356	6	82 137 169 310	6400	186	81920	370	91
EP20K200E	P208 P240 B356	6	130 162 265	8320	240	106496	370	136
EP20K200	P208 P240 B356	6	138 168 271	8320	239	106496	370	170

Двунаправленный ввод/вывод обеспечивается на частотах до 370 МГц. Поддерживаются режимы LVDS, SSTL, GTL+ ввода/вывода. Имеется встроенная ФАПЧ на линиях синхронизации. В целом, это семейство предоставляет проектировщику настолько большие возможности, что одной из важнейших проблем становится маркетинг – что делать и в расчете на какого потребителя.

Семейство ACEX 1K создано фирмой Altera как альтернатива полузаказным БИС (ASIC – Application Specific Integrated Circuit), поэтому микросхемы этой серии имеют высокую степень интеграции и низкую приведенную стоимость логической ячейки. БИС имеют корпуса с общим количеством выводов от 100 до 484. Высокая степень интеграции достигнута благодаря использоваю низковольтного напряжения питания 2,5 В, характерной особенностью является интерфейс MultiVoltI/O. Двунаправленный ввод/вывод обеспечивается на частотах до 250 МГц.

Семейство включает следующие микросхемы:

Тип микросхемы	Корпуса	Входы	I/O	Ячейки	Объем кода, Кбайт	RAM, бит	Fmax, MHz	Ориентировочная цена, $
EP1K10	T100 T144 P208 B256	6	60 86 124 124	576	22	12288	250	10
EP1K30	T144 P208 B256	6	96 141 165	1728	58	24576	250	16
EP1K50	T144 P208 B256	6	96 141 180	2880	76	40960	250	22
EP1K100	P208 B256	6	141 180	4992	164	49152	250	34

Организация матриц различных семейств ПЛИС Altera, структура логических ячеек, динамические параметры и другая техническая информация приведены в каталогах фирмы Altera, часть информации в переводе можно почерпнуть в [2]. Мы на этом останавливаться не будем, поскольку считаем основной целью этой главы рассмотрение всего цикла реализации на ПЛИС проекта цифрового устройства, особенно с использованием описания на языке AHDL.

5.2. Система проектирования MAX+plusII

В качестве средства автоматизации проектирования устройств на основе собственных микросхем фирма Altera разработала систему MAX+plusII. Эта система состоит из 11 программ-обработчиков проекта (приложений), которые функционируют под управлением программы Manager. Система является полноцикловой, т.е. поддерживает не отдельные этапы проектирования (как системы типа CAD – computer aided design), а сквозной процесс от ввода и контроля описания до программирования микросхемы ПЛИС. Такие системы получили название EDA – electronic design automation.

Вид окна Manager системы MAX+plusII с открытым списком приложений приведен на рис. 5.1.

Проект в системе MAX+plusII может быть иерархическим и состоять из набора модулей. Каждый модуль содержит описание части проекта, формой описания может быть графическое представление принципиальной схемы, текст на языках AHDL/VHDL/Verilog, логико-временные диаграммы функционирования.

Для создания исходных модулей в состав приложений включены редакторы: графический – Graphic Editor, текстовый – Text Editor, логико-временных диаграмм Waveform Editor. Любой файл (модуль) может быть

представлен в графическом виде как символ. Это можно сделать явно с использованием редактора символов – Symbol Editor. Далее эти символы могут быть использованы при создании иерархического проекта.

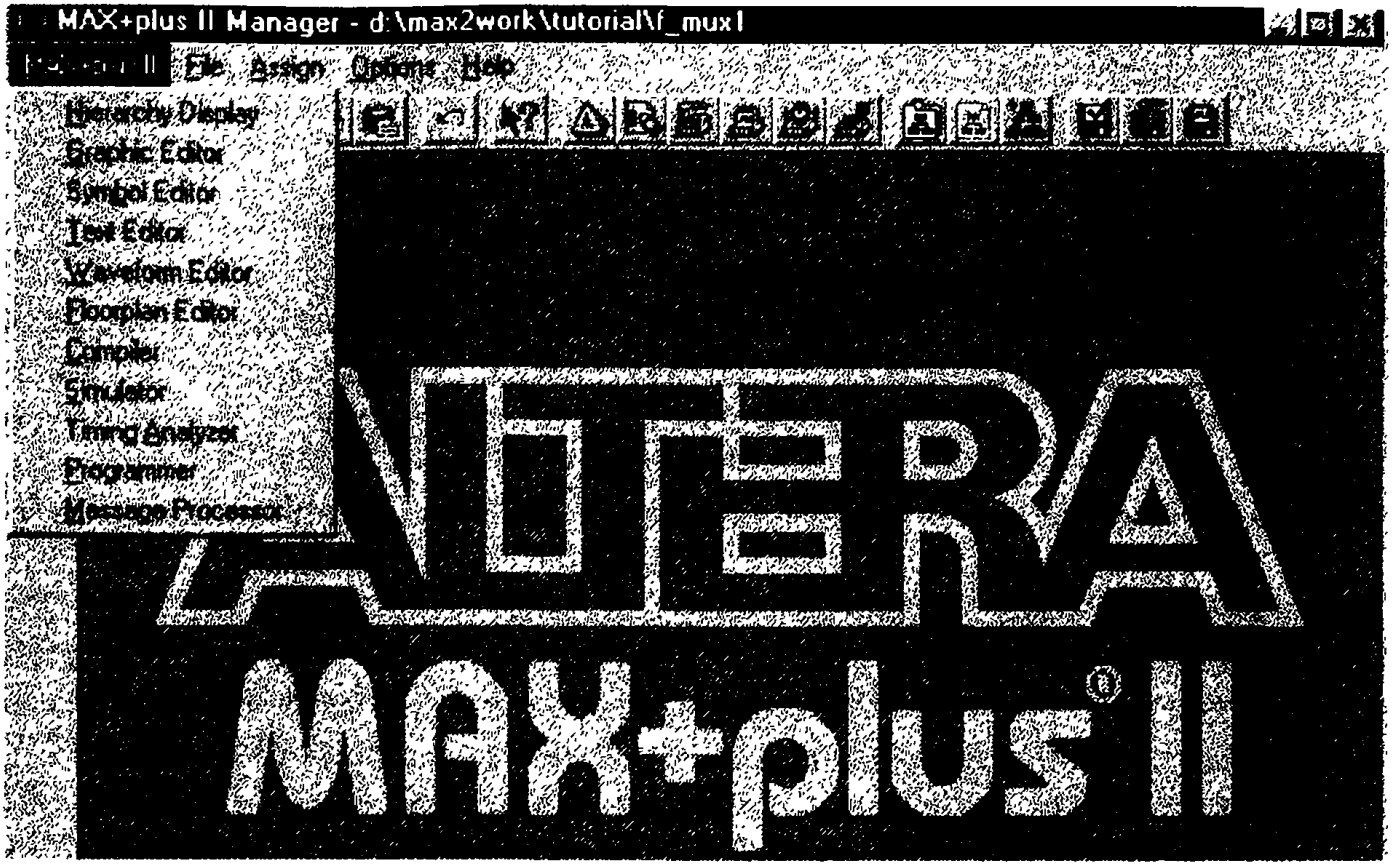


Рис. 5.1. Вид окна Manager системы Max+plusII с открытым списком приложений

Пользователю доступны обширные библиотеки *примитивов, макрофункций* и стандартных компонентов 74 серии. При создании графических модулей возможен импорт файлов из известной системы OrCAD. Система MAX+plusII работает в среде Windows и в полной мере использует преимущества многооконного интерфейса. В частности, различные исходные модули можно одновременно просматривать в окнах с помощью различных редакторов.

В приложение Compiler входят все средства обработки исходных модулей, включая трансляцию, размещение и трассировку на ячейках определенного типа микросхемы ПЛИС, связывание. Перед трансляцией с помощью препроцессоров различные формы описания приводятся к единообразному представлению. В процессе трансляции формируются как целевые файлы, так и вспомогательные (служебные). Перечень формируемых файлов определяется опциями (ключами) трансляции.

Анализ иерархического проекта может быть выполнен с помощью приложения Hierarchy Display. Отображаются все модули проекта и их взаимосвязи, а также все типы файлов, сформированные в процессе обработки проекта.

Перед программированием ПЛИС целесообразно выполнить проверку функционирования спроектированного устройства. Такая проверка на уровне моделирования может быть выполнена с помощью приложения Simulator. Входными данными для него являются файл проекта после трансляции и тестовые векторы, а результатом - логико-временные диаграммы функций выходов.

Следует отметить, что при функциональном моделировании проверяется первый вариант проекта, который не учитывает особенностей реализации на конкретной микросхеме ПЛИС. После успешного выполнения этого этапа проект привязывается к структуре микросхемы, система выполняет конфигурацию ячеек ПЛИС и трассировку связей между ними. При этом второй вариант приобретает некоторую избыточность, например из-за необходимости увеличить коэффициент разветвления по определенному выходу.

После завершения проверки функционирования возможен анализ временных параметров, определение критического пути и т.д. с помощью приложения Timing Analyser. И в этом случае результаты получаются с помощью моделирования на инструментальном компьютере.

Увидеть связи логических элементов внутри ПЛИС и назначения выводов микросхемы, изменить конфигурацию вручную можно с помощью редактора FloorPlan Editor.

На завершающем этапе работы осуществляется программирование микросхемы ПЛИС с использованием приложения Programmer.

Приложение Message Processor формирует сообщения об ошибках трансляции проекта и другую служебную информацию.

В целом, проектирование цифровых систем на основе БИС программируемой логики связано с необходимостью изучения большого объема информации по нескольким направлениям: характеристики и особенности семейств ПЛИС, методы работы с использованием определенной системы проектирования, спецификации языка HDL. Чрезвычайно важен правильный подход и очередность изучения информации. По нашему мнению, неудачи многих попыток связаны с тем, что в начале изучалась внутренняя структура микросхем ПЛИС, в то время как ключом к успеху является освоение языка HDL и системы проектирования.

Систематическое описание системы проектирования MAX+PlusII осложняется тем, что состав меню команд и опции команд зависят от режима работы и вызванного приложения. Кроме того, система весьма развитая и многие ее возможности на первых этапах работы могут не использоваться. Все это привело нас к мысли создать комплекс учебных средств для изучения проектирования цифровых систем на основе БИС программируемой логики фирмы Altera с применением системы проектирования MAX+PlusII и построить его по принципу «делай как я». Это обеспечивает быстрое практическое освоение методов проектирования и позволяет добиться ре-

ального результата. Дальнейшее самостоятельное продвижение облегчается наличием развитого режима поддержки Help системы MAX+PlusII.

Материал первых работ учебного практикума излагается в параграфах 5.3–5.5, основные конструкции языка AHDL рассмотрены в параграфе 5.6, а состав и особенности всего комплекса – в параграфе 5.7. Лаборатория «Микропроцессорные системы» МИФИ на базе этого комплекса проводит учебные циклы «Проектирование цифровых устройств на ПЛИС Altera с использованием системы Max+plusII» для специалистов промышленности.

5.3. Графический ввод схемы и симуляция в системе MAX+plusII

Предположим, что проектируемое устройство определено булевым уравнением, а описание проекта в системе MAX+plusII предполагается выполнить в графической форме с использованием библиотеки примитивов. Последовательность решения такой задачи следующая:

- исходя из уравнения необходимо определить количество переменных и построить таблицу истинности;
- используя графический редактор, ввести схему устройства. Начинать рекомендуется с входных портов, количество которых определяется количеством переменных в уравнении. Далее анализируется вид членов уравнения и вызываются соответствующие примитивы из библиотеки системы MAX+plusII. В завершение проводятся межсоединения (цепи и шины), вызываются примитивы выходных портов;
- используя редактор временных диаграмм, на основе таблицы истинности формируются тестовые векторы для проверки (верификации) соответствия введенной схемы и первоначального уравнения. Для этого используется функциональная симуляция;
- выполняется трансляция проекта, вызывается симулятор, который на основе тестовых векторов формирует диаграмму выходной функции устройства;
- сравнивая диаграмму состояний функции с таблицей истинности делается заключение о правильности функционирования разработанного устройства.

В качестве примера в данной работе будем рассматривать цифровое устройство, которое выполняет логическую функцию

$$f = x_1 x_2 + x_2 \overline{x}_3$$

Таблица истинности имеет следующий вид

x1	x2	x3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Осуществим ввод принципиальной схемы устройства с использованием графического редактора системы MAX+plusII, выполним трансляцию проекта с использованием приложения Compiler и проверим выполнение таблицы истинности, используя приложение Simulator.

Определение имени проекта. Разрабатываемое устройство представляется в системе MAX+plusII как проект. В начале работы с системой необходимо определить текущий проект, т.е. указать его имя и директорию. Выберем директорию d:\max2work\tutorial, а в качестве имени проекта укажем graphic1. Из меню Manager выберите File|Project|Name, откроется диалоговое окно, приведенное на рис. 5.2. Имя тома и директория выбираются с помощью соответствующих меню, имя файла вводится в строке File Name. Завершается определение вводом OK. Имя проекта отобразится в титульной строке окна Manager.

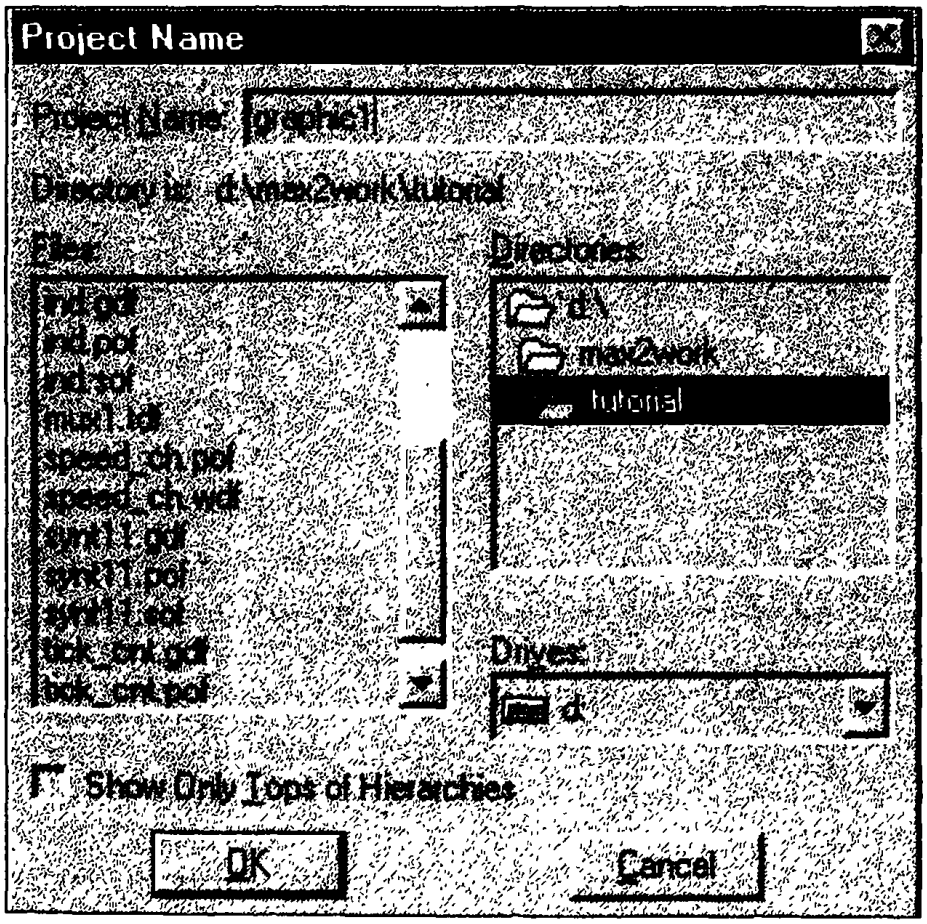


Рис. 5.2. Вид диалогового окна определения имени проекта

Использование графического редактора. Для вызова графического редактора нужно в меню Manager выбрать Max+plusII |Graphic Editor. Откроется окно графического редактора, в титульной строке окна появится сообщение (Untitled1 - Graphic Editor), говорящее о том, что текущим приложением системы MAX+plusII является графический редактор и открыт неименованный файл. Строка меню Manager содержит имена команд и набор инструментальных панелей, которых не было на рис. 5.1, т.е. вид окна Manager зависит от текущего приложения. Чтобы узнать назначение каждой панели, нужно навести на нее указатель мыши, информация высветится под окном в строке состояния.

Графическому файлу со схемой узла необходимо присвоить имя с расширением .gdf (Graphic Design File). Для этого выберите File|Save As и в строке File Name появившегося диалогового окна укажите имя graphic1.gdf, введите OK.

Для ввода графических изображений элементов будем импортировать их из библиотеки, которая в этой системе называется Primitives. Дважды щелкните мышью в центре экрана графического редактора. Откроется диалоговое окно, в котором в меню Symbol Libraries указанная библиотека находится по адресу d:\maxplus2\max2lib\prim. Дважды щелкните мышью по этой строке, в меню Symbol Files появится список логических элементов. Двойной щелчок по имени *and2* приводит к копированию элемента в окно графического редактора в позицию, определенную ранее курсором. Щелчок мышью по элементу производит его выбор, о чем свидетельствует окрашивание в красный цвет. После этого, передвигая курсор мыши при нажатой кнопке 1, можно двигать элемент по окну редактора. Для определения положения элемента полезна сетка, которая появляется при активизации опции Option|Show guidelines.

Для реализации функции *f* нужен еще один элемент *and2*, два элемента *or2* и элемент *not*. Введите эти элементы (примитивы) указанным выше образом.

После того, как логические элементы введены, нужно ввести символы входных и выходных портов. Их примитивы находятся в той же библиотеке под именами *input* и *output*. Введите три входных порта и один выходной, чтобы получить вид экрана, приведенный на рис. 5.3.

Далее необходимо присвоить имена всем портам. Для этого дважды щелкните мышью по слову PIN_NAME на символе входного порта, находящегося в верхнем левом углу экрана. Слово засветится, позволяя прямо набрать имя порта. Ввод Enter непосредственно после имени переводит курсор на следующий порт и так далее. Присвойте имена *x2* и *x3* оставшимся двум входным портам и имя *f* выходному порту.

Следующим шагом является ввод линий, соединяющих логические элементы между собой и с портами.

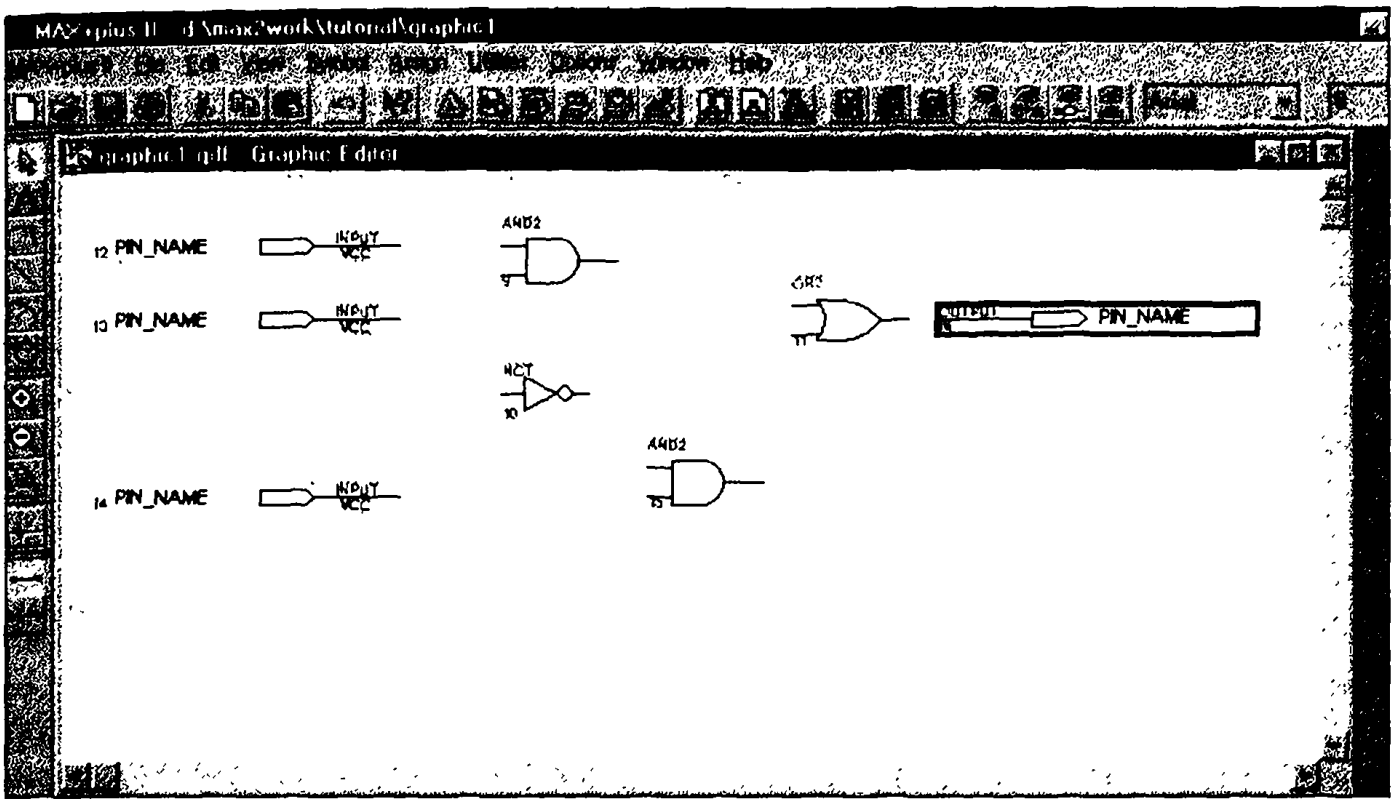


Рис. 5.3. Вид окна графического редактора с изображениями примитивов

Нажмите панель «Выбор объекта» – верхнюю в вертикальном меню слева от окна редактора (на ней изображена стрелка). Если после этого подвести курсор к концу линии вывода (pinstub) порта *x1*, его указатель приобретет вид креста, а после нажатия правой клавиши мыши потянет за собой соединительную линию (node), которая кончается при отпускании клавиши. Соединение выхода и входа двух элементов выполняется в виде горизонтальных и вертикальных отрезков прямых. Любой отрезок можно выделить, щелкнув по нему мышью (он станет красным), и стереть (например клавишей Delete). Проведите все соединения, чтобы схема приобрела вид соответствующий рис. 5.4.

Работа с компилятором. После ввода схемы система проектирования анализирует ее и генерирует булевы уравнения для всех логических функций. Этот этап обработки выполняет приложение-компилятор, который вызывается выбором Max+PlusII|Compiler или щелчком по панели компилятора в меню инструментов. Перед компиляцией нужно выбрать тип микросхемы ПЛИС, на которой будет реализован проект.

Наберите Assign|Device и в открывшемся окне укажите тип микросхемы ПЛИС, установленной на плате – EPF8282ALC84-4. Время перекомпиляции проекта сокращается, если установлена опция Smart Recompile (Processing|Smart Recompile). Если выбрать Processing|Design Doctor, специальная утилита проверит все файлы проекта на соответствие правилам реализации на выбранном типе ПЛИС. Для дальнейшей симуляции понадобится SNF-файл, для его генерации нужно указать Processing | Functional SNF Extractor. После установки опций щелчок по клавише Start запускает

процесс компиляции, после окончания которого высвечиваются сообщения об ошибках и предупреждения. После успешной трансляции закройте окно компилятора (панелью X в верхнем правом углу).

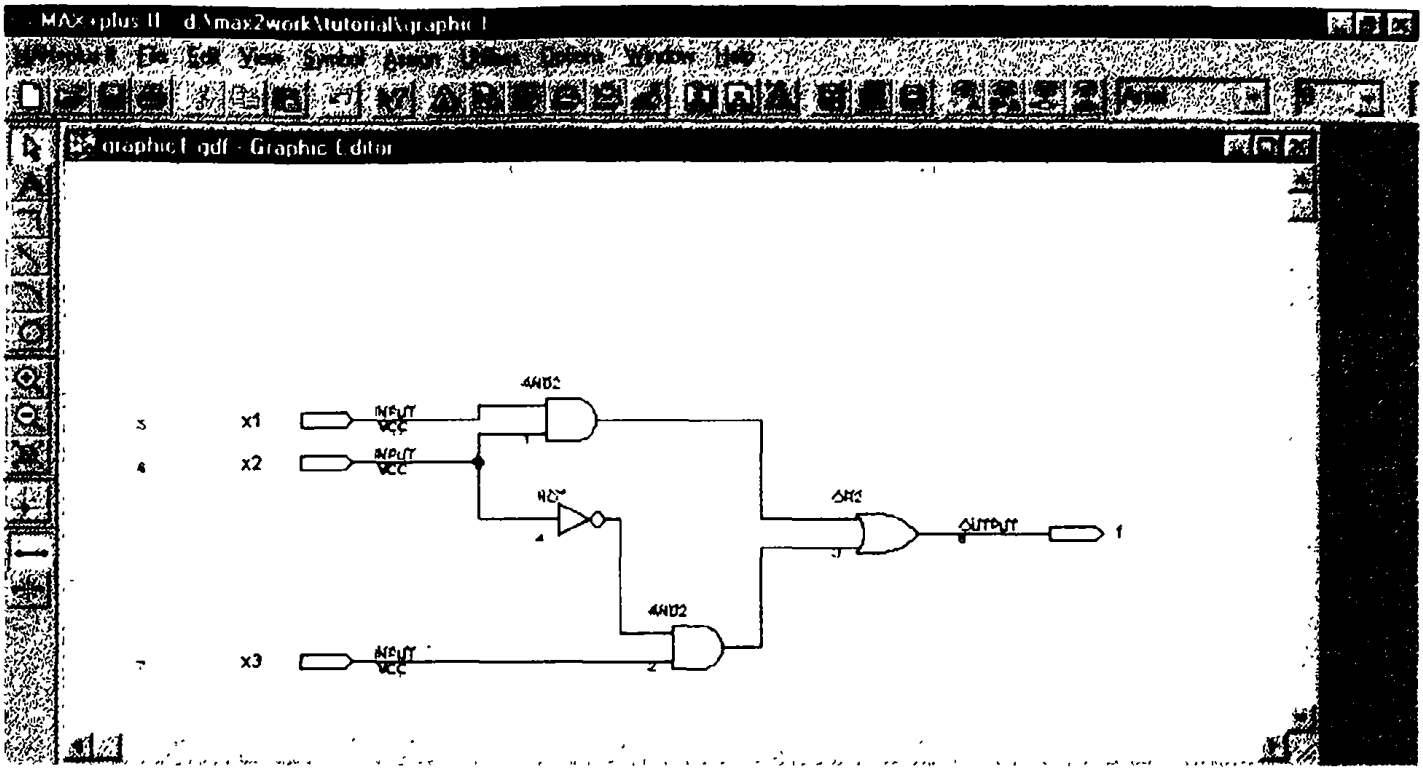


Рис. 5.4. Вид окна графического редактора со схемой устройства graphic1

Симуляция. Симуляцией обычно называют процесс функционального моделирования с использованием программно-логической модели. Перед проверкой функционирования схемы необходимо создать тестовые векторы, которые представляют значения входных сигналов. Мы будем использовать для их создания редактор диаграмм, который выбирается последовательностью команд Max+PlusII|Waveform Editor. Когда окно редактора откроется, создайте файл *graphic1.scf,* последовательностью File|Save As и указанием *graphic1.scf* в строке File Name открывшегося диалогового окна.

Далее определим входные и выходные линии схемы для процесса симуляции. Для этого используем линии, занесенные в SNF-файл (Simulator Netlist File), созданный на этапе компиляции схемы. Введите Node|Enter Node from SNF. Откроется экран, в котором имеется два окна: Available Nodes & Groups и Selected Nodes & Groups. После нажатия List в первом окне появится список входных и выходных линий из SNF-файла. Нужно скопировать входные линии x1, *x2*, *x3* и выходную линию *f* во второе окно. Для этого нужно отметить линии поодиночке или блоком и нажать панель ⇒ между окнами. Чтобы отметить одну линию, нужно щелкнуть по ней мышью. Чтобы отметить блок, нужно протащить указатель по списку при нажатой правой клавише мыши. В завершение введите OK и вернитесь в окно редактора.

Определим параметры процесса симуляции, значения входных переменных для нашей схемы. Вначале вводом File|End Time откроем окно определения времени симуляции, введем значение 160 ns и ОК. Далее определим интервал сетки окна редактора, введя Options|Grid Size и набрав 20 ns. После возврата в окно редактора (вводом ОК) экран системы выглядит так, как приведено на рис. 5.5. На входных линиях значения логического «0», а выходная линия заштрихована, что указывает на неопределенность значения выходной переменной.

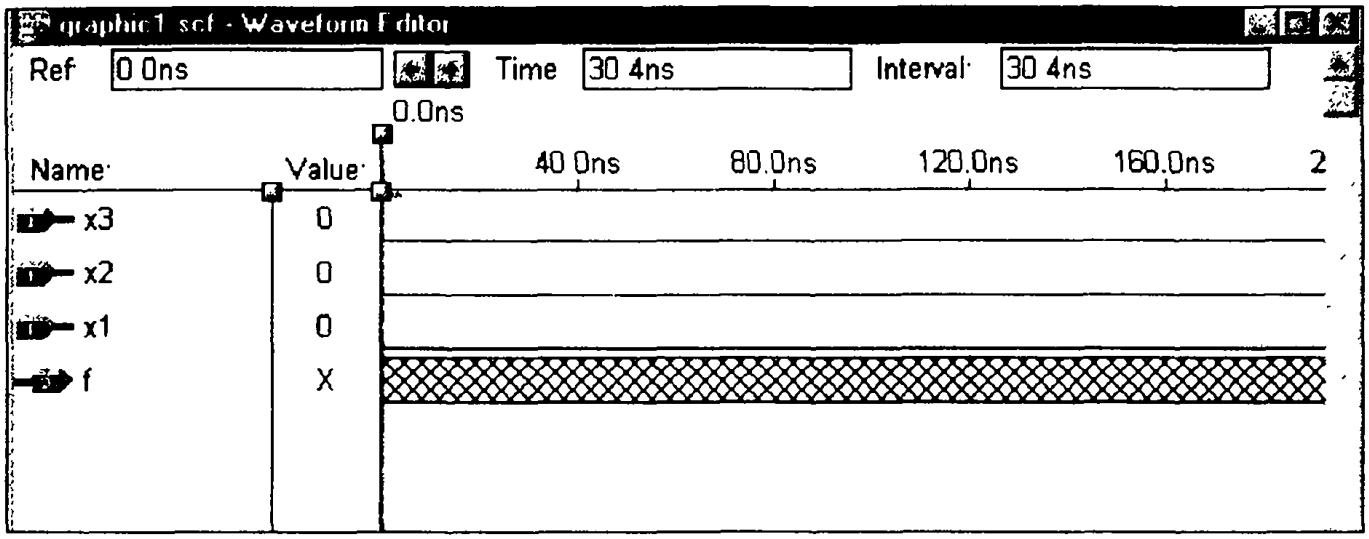


Рис. 5.5. Вид окна редактора диаграмм (определены входные и выходные линии и сетка)

Для полной симуляции функционирования нашего устройства необходимо подать на входы все комбинации значений переменных. Поскольку переменных три, комбинаций $2^3 = 8$. Длительность каждой комбинации при полном времени симуляции в 160 nc равна 20 nc. Таким образом, переменная *x3* должна иметь значение логической «1» в интервалы времени 20–40nc, 60–80nc, 100–120nc. Переменная *x2* должна иметь значение логической «1» на интервалах 40–80nc, 120–160nc, а переменная *x3* на интервале 80–160nc.

Для редактирования временной диаграммы переменной x3 протащите указатель при нажатой правой кнопке мыши над линией логического «0» во втором интервале сетки. Этот интервал будет «залит» черным цветом, в окнах Ref: и Time: отобразятся значения 20 nc и 40 nc. Переведите указатель на панель установки значения «1» в левом вертикальном меню инструментов и щелкните мышью. Временная диаграмма x3 в указанном интервале примет значение «1», заливка исчезнет.

Аналогичными действиями отредактируйте диаграммы переменных x3, x2, x1 так, чтобы они приняли значения, указанные в предыдущем абзаце (рис. 5.6). Сохраните созданный файл (комбинация «горячих» клавиш Ctrl+S).

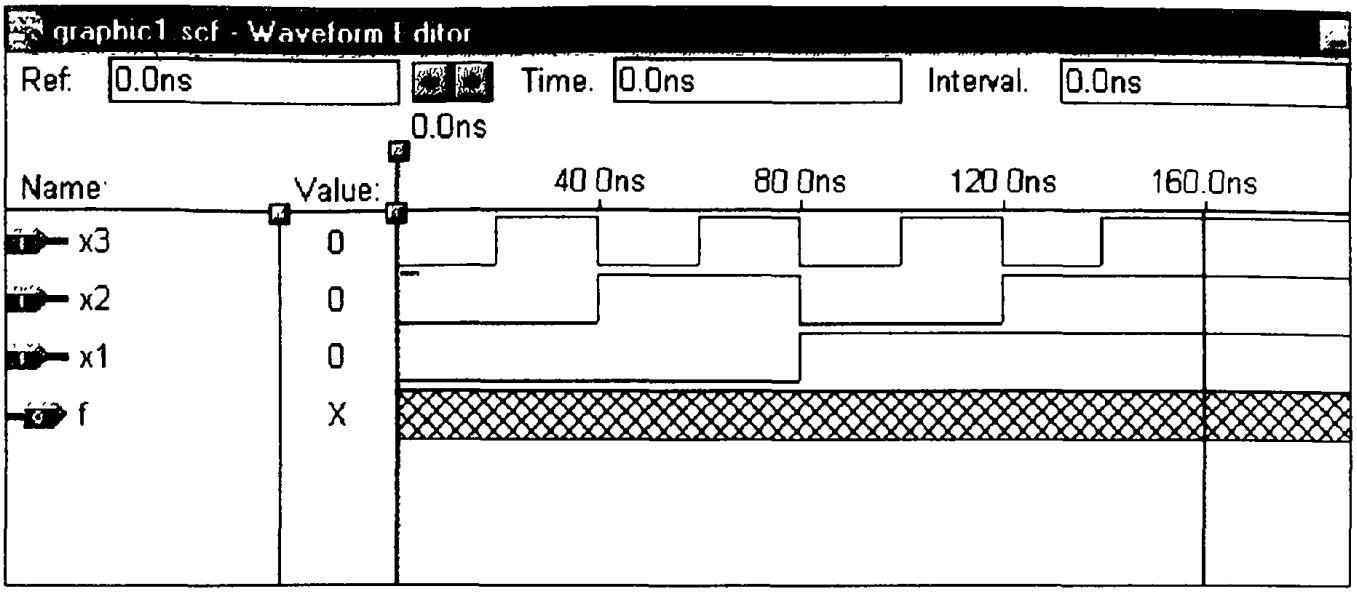


Рис. 5.6. Окно редактора диаграмм с определенными для симуляции диаграммами входных переменных

Для вызова приложения-симулятора выберите Max+plusII |Simulator или нажмите соответствующую панель на верхнем меню инструментов. В открывшемся окне симулятора в заголовке указан режим функциональной симуляции, потому что на этапе компиляции была введена опция Processing | Functional SNF Extractor. В качестве входного файла указан *graphic1.scf*.

Укажите в качестве Start Time значение 0.0 nc, а в качестве End Time значение 160.0 nc, и щелкните по панели Start. После сообщения об отсутствии ошибок щелкните по OK и вернитесь в окно симулятора.

Результаты симуляции записаны в файл *graphic1.scf* и отображаются в окне редактора диаграмм (рис. 5.7).

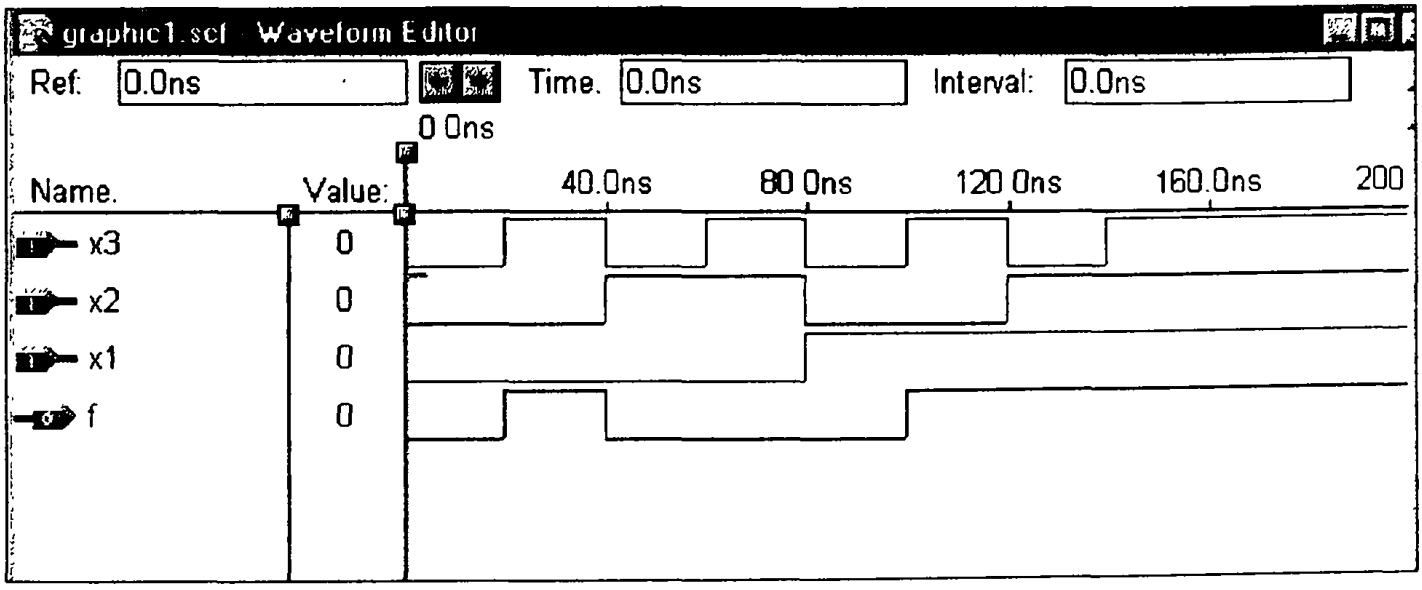


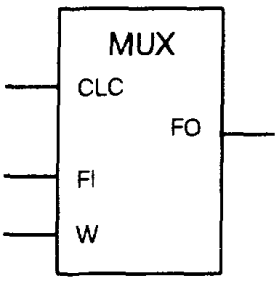
Рис. 5.7. Окно редактора диаграмм с результатами симуляции

Проверьте, что значения функции *f* соответствуют таблице истинности. Закройте окно редактора диаграмм.

5.4. Описание схемы на AHDL, работа с монитором иерархии проекта MAX+plusII

Система MAX+plusII имеет возможность ввода текстового описания цифровой схемы на языке AHDL (Altera Hardware Description Language), созданного с помощью встроенного или любого другого текстового редактора. Текстовые файлы в системе MAX+plusII имеют расширения .tdf (Text Design File). Встроенный текстовый редактор способен оказывать мощную поддержку пользователю, предоставляя шаблоны конструкций языка AHDL.

Опишем в виде текста на языке AHDL мультиплексер, схема которого приведена ниже.



Сохраним описание в файле mux1.tdf. Далее с помощью графического редактора создадим комбинированный проект верхнего уровня, подключив к одному из входов мультиплексера комбинационное устройство, описанное в виде принципиальной схемы в предыдущем параграфе (графический файл graphic1.gdf). Проект верхнего уровня рассмотрим с помощью монитора иерархии проекта, а затем проверим работу устройства методом моделирования с использованием приложения Simulator.

Создание текстового файла. Для вызова приложения-текстового редактора нужно выбрать File|New и в диалоговом окне указать Text Editor file. После ввода OK откроется окно текстового редактора с неименованным файлом. Нужно присвоить ему имя и сохранить (File|Save As) в нужной директории, например, с именем mux1.tdf. Связать проект с текущим файлом можно, выбрав File|Project|Set Project to Current File. Указанное имя mux1 появится в заголовке окна Manager. После этих шагов можно создавать собственно текст описания устройства на языке AHDL. Для этого первоначально целесообразно использовать шаблоны конструкций языка. Шаблоны доступны после выбора Templates|AHDL Template. В диалоговом окне перечислены доступные разделы, первым идет Overall Structure, помогающий формировать структуру программы. Выберите этот раздел, и после появления его содержимого в окне текстового редактора, ознакомьтесь со структурой программы. Видно, что мно-

гие разделы не обязательны. Полезен раздел Title, задающий текст заголовка для файла Report File (.rpt). Его шаблон можно вызвать, дважды щелкнув по разделу Templates|AHDL Template|Title Statement. После двойного щелчка между кавычками это поле «зальется», позволяя прямо вводить текст. Введите слово *Multiplexer*. Получить информацию о назначении и синтаксисе этого поля можно, выбрав Help|AHDL|Design Structure|Title Statement. После ввода заголовка соответствующую строку Title от Overall Structure нужно стереть.

Обязательным разделом является Subdesign, описывающий входные, выходные и двунаправленные порты. Получить информацию об этом разделе можно, выбрав Help|AHDL|Design Structure|Subdesign. Вызовите шаблон, дважды щелкнув по разделу Templates|AHDL Template|Subdesign, после чего укажите в качестве входных портов *clc, fi, w*, а в качестве выходного *fo*, остальное сотрите.

Булевы уравнения связывают значения сигналов на входных и выходных портах. Они вводятся в разделе Logic, который начинается ключевым словом BEGIN и завершается словом END. В этот раздел вложим раздел If Then, описывающий мультиплексер на два направления с управляющим входом *w*. Вызовите шаблон этого раздела, в качестве логического выражения после ключевого слова IF укажите просто имя входного переключающего сигнала *w*. После ключевого слова THEN введите выражение *fo = fi*, выполняющееся при истинном значении выражения («1» на входе *w)*. После ключевого слова ELSE введите выражение *fo = clc*, выполняющееся при ложном значении выражения («0» на входе *w)*. Строку ELSEIF уберите. В результате окно редактора должно содержать текст, приведенный на рис. 5.8.

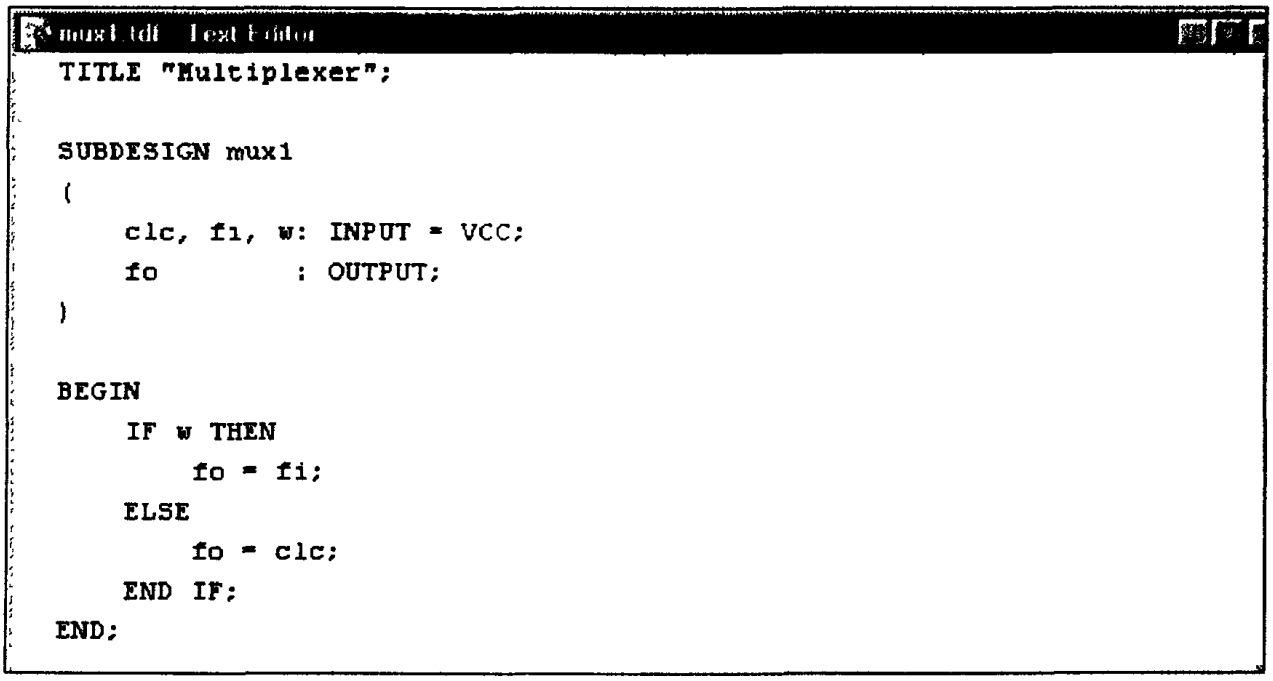


Рис. 5.8. Окно редактора текста с программой на языке AHDL

Сохраните файл (Ctrl+S) и проверьте его на синтаксические ошибки, выбрав File | Project | Save&Check (Ctrl+K). Создайте из него символьный файл *mux1.sym*, выбрав File | Create Default Symbol.

Создание графического файла верхнего уровня. Сейчас с помощью графического редактора мы создадим головной файл проекта *f_mux1.gdf*, включающий как составные части ранее созданные файлы *graphic1* и *mux1*. Схема, которая будет получена в результате, приведена на рис. 5.9.

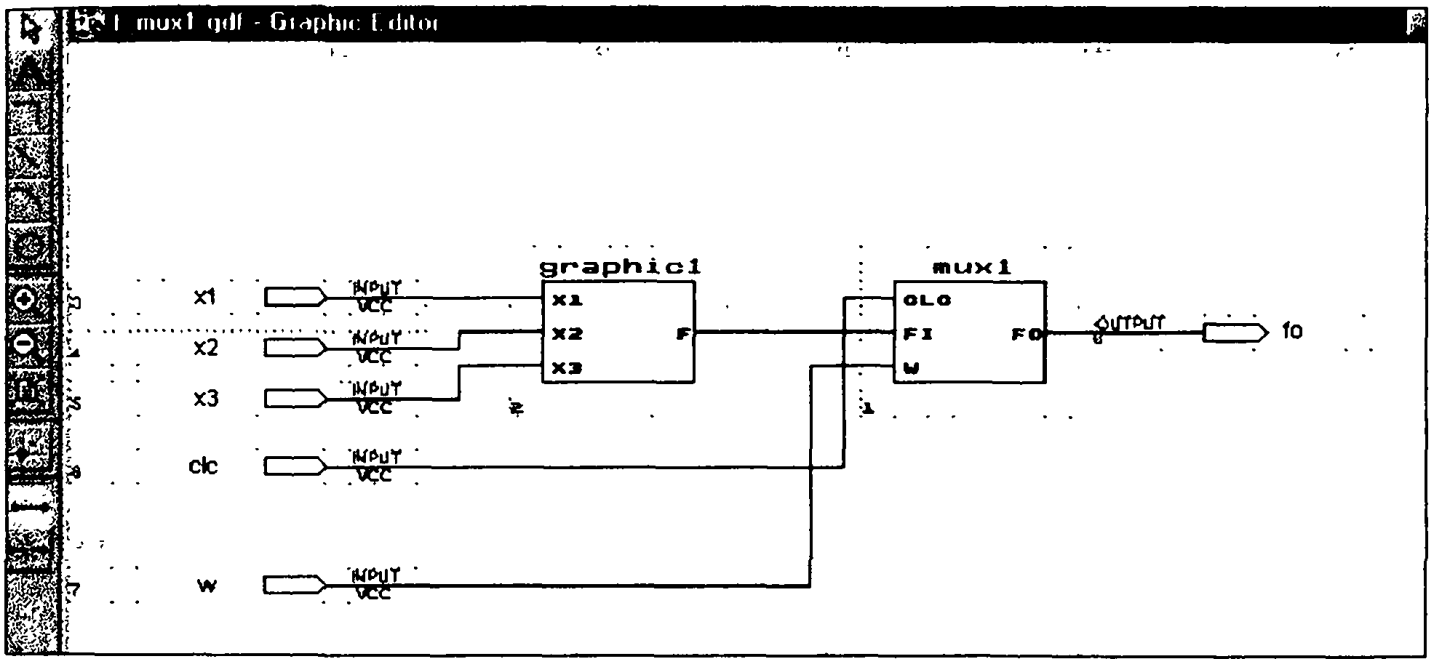


Рис. 5.9. Графический файл верхнего уровня

Создание иерархического проекта, состоящего из различных модулей, имеет следующие преимущества:

- улучшается восприятие проекта и связей внутри него;
- после определения интерфейса между модулями реализацию отдельных модулей можно поручить разным специалистам;
- созданные и отлаженные модули можно использовать в этом проекте и последующих, как библиотечные.

Последовательность действий по созданию иерархического проекта следующая:

- создайте новый графический файл и сохраните его с именем *f_mux1.gdf;*
- укажите в качестве имени текущего проекта *f_mux1* (как делалось ранее или комбинацией клавиш *Ctrl+Shift+J*);
- скопируйте в окно графического редактора символьный файл созданного в предыдущей работе графического файла *graphic1*. Вначале символьный файл нужно создать, открыв файл *graphic1.gdf* и выбрав File | Create Default Symbol. Далее закройте этот файл, вернитесь в головной файл проекта *f_mux1.gdf* и щелкнув дважды мышью в нуж-

ном месте экрана (перед предыдущим изображением) введите в диалоговом окне имя *graphic1*. После ввода OK условное символьное изображение описанной ранее в графической форме схемы появится в выбранном месте экрана;

- введите, используя библиотеку примитивов, пять входных портов и один выходной;
- определите имена всех портов;
- соедините линии выводов входных портов и символьных изображений обоих устройств, а также выход устройства mux1 с выводом выходного порта.

Сохраните получившийся графический файл, который должен содержать изображение, приведенное на рис. 5.9.

Работа с монитором структуры проекта. Структуру (иерархию) проекта можно увидеть, используя специальное приложение. Введите команду Max+plusII |Hierarhy Display или щелкните по соответствующей панели меню инструментов. Откроется окно, в котором проект представлен в виде, соответствующем рис. 5.10. Структура головного проекта изображена в виде дерева, указаны имя каждого файла и исходный тип, а также иконка. Двойной щелчок по иконке открывает исходный файл нижнего уровня с помощью соответствующего редактора. Слева от каждой ветви указаны файлы с такими же именами, но другими расширениями, которые созданы в процессе обработки исходного файла.

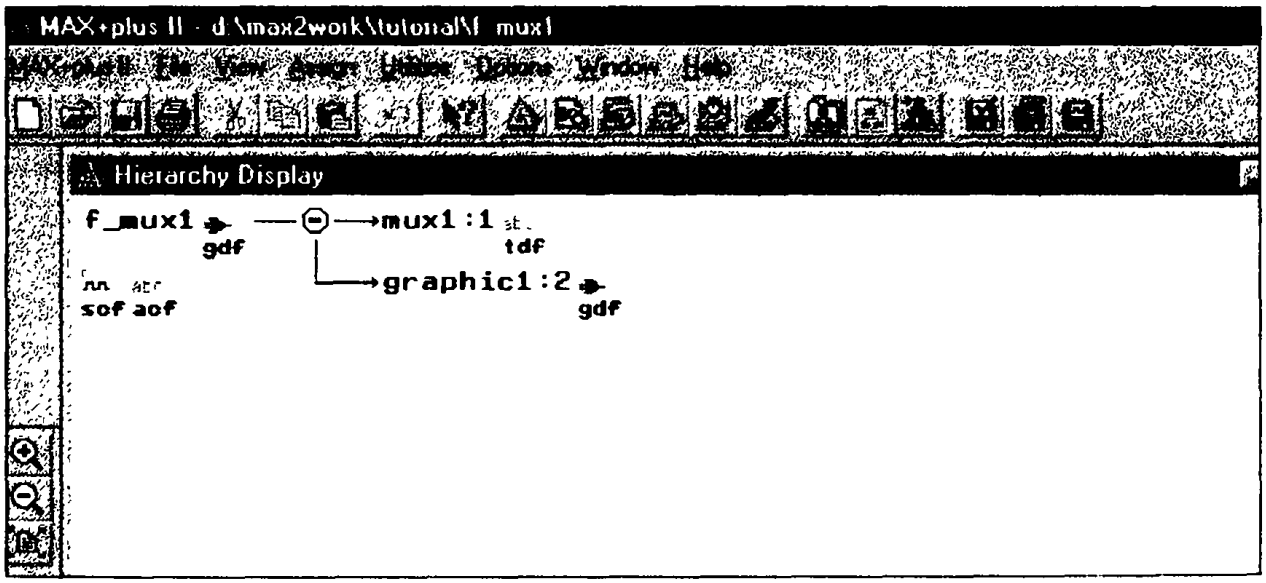


Рис. 5.10. Окно монитора структуры проекта

Симуляция. Выполним симуляцию нашего иерархического проекта. Последовательность действий следующая:

- создадим с помощью редактора временных диаграмм файл, где будут находиться тестовые векторы и результаты симуляции. Для этого от-

кроем редактор временных диаграмм (Max+PlusII|Waveform Editor), сохраним файл с именем *f_mux1.scf* (File|Save As);

- определим входные и выходные линии для симуляции. Для этого откроем графический файл *f_mux1.gdf* (File|Open), вызовем компилятор (Max+PlusII|Compiler) и установив опцию функциональной симуляции (Processing | Functional SNF Extractor) запустим процесс трансляции. Далее закроем файл *f_mux1.gdf* и вернемся т.о. в редактор временных диаграмм. Откроем список доступных в SNF-файле цепей (Node|Enter Node from SNF) и скопируем имена входов и выходов в список выбранных цепей. После ввода OK в окне редактора будут видны входные линии *x1, x2, x3, clc, w* и выходная линия *fo*;
- определим время симуляции и интервал временной сетки. Время симуляции целесообразно установить равным 320нс (File | End Time), а интервал сетки равным 10 нс (Options | Grid Size);
- определим значения входных векторов таким образом, чтобы временные диаграммы соответствовали приведенным на рис. 5.11. Сохраним созданный файл (комбинация «горячих» клавиш Ctrl+S);
- вызовем симулятор (Max+PlusII|Simulator), укажем в качестве End Time: значение 320.0 nc и щелкнем по панели Start. После сообщения об отсутствии ошибок указав OK вернемся в окно симулятора. Результаты симуляции записаны в файл *f_mux1.scf* и отображаются в окне редактора диаграмм (рис. 5.12).

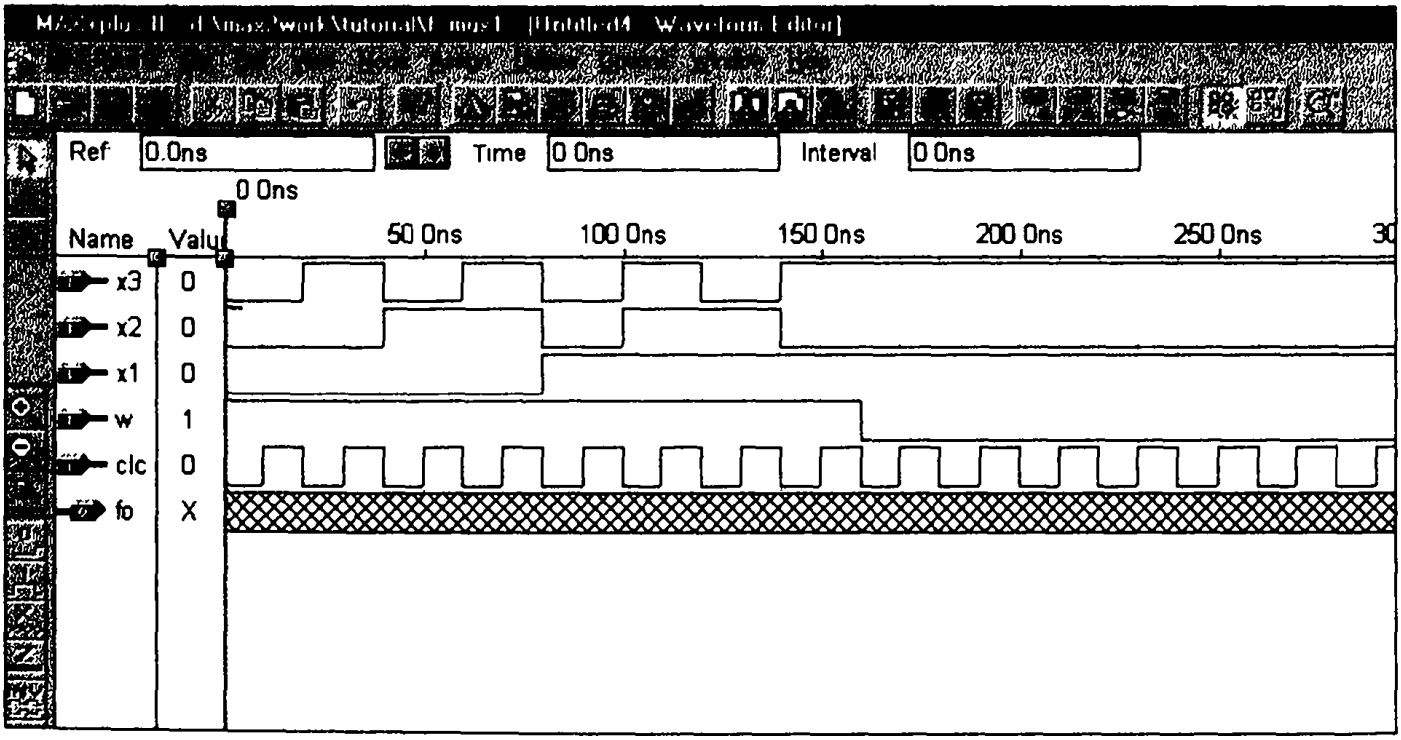


Рис. 5.11. Диаграммы входных сигналов для симуляции

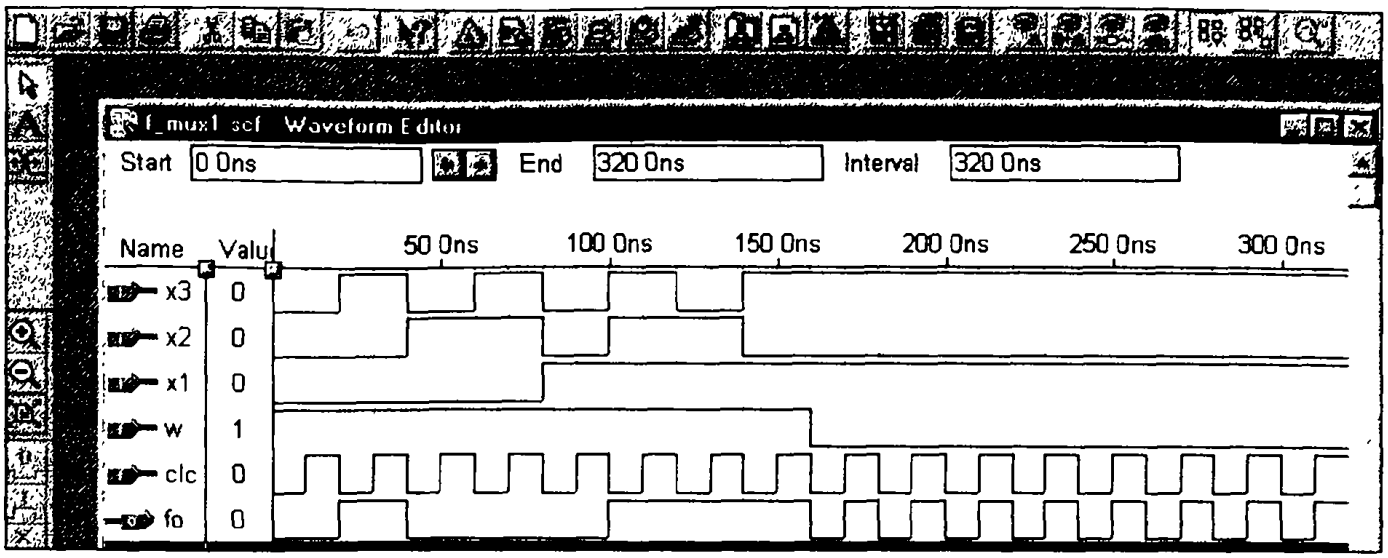


Рис. 5.12. Результаты симуляции проекта *f_mux1*

Проверьте, что значения функции *f* соответствуют таблице истинности. Закройте окно редактора диаграмм.

5.5. Проект АЛУ RISC-микроконтроллера

Степень интеграции современных ПЛИС и уровень развития систем проектирования типа MAX+plusII позволяют рассматривать программируемую логику как альтернативу специализированным и заказным микросхемам. Кроме того, с помощью ПЛИС можно реализовать прототип цифрового устройства или процессора с заданными характеристиками. Далее, через описание на одном из языков HDL (Hardware Description Language) проект может быть перенесен и реализован в виде специализированной БИС.

В этом параграфе будет продемонстрирована возможность реализации на ПЛИС EPF8282ALC84 арифметико-логического устройства (АЛУ) RISC-микроконтроллеров типа PIC16 фирмы MicroChip. Кроме реализации фрагмента АЛУ, проект включает интерфейс ввода данных с переключателя макета и вывода на устройства отображения. Это позволяет провести тестирование разработанного устройства.

RISC-микроконтроллеры представляют собой современное поколение микропроцессорных средств. Они характеризуются следующими особенностями архитектуры:

- код программы и данные выбираются из разных матриц памяти – так называемая гарвардская архитектура. При этом размер слова памяти программ и данных может отличаться. В рассматриваемых микроконтроллерах PIC16 слово команды 14-разрядное, а слово данных 8-разрядное;

- набор команд сокращен, он обеспечивает только простые арифметические операции и минимальный набор условных переходов. Слово команды имеет фиксированную длину, поэтому все команды выбираются за один машинный цикл;
- память программ внутренняя, это ускоряет выборку команд;
- команды слабо кодированы, имеют отдельные поля операций, адресации и данных. Это исключает дешифрацию и ускоряет выполнение команд.

Благодаря таким особенностям все команды, за исключением команд управления, выполняются за один машинный цикл.

Структура устройства, которое мы будем проектировать, имеет вид, приведенный на рис. 5.13.

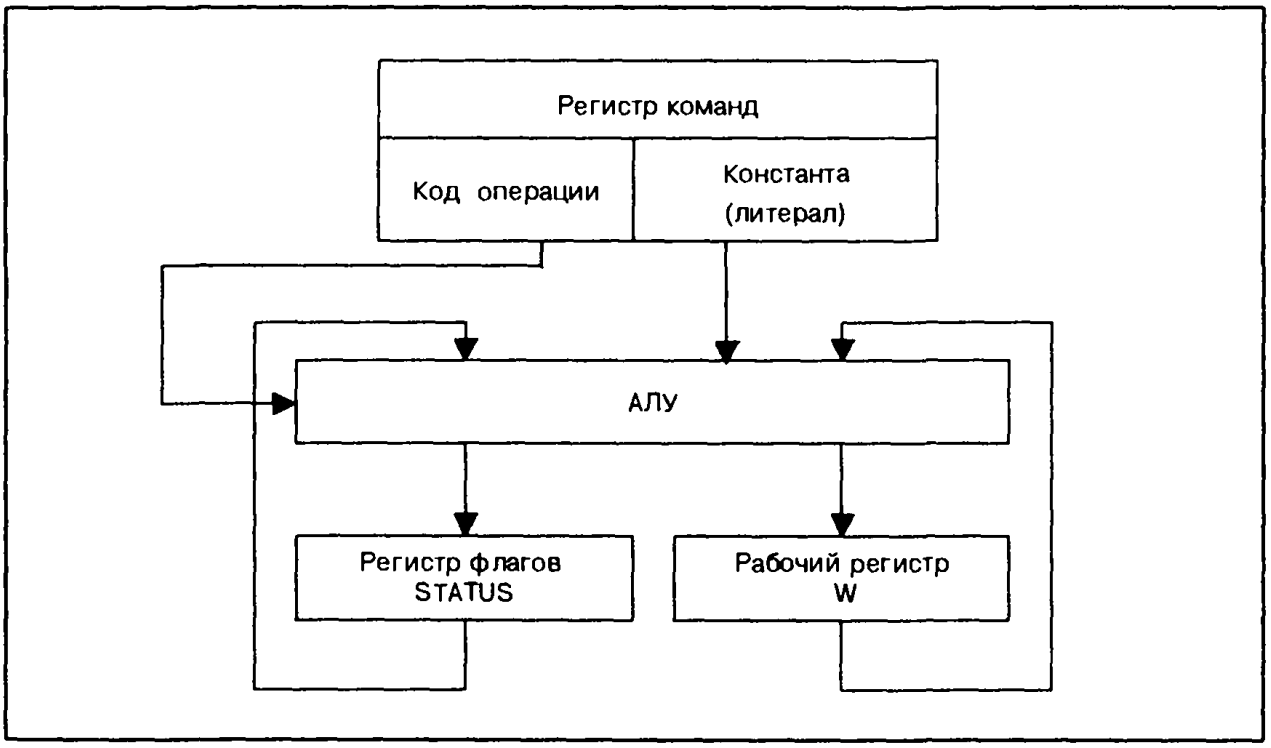


Рис. 5.13. Структура устройства на основе регистрового АЛУ

Устройство, кроме комбинационной схемы 8-разрядного АЛУ, содержит регистр команд, рабочий регистр W, регистр флагов STATUS. АЛУ в соответствии с кодом команды выполняет операции целочисленного сложения и вычитания, сдвиги вправо и влево, логические операции. В двухоперандных командах один из операндов берется из рабочего регистра W, а другой извлекается из регистрового файла или является частью кода команды (константа). В одноперандных командах операнд берется из регистрового файла или рабочего регистра.

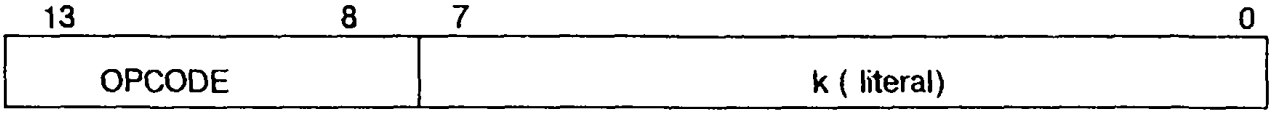
Результатом АЛУ-операции является новое содержимое рабочего регистра W и регистра флагов STATUS. Регистр STATUS включает следующие флаги:

Бит регистра STATUS	Флаг	Признак флага
STATUS.0	C (Carry)	Перенос из старшего разряда
STATUS.1	DC (Digit Carry)	Перенос из младшей тетрады
STATUS.2	Z (Zero)	Признак нулевого результата

При реализации устройства будем рассматривать подмножество команд микроконтроллеров PIC16, которое содержит арифметические команды над литералом и содержимым рабочего регистра.

Результат заносится в рабочий регистр, признаки результата заносятся в регистр флагов. Код операции содержится в команде в поле OPCODE, литерал находится в команде в поле k (literal).

Длина слова команды большинства микроконтроллеров PIC16 равна 14 битам. Формат команд рассматриваемого подмножества следующий:



У рассматриваемого подмножества команд старшие 2 бита содержат код 11, поэтому мы их учитывать не будем. Следующие 4 бита (11..8) содержат уникальный код операции, младшие 8 бит содержат значение литерала (константы).

Таблица реализуемых устройством команд

Мнемоника	Описание	Биты <11..8>	Изменяются флаги
ADDLW	Сложение константы и содержимого регистра W	111x	C,DC,Z
ANDLW	Логическое И (AND) значения константы и содержимого регистра W	1001	Z
IORLW	Логическое ИЛИ (OR) значения константы и содержимого регистра W	1000	Z
MOVLW	Пересылка константы в регистр W	00xx	
SUBLW	Вычитание содержимого регистра W из значения константы	110x	C,DC,Z
XORLW	Исключающее ИЛИ (XOR) значения константы и содержимого регистра W	1010	Z

"x" означает безразличное значение бита.

Addlw

Синтаксис: [метка] addlw k

Операнды: $0 \le k \le 255$

Действие: (W)+k → (W)

Признаки: C,DC,Z

Кодирование:

11	111x	kkkk	kkkk

Описание: содержимое регистра W добавляется к 8-битному литералу 'k', результат помещается в W

Слов: 1
Циклов: 1
Пример: addlw 15h
Перед выполнением
W = 10h
После выполнения
W = 25h

Andlw

Синтаксис: [метка] andlw k
Операнды: 0 ≤ k ≤ 255
Действие: (W) AND k → (W)
Признаки: Z
Кодирование:

11	1001	kkkk	kkkk

Описание: логическое И содержимого регистра W и 8-битного литерала 'k', результат помещается в W.
Слов: 1
Циклов: 1
Пример: andlw 5Fh
Перед выполнением
W = 0A3h
После выполнения
W = 03h

Iorlw

Синтаксис: [метка] iorlw k
Операнды: 0 ≤ k ≤ 255
Действие: (W) OR k → (W)
Признаки: Z
Кодирование:

11	1000	kkkk	kkkk

Описание: логическое ИЛИ содержимого регистра W и 8-битного литерала 'k'. Результат помещается в W.
Слов: 1
Циклов: 1
Пример: iorlw 35h
Перед выполнением
W = 9Ah
После выполнения
W = 0BFh

Movlw

Синтаксис: [метка] movlw k
Операнды: 0 ≤ k ≤ 255
Действие: k → (W)
Признаки: Нет
Кодирование:

11	00xx	Kkkk	kkkk

Описание: 8-битный литерал 'k' загружается в W.
Слов: 1
Циклов: 1
Пример: movlw 5Ah
После выполнения
W = 5Ah

Sublw

Синтаксис: [метка] sublw k
Операнды: $0 \le k \le 255$
Действие: $k - (W) \rightarrow (W)$
Признаки: C,DC,Z
Кодирование:

11	110x	Kkkk	kkkk

Описание: содержимое регистра W вычитается из 8-битного литерала 'k', результат помещается в W.
Слов: 1
Циклов: 1
Пример 1: sublw 02h
Перед выполнением
W = 01h
C = ??
После выполнения
W = 01h
C = 1 ;результат положительный
Z = 0
Пример 2: Перед выполнением
W = 02h
После выполнения
W = 00h
C = 1 ;результат нулевой
Z = 1
Пример 3: Перед выполнением
W = 03h
C = ??
После выполнения
W = FFh
C = 0 ;результат отрицательный
Z = 0

Xorlw

Синтаксис: [метка] xorlw k
Операнды: $0 \le k \le 255$
Действие: $(W)\ XOR\ k \rightarrow (W)$
Признаки: Z
Кодирование:

11	1010	kkkk	kkkk

Описание: логическое ИСКЛЮЧАЮЩЕЕ ИЛИ содержимого регистра W и 8-битного литерала 'k'. Результат помещается в W.
Слов: 1
Циклов: 1
Пример: xorlw AFh
Перед выполнением
W = 0B5h
После выполнения
W = 1Ah

Распределение ресурсов макета. Описание платы, на которой будет проверяться работа проектируемого устройства, приведено в параграфе 5.7. Вначале распределим ресурсы платы для ввода данных, выполнения команд и индикации результатов. Предлагается код команды вводить с

8-разрядного переключателя, а сигнал на выполнение команды – с левой кнопки. Собственно АЛУ с регистрами W и STATUS, естественно, реализуется с помощью ПЛИС. Код из регистра W может быть отображен на двух семисегментных индикаторах, а значение флагов регистра STATUS – с помощью линейки светодиодов.

Описание интерфейса устройства. Рассмотрим и опишем набор сигналов и портов проектируемого устройства на языке AHDL с учетом последующей реализации на нашей плате.

Слово команды разобьем на две части – поле кода операции и поле константы (литерала):

```
OPCODE[3..0], DATA[7..0]: INPUT;
```

Входным также является сигнал от кнопки, по которому выполняется команда:

```
CLK      : INPUT;
```

Выходными являются сигналы с регистра W на семисегментные индикаторы (после дешифрации) и три бита флагов регистра STATUS:

```
WMSB[3..0], WLSB[3..0]   : OUTPUT;
FLAGS[2..0]        :OUTPUT;
```

Декомпозиция. Устройство предлагается разрабатывать как иерархический проект, включающий текстовые файлы (модули) нижнего уровня. Перечень файлов и назначение модулей следующие:

Имя файла	Описание модуля
alublock.tdf	Фрагмент структуры PIC16 (исследуемое устройство), объединяющий дешифратор команд, АЛУ и регистры.
copdecdr.tdf	Дешифратор команд.
alu.tdf	Комбинационное арифметико-логическое устройство.
Lab5.tdf	Интерфейс к LabKit-8000, отображающий порты исследуемого устройства на внешние ресурсы платы.
decode7.tdf	Дешифратор семисегментного дисплея.

Взаимосвязи между модулями проекта легче проследить, имея перед глазами иерархическое представление их набора, как например на рис. 5.14.

Модуль alu.tdf имеет следующий вид:

```
SUBDESIGN ALU
(
    COP[2..0]  : INPUT;
    W[7..0]    : INPUT,
    DATA[7..0] : INPUT;                    -- описание входных
    Z,DC,C     : INPUT,
    ZR,DCR,CR  : OUTPUT;                   -- и выходных сигналов
    RESULT[7..0]: OUTPUT,
)
BEGIN
```

```
case COP[2..0] is                                                    -- дешифратор кода операции
when H"0" =>(DCR,RESULT[3..0]) = (B"0",W[3..0]) + (B"0",DATA[3..0]);   -- ADD
            ( CR,RESULT[7..4]) = (B"0",W[7..4]) + (B"0",DATA[7..4]) + (B"0000",DCR);
when H"1" => RESULT[7..0] = W[7..0] & DATA[7..0];                    -- AND
when H"2" => RESULT[7..0] = W[7..0] # DATA[7..0];                    -- OR
when H"3" => RESULT[7..0] = DATA[7..0];                              -- MOV
when H"4" => RESULT[7..0] = W[7..0] - DATA[7..0];                    -- SUB
                    -- здесь для единообразия оставлен порядок операндов (W - k)
                    (DCR,RESULT[3..0]) = (B"1",W[3..0]) - (B"0",DATA[3..0]);
                    ( CR,RESULT[7..4]) = (B"1",W[7..4]) - (B"0",DATA[7..4]) - (B"0000",!DCR);
                    -- исходный вариант (как в PIC16)
                    -- (DCR,RESULT[3..0]) = (B"1",DATA[3..0]) - (B"0",W[3..0]);
                    --( CR,RESULT[7..4]) = (B"1",DATA[7..4]) - (B"0",W[7..4]) - (B"0000",!DCR);
when H"5" => RESULT[7..0] = W[7..0] $ DATA[7..0];                    -- XOR
-- при неправильном значении кода арифметической операции никаких действий не производится
when others => RESULT[7..0] = W[7..0];                               -- NOP
end case;

if (RESULT[] == 0) then                                              -- формирование флага "нуля"
      ZR = B"1";
else
      ZR = B"0";
end if;

END;
```

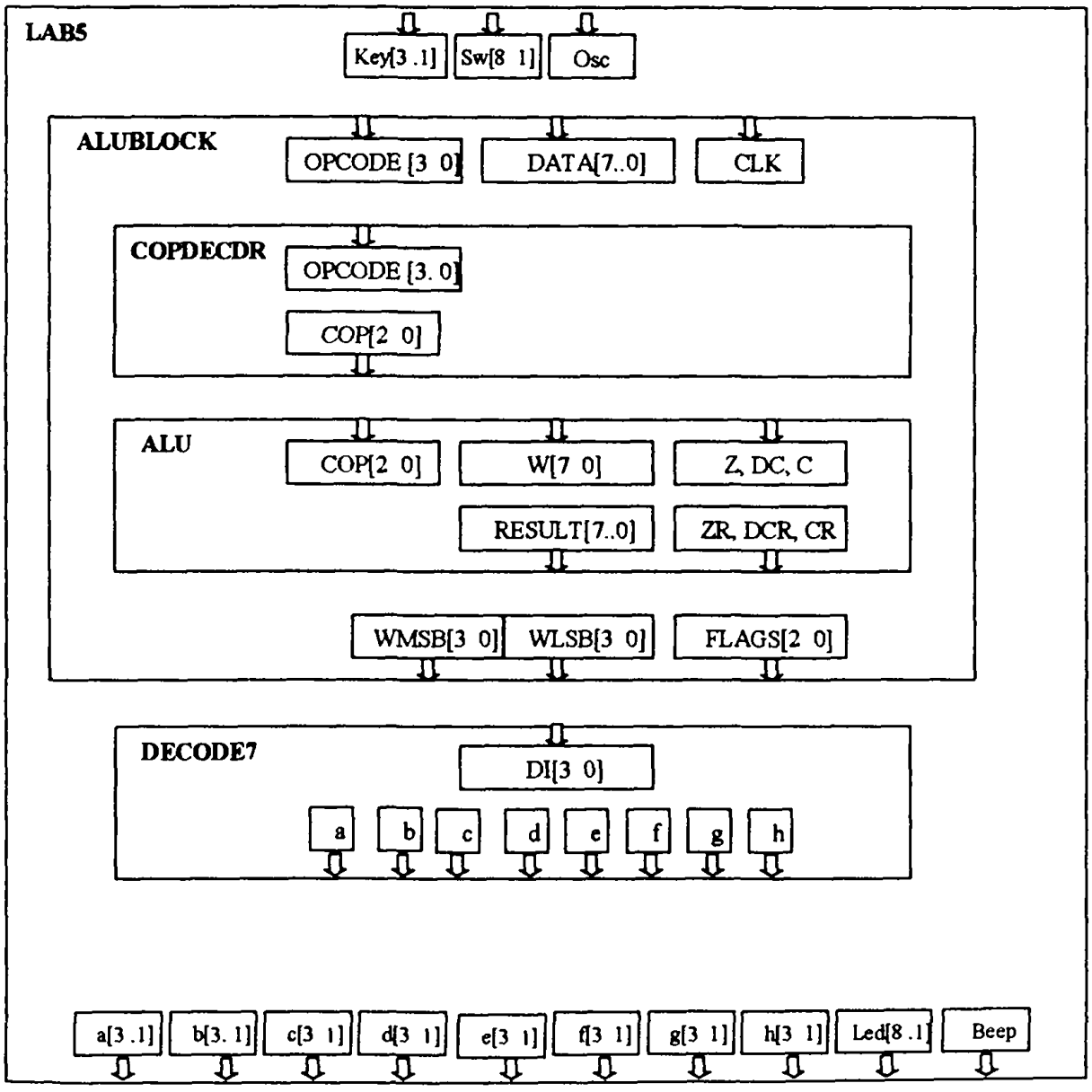


Рис. 5.14. Иерархия программы, описывающей регистровое АЛУ и интерфейс

В секции SUBDESIGN описаны входные и выходные порты: кода операции, внутренних данных из регистра W, данных извне, входные значения флагов, выходные значения флагов, байт результата. Собственно схема АЛУ является чисто комбинационной, как только появились данные на входах, через некоторое время задержки, на выходных линиях получается код результата. стробирующего сигнала нет.

В секции логики этого модуля посредством оператора CASE описаны варианты получения результата при выполнении каждой из команд. Результат операции АЛУ вычисляется с помощью арифметических и логических выражений. Из-за наличия флага полупереноса DC байт результата вычисляется в два приема обработкой тетрад операндов. Далее с помощью оператора IF устанавливаются выходные значения флагов.

Далее рассмотрим файл copdecdr.tdf. Этот модуль формирует из четырех разрядов кода операции всего устройства три разряда кода операции для модуля АЛУ. Его текст имеет следующий вид:

```
SUBDESIGN COPDECDR
(
        OPCODE[3..0]                    : INPUT;
        COP[2..0]  : OUTPUT;
)

BEGIN
        TABLE
                OPCODE[3..0]        =>      COP[2..0];
                B"111X"     =>      H"0"; -- ADDLW
                B"1001"     =>      H"1"; -- ANDLW
                B"1000"     =>      H"2"; -- IORLW
                B"00XX"     =>      H"3"; -- MOVLW
                B"110X"     =>      H"4"; -- SUBLW
                B"1010"     =>      H"5"; -- XORLW
    END TABLE;
END;
```

Наконец, текст головного файла alublock.tdf имеет следующий вид:

```
INCLUDE "COPDECDR.INC";
INCLUDE "ALU.INC";

SUBDESIGN ALUBLOCK
(
    OPCODE[3..0] :Input;
    DATA[7..0]   :Input;
    CLK          :Input;

    WMSB[3..0]   :output;
    WLSB[3..0]   :output;
    FLAGS[2..0]  :output;
)

VARIABLE

    W[7..0]      :DFF;
    STATUS[2..0] :DFF;
    DECODER      :COPDECDR;
    ARITHUNIT    :ALU;
```

```
BEGIN
DEFAULTS
    W[].d = 0;
    STATUS[].d = 0;
END DEFAULTS;

    W[].clk = CLK; -- фронт тактового сигнала
    W[].prn  = VCC;
    W[].clrn = VCC;

    -- новые значения флагов фиксируются только при
    определенных инструкциях
    STATUS[2].clk = CLK&OPCODE[3]; -- Z
    STATUS[1..0].clk = CLK&OPCODE[2]; -- DC,C
    STATUS[].prn  = VCC;
    STATUS[].clrn = VCC;

    DECODER.OPCODE[] = OPCODE[];

    ARITHUNIT.COP[] = DECODER.COP[];
    ARITHUNIT.W[] = W[].q;
    ARITHUNIT.DATA[] = DATA[];
            (ARITHUNIT.Z,ARITHUNIT.DC,ARITHUNIT.C) = STATUS[2..0].q;

    W[7..0].d = ARITHUNIT.RESULT[7..0];
    STATUS[2..0].d = (ARITHUNIT.ZR,ARITHUNIT.DCR,ARITHUNIT.CR);

    WMSB[3..0] = W[7..4].q;
    WLSB[3..0] = W[3..0].q;
    FLAGS[2..0] = STATUS[2..0].q;

    END;
```

Первые две строки представляют собой директивы, указывающие компилятору, что в текст головного модуля включается текст файлов copdecdr.tdf и alu.tdf.

В секции SUBDESIGN описаны входные и выходные порты устройства в соответствии с соглашениями предыдущего параграфа.

В секции описания переменных регистры W и STATUS определены как наборы D-триггеров из библиотеки примитивов, защелкивающих новые значения по фронту тактового сигнала. Третья и четвертая строки секции описывают переменные DECODER и STATUS как устройства, описанные в присоединенных файлах.

В секции логики (начинается с BEGIN) первые три строки определяют значения стробирующего сигнала, входов сброса и установки для триггеров регистра W. Видно, что все разряды регистра управляются одинаково. Следующие четыре строки определяют значения сигналов для триггеров регистра STATUS. Отдельные триггеры (флаги) стробируются поразному. Анализ таблицы команд показывает, что флаг Z обновляется при OPCODE[3] = 1, а флаги C и DC при OPCODE[2] = 1. Стробирующим сигналом для триггеров регистра STATUS должна быть функция И сигналов CLC и соответствующих битов кода операции. Входы сброса и установки всех триггеров регистра STATUS управляются одинаково.

В восьмой строке на вход модуля DECODER подается 4-разрядный код операции с входов устройства OPCODE[]. Далее с выходов модуля DECODER преобразованный 3-разрядный код поступает на входы кода операции модуля АЛУ. В строках 10, 11 и 12 на соответствующие входы этого модуля подаются данные с регистра W, с входов данных устройства, с регистра STATUS. В следующих двух строках результат с выходов модуля АЛУ заносится в регистр W, а флаги заносятся в регистр STATUS.

В последних трех строках содержимое регистра W выводится в виде двух тетрад на выходы всего устройства, а содержимое регистра STATUS – на выходы FLAGS.

Функциональное моделирование – симуляция. Для предварительной проверки устройства на функциональном уровне произведем симуляцию, задав в качестве тестовых векторов коды арифметических и логических команд из рассматриваемого подмножества. Результат выполнения сравним с описанием каждой из команд.

Откроем головной файл проекта alublock.tdf и определим его имя в качестве имени текущего проекта (меню File|Project|Set Project To Current File).

Откроем окно приложения Compiler (команда MAX+plusII|Compiler) и выберем в меню Processing пункт Functional SNF Extractor, установим опцию компилятора Total Recompile. Выполним трансляцию проекта.

Откроем редактор WaveForm Editor и создадим входные векторы. Для этого выберем пункт меню Node|Enter Nodes from SNF и в диалоговом окне выберем из списка входов и выходов следующие группы линий: OPCODE[3..0], DATA[7..0], WMSB[3..0], WLSB[3..0] и CLK. Выберем интервал моделирования и шаг тактовой сетки в соответствии с количеством команд, которые будут проверяться. В соответствии с этим зададим параметр End Time (меню File) и Grid Size (меню Options). После этого зададим различные комбинации кодов операции и данных (содержимое рабочего регистра W на начало моделирования не определено, поэтому разумно первой командой загрузить в него начальное значение). На каждую операцию необходимо установить фронт тактового сигнала CLK. Откроем приложение Simulator и запустим процесс на выполнение. Выполнив симуляцию, сравним полученные значения с предполагаемыми. В случае, если значения не совпадают, необходимо исправить текст программы и повторить процедуру сначала.

Реализация устройства на плате лабораторного макета. Для реализации и тестирования разработанного АЛУ используем плату LabKit-8000. Для этого нужен файл lab5.tdf, осуществляющий интерфейс исследуемого устройства с аппаратными ресурсами платы:

```
TITLE "LabKit 8000. PIC16 ALU.";

INCLUDE "ALUBLOCK.INC";
INCLUDE "DECODE7.INC";
```

```
SUBDESIGN Lab5
(
    Osc             :input; -- вход тактовой часоты
    Key[3..1]       :input;
    Sw[8..1]        :input;

    Led[8..1]       :output;
    Beep            :output;

    a[3..1]         :output;
    b[3 .1]         :output;
    c[3..1]         :output;
    d[3..1]         :output;
    e[3..1]         :output;
    f[3..1]         :output;
    g[3..1]         :output;
    h[3..1]         :output;
)

VARIABLE

    ARITHUNIT       :ALUBLOCK;
    Digit[3 .1]     :DECODE7;

BEGIN

 ARITHUNIT.OPCODE[] = !Sw[1..4];
 ARITHUNIT.DATA[] = (B"0000",!Sw[5..8]);

    ARITHUNIT.CLK = !Key[1]; -- нарастающий фронт

 Digit[1].DI[] = ARITHUNIT.WMSB[];
 Digit[2].DI[] = ARITHUNIT.WLSB[];
 if (Key[2]==B"1") then -- data (literal)
  Digit[3].DI[3..0] = !Sw[5..8];
 else
  Digit[3].DI[3..0] = !Sw[1..4];
 end if;

(a[3..1],b[3..1],c[3..1],d[3..1],e[3..1],f[3..1],g[3..1],h[3..1]) = (Digit[3..1].a,Digit[3..1].b,Digit[3..1].c,Di
git[3..1].d,Digit[3..1].e,Digit[3..1].f,Digit[3..1].g,Digit[3..1].h);

 Led[1..] = (B"11111",!ARITHUNIT.FLAGS[]);

 Beep = !Key1;

END;
```

После заголовка следуют директивы INCLUDE, присоединяющие к данному тексту файлы ALUBLOCK и DECODE7, из которых первый уже рассмотрен в данной работе, а второй реализует дешифратор двоичного кода в коды управления 7-сегментного индикатора и имеет следующий вид:

```
TITLE     "LabKit 8000. Decoder7";

SUBDESIGN decode7
(
    DI[3..0]     : INPUT;
    a,b,c,d,e,f,g            : OUTPUT;
)

BEGIN
```

```
TABLE
        DI[ ]       =>  (a, b, c, d, e, f, g);
        B"0000"           =>; B"1111110"; -- H"0"
        B"0001"           =>; B"0110000"; -- H"1"
        B"0010"           =>; B"1101101"; -- H"2"
        B"0011"           =>; B"1111001"; -- H"3"
        B"0100"           =>; B"0110011"; -- H"4"
        B"0101"           =>; B"1011011"; -- H"5"
        B"0110"           =>; B"1011111"; -- H"6"
        B"0111"           =>; B"1110000"; -- H"7"
        B"1000"           =>; B"1111111"; -- H"8"
        B"1001"           =>; B"1111011"; -- H"9"
        B"1010"           =>; B"1110111"; -- H"A"
        B"1011"           =>; B"0011111"; -- H"B"
        B"1100"           =>; B"1001110"; -- H"C"
        B"1101"           =>; B"0111101"; -- H"D"
        B"1110"           =>; B"1001111"; -- H"E"
        B"1111"           =>; B"1000111"; -- H"F"

    END TABLE;
END;
```

В секции SUBDESIGN файла lab5.tdf в качестве входных описаны порты устройства, присоединенные в кнопкам и переключателям, а также вход синхронизации. В качестве выходных описаны порты, присоединенные к светодиодам, пьезодинамику и семисегментным индикаторам.

В секции описания переменных имя ARITHUNIT присвоено устройству типа ALUBLOCK, а имя DIGIT – группе устройств типа DECODE7.

В секции логики на входы кода операции разработанного устройства подключаются линии с портов переключателей 1..4. На входах данных в старшей тетраде фиксируются нули, а на младшую подключаются линии с портов переключателей 5..8. На стробирующий вход подключается линия с порта клавиши 1. Далее текущее содержимое рабочего регистра W подается на входы дешифраторов семисегментных индикаторов 1 и 2. На дешифратор индикатора 3 подается младшая тетрада константы или код операции, в зависимости от того, нажата кнопка 2 или нет. В завершение на выходные порты интерфейсного модуля подключаются выходные линии дешифраторов, на порты светодиодов – линии с регистра флагов устройства, на порт управления звуковым сигналом – линия с порта кнопки 1.

Следует обратить внимание на то, что некоторые входные и выходные сигналы предварительно инвертируются и задаются в обратном порядке, что связано со значениями активных уровней сигналов кнопок, переключателей и индикаторов на плате, а также их расположением (Приложение 3). Видно, что тактовый сигнал выводится на кнопку Key1, код команды задается переключателями Sw, причем порядок следования битов обратный, что соответствует общепринятому положению младшего бита справа. Код операции задается старшей тетрадой, а литерал младшей (что соответствует младшим четырем битам литерала, старшие всегда устанавливаются в «0»). Флаги регистра STATUS отображаются на крайних правых свето-

диодах линейки (Z,DC,C соответственно, что соответствует битам STATUS[2..0]).

Для выполнение работы необходимо установить в качестве имени проекта lab5. Для этого выберем в меню File|Progect|Name и в диалоговом окне выберем имя lab5. Осуществим трансляцию проекта с помощью приложения Compiler, предварительно установив опцию Total Recompile в меню Processing.

Подключим питание к плате макета, откроем приложение Programmer и загрузим код конфигурации в ПЛИС.

Тестирование устройства. Проверка функционирования устройства осуществляется следующим образом:

1. с помощью тумблеров необходимо задать входное слово команды в соответствии с таблицей;
2. индикаторы 1 и 2 отображают текущее содержимое рабочего регистра, а младшие три бита регистра STATUS отображаются на линейке светодиодов;
3. нажатие кнопки 1 инициирует выполнение команды;
4. индикаторы отображают новое содержимое рабочего регистра W – результат выполнения арифметической операции, линейка светодиодов отображает новые значения флагов операции.

5.6. Язык AlteraHDL

После первоначального знакомства с процессом создания и обработки проекта в системе MAX+plusII рассмотрим более подробно конструкции языка AHDL. Формальное описание языка с использованием форм Бэкуса-Наура приведено в фирменной документации и доступно при помощи команды Help системы MAX+plusII.

5.6.1. Структура программы на языке AHDL

Программа на языке AHDL имеет определенную структуру. Разделы программы должны следовать в следующем порядке:

Title Statement
Include Statement
Constant Statement
Define Statement
Parameters Statement
Function Prototype Statement
Options Statement
Assert Statement

Subdesigh Section
Variable Section
- Instance Declaration
- Node Declaration
- Register Declaration
- State Machine Declaration
- Machine Alias Declaration
- If Generate Statement

Logic Section
- Defaults Statement
- Assert Statement
- Boolean Equations
- Boolean Control Equations
- Case Statement
- For Generata Statement
- If Then Statement
- If Generata Statement
- In-Line Logic Function Reference
- Truth Table Statement

Из всех перечисленных разделов являются обязательными только Subdesign Section и Logic Section.

Title Statement. Раздел заголовка позволяет дать название модулю. С этим именем будет сформирован файла отчета (Report File). В файле этот раздел может присутствовать только один раз.

Правила синтаксиса:

- Раздел начинается с ключевого слова *TITLE*, за которым следует текстовая строка, заключенная в двойные кавычки (“). В конце заголовка ставится символ (;).
- Текстовая строка может содержать до 255 символов включительно. Она не должна содержать символов конца строки или конца страницы.
- Для включения в текстовую строку двойных кавычек следует использовать дополнительную пару двойных кавычек: “ “ “ *FLEX 8000* “ “ Programmable Timer”.

Include Statement. Этот оператор позволяет включить содержимое указанного файла в текст данной программы. В файле этот оператор может использоваться неограниченное число раз.

Правила синтаксиса:

- Оператор начинается с ключевого слова INCLUDE, за которым в двойных кавычках (“) указывается имя файла включения (Include file). Далее ставится символ (;).

- Если явно не задано расширение файла включения, то компилятор ищет файл, имеющий заданное имя и расширение .INC.
- Имя файла, указанное в операторе включения, не должно содержать пути к файлу.

Constant Statement. Оператор определения константы позволяет присвоить имени константы значение числа, либо результата вычисления арифметического выражения. В файле этот оператор может использоваться неограниченное число раз.

Правила синтаксиса:

- Оператор начинается с ключевого слова CONSTANT, за которым следует символическое имя, символ (=), число или арифметическое выражение. Далее ставится точка с запятой (;).
- Имя константы должно быть уникальным и не должно содержать пробелов.
- Ссылка на константу допускается только после ее определения.
- При определении константы могут использоваться заданные ранее константы.
- Циклическое определение констант недопустимо.

Арифметическое выражение, определяющее константу, вычисляется компилятором и заменяется числом на этапе анализа синтаксиса. Применение арифметических выражений при определении констант не приводит к использованию логических ресурсов PLD.

Define Statement. Оператор обозначения позволяет присвоить арифметическим выражениям символические имена. В файле этот оператор может использоваться неограниченное число раз.

Правила синтаксиса:

- Оператор начинается с ключевого слова DEFINE, за которым следует символическое имя арифметического выражения со списком аргументов, заключенным в скобки. За списком аргументов ставится знак (=), далее следует арифметическое выражение и символ (;).
- Список аргументов может содержать один или более аргументов. Аргументы в списке отделяются друг от друга запятой.
- Символическое имя должно быть уникальным и не должно содержать пробелов.
- Ссылка на символическое имя арифметического выражения допустима только после его описания в разделе DEFINE.
- Обозначаемое арифметическое выражение может содержать символическое имя обозначенного ранее арифметического выражения.

Parameters Statement. Оператор объявления параметров позволяет объявить параметры, управляющие реализацией параметризированных модулей. В файле этот оператор может использоваться неограниченное число раз.

Примеры:

```
1. PARAMETERS
   (FILENAME="myfile.mif",
    WIDTH,
    AD_WIDTH,
    NUMWORDS=2^AD_WIDTH);
2. PARAMETERS (DEVICE_FAMILY);
```

DEVICE_FAMILY предопределенный параметр языка AHDL. Его значение может быть задано в окне Device (меню Assign).

Правила синтаксиса:

- Оператор начинается с ключевого слова PARAMETERS, за которым следует заключенный в скобки список параметров. После закрывающей скобки ставится символ (;).
- Список параметров может содержать один или несколько параметров.
- Параметры в списке отделяются друг от друга запятой (,).
- Значение параметра по умолчанию (defaults) указывается после символа (=). Задание этого значения необязательно.
- Параметр может задаваться через другой параметр.
- Циклическое задание параметров недопустимо.
- Заданное пользователем имя параметра должно быть уникальным и не содержать пробелов.
- Областью действия параметра является текст, в котором он был объявлен.

Function Prototype Statement. Оператор описания прототипа позволяет описать параметры модулей (примитивов), которые будут использованы в тексте. В файле этот оператор может использоваться неограниченное число раз.

Примеры:

```
1. FUNCTION  lpm_add_sub
     (Cin,  dataa[LPM_WIDTH-1 . . 0], datab[LPM_WIDTH-1 . . 0], add_sub) --входы модуля
      WITH (LPM_WIDTH, LPM_REPRESENTATION,
   LPM_DIRECTION, ADDERTYPE,
   ONE_INPUT_IS_CONSTANT) -- передаваемые параметры
   RETURNS (result[LPM_WIDTH-1 . . 0], Cout, overflow); -- выходы модуля

2. FUNCTION compare (a[3..0], b[3..0])
     RETURNS (less, equal, greater);
```

Правила синтаксиса:

- Оператор начинается с ключевого слова FUNCTION, за которым следует имя модуля, далее заключенный в скобки список входов модуля. При описании прототипа параметризированного модуля далее следует ключевое слово WITH и заключенный в скобки список передаваемых в модуль параметров. Затем указывается ключевое слово RETURNS, далее заключенный в скобки список выходов модуля. Оператор оканчивается символом (;).
- Элементы списков отделяются друг от друга запятыми.

Options Statement. Этот оператор позволяет определить в группе элемент с меньшим индексом (BITO) как:

- LSB – младший разряд. При этом индексы в группах должны указываться в убывающей слева направо последовательности.
- MSB – старший разряд. При этом индексы в группах должны указываться в возрастающей слева направо последовательности.
- ANY. При этом порядок перечисления индексов в группе произвольный.

При нарушении порядка перечисления индексов в группах появляется предупреждение компилятора.

Пример:

```
OPTIONS  BITO = MSB;
```

Правила синтаксиса:

- Оператор задания опции начинается с ключевого слова OPTIONS, за которым следует ключевое слово BITO, символ «равно» (=) и одно из ключевых слов LSB, MSB, ANY. Далее ставится точка с запятой (;).
- Если опция не задана, то по умолчанию предполагается, что BITO = LSB.
- Значение опции, заданное в файле верхнего уровня иерархии описаний проекта, распространяется и на файлы с описанием модулей нижних уровней иерархии, если в них явно не задано другое значение опции.
- В файле этот оператор может использоваться только один раз.

Assert Statement. Оператор контроля позволяет проверить истинность арифметического выражения, а для случая, когда оно ложно, указать текст сообщения и определить реакцию компилятора. В файле этот оператор может использоваться неограниченное число раз.

Пример:

```
ASSERT (WIDTH>0)
REPORT "Width (%) must be apositive integer" WIDTH
SEVERITY ERROR;
```

Правила синтаксиса:

- Оператор начинается с ключевого слова ASSERT, за которым следует арифметическое выражение. Если значение выражения False, то контролируемое условие считается невыполненным, и приложение Message Processor отображает заключенное в двойные кавычки сообщение, следующее за ключевым словом REPORT.
- Сообщение может содержать символ процент (%), который заменяется значением переменной, указываемой после закрывающих двойных кавычек. Если используется несколько символов процента (%), то переменные перечисляются через запятую в том порядке, в котором должны подставляться их значения.
- Необязательное ключевое слово SEVERITY позволяет задать реакцию компилятора: ERROR – ошибка, WARNING – предупреждение, INFO – информация. При отсутствии ключевого слова SEVERITY сообщение, по умолчанию, имеет значение ERROR.
- Если ключевое слово REPORT и соответствующее сообщение не были указаны в операторе, то при невыполненных условиях контроля процессор сообщений отображает следующую строчку:

  <ERROR, WARNING, INFO> : Line <номер строки>, File <имя файла> Assertion failed

- Оператор контроля оканчивается символом (;).
- Допустимо применение оператора в разделе описания логики (Logic Section).

Subdesign Section. Раздел описания модуля позволяет задать имя модуля и перечислить его выводы. Имя модуля должно совпадать с именем его текстового файла. Если модуль является модулем верхнего уровня иерархии, то его имя является одновременно и именем проекта, поэтому это имя будут иметь все вспомогательные файлы. В файле этот раздел может присутствовать только один раз.

Правила синтаксиса:

- Раздел начинается с ключевого слова SUBDESIGN, за которым следует имя модуля. Максимальная длина имени 32 символа.
- После имени модуля следует заключенный в круглые скобки список его выводов.
- Типы выводов: INPUT – вход, OUTPUT – выход, BIDIR – двунаправленный вывод, MACHINE INPUT – вход импортируемых состояний автомата, MACHINE OUTPUT – выход экспортируемых состояний автомата. Выводы MACHINE INPUT и MACHINE OUTPUT не могут быть использованы в описании модуля верхнего уровня проекта, т.е. модуля, выводы которого являются выводами микросхемы.

- Выводы перечисляются через запятую в одну или несколько строчек. В конце перечня однотипных выводов символ (:), затем ключевое слово, указывающее тип выводов, и далее символ (;).
- После ключевого слова INPUT может быть указано значение по умолчанию (GND, VCC) входного сигнала. Это значение подается на вход в случае, если он окажется неподключенным при использовании данного модуля в иерархическом проекте.

Variable Section. Раздел переменных позволяет задать внутренние переменные модуля, используемые в разделе описания логики (Logic Section). В файле этот раздел может присутствовать только один раз.

Переменные в языке AHDL представляют собой имена цепей; цепей с тремя состояниями; модулей, используемых в описании.

Правила синтаксиса:

- Раздел начинается с ключевого слова VARIABLE. Далее следуют: символическое имя переменной, символ (:), тип переменной, символ (;). Имена однотипных переменных могут быть перечислены через запятую.
- Допустимые типы переменных: NODE – цепь; TRI_STATE_NODE – цепь с тремя состояниями; модуль более низкого уровня иерархии; примитив; конечный автомат; псевдоним конечного автомата.
- Раздел переменных может содержать оператор IF GENERATE, позволяющий, в зависимости от значения оцениваемого в операторе арифметического выражения, управлять заданием переменных.

Logic Section. Раздел описания логики позволяет дать функциональное описание проектируемого модуля. В файле этот раздел должен быть единственным.

Правила синтаксиса:

- Раздел начинается с ключевого слова BEGIN и заканчивается ключевым словом END, за которым следует точка с запятой (;).
- В разделе могут быть использованы:
  - Defaults Statement (Оператор задания исходных значений);
  - Boolean Equations (булевы уравнения);
  - Boolean Control Equations (булевы уравнения для описания функционирования);
  - Case Statement (Оператор CASE);
  - If Then Statement (Оператор If Then);
  - If Generate Statement (Оператор IF Generate);
  - In-line Logic Function reference (непосредственное обращение к функциям);
  - For Generate Statement (Оператор For Generate);
  - Truth Table Statement (Таблица истинности);
  - Assert Statement (Оператор контроля).

Если в разделе логики используется оператор задания значений по умолчанию (Defaults Statement), то он должен находиться сразу после ключевого слова BEGIN. Другие операторы могут следовать в произвольном порядке.

5.6.2. Числа

Язык AHDL поддерживает все комбинации десятичных, двоичных, восьмеричных и шестнадцатеричных чисел.

Основание	Формат
Десятичное	<последовательность цифр 0 .. 9>
Двоичное	B<последовательность цифр 0, 1,X>
Восьмеричное	O<последовательность цифр 0 .. 7> или Q<последовательность цифр 0 .. 7>
Шестнадцатеричное	X<последовательность цифр 0 .. 9, A .. F> или H<последовательность цифр 0 .. 9, A .. F>

Пример:

```
SUBDESIGN decode1
    (
    address[15..0] : INPUT;
    chip_enable : OUTPUT;
    )
BEGIN
    chip_enable = (address[15..0] == H»0370»)
END;
```

Это дешифратор адреса, формирующий высокий активный уровень сигнала разрешения доступа к магистрали, если адрес на входе равен шестнадцатеричному числу 370h. В этом примере десятичные числа использованы для указания размерности массива разрядов шины адреса. Шестнадцатеричное число H«0370» – это значение адреса, при котором формируется активный разрешающий сигнал.

5.6.3. Имена

Имена могут присваиваться следующим элементам:

- портам, цепям и шинам;
- константам;
- переменным конечных автоматов, битам состояний и именам состояний;
- экземплярам;
- параметрам;
- оценочным функциям;

- именованным операторам;
- сегментам памяти;
- подпроектам.

Синтаксис имен в языке AHDL подчиняется общепринятым правилам многих других языков:

- Имя может содержать буквы, цифры и символ (_). Прописные и строчные буквы не различаются. Имя не должно быть зарезервированным словом.
- Имя должно начинаться с буквы и не должно заканчиваться символом (_).
- Строковые имена заключаются в одиночные кавычки (').

Имена цепей могут содержать до 32 символов. Если имя цепи не присвоено, она получает имя по умолчанию.

В следующих таблицах приведены зарезервированные слова (ключевые слова и служебные имена), а также символы языка AHDL.

Ключевые слова

AND	FUNCTION	OUTPUT
ASSERT	GENERATE	PARAMETERS
BEGIN	GND	REPORT
BIDIR	HELP_ID	RETURNS
BITS	IF	SEGMENTS
BURIED	INCLUDE	SEVERITY
CASE	INPUT	STATES
CLIQUE	IS	SUBDESIGN
CONNECTED_PINS	LOG2	TABLE
CONSTANT	MACHINE	THEN
DEFAULTS	MOD	TITLE
DEFINE	NAND	TO
DESIGN	NODE	TRI_STATE_NODE
DEVICE	NOR	VARIABLE
DIV	NOT	VCC
ELSE	OF	WHEN
ELSIF	OPTIONS	WITH
END	OR	XNOR
FOR	OTHERS	XOR

Служебные имена

CARRY	JKFFE	SRFFE
CASCADE	JKFF	SRFF
CEIL	LATCH	TFFE
DFFE	LCELL	TFF
DFF	MCELL	TRI
EXP	MEMORY	USED
FLOOR	OPENDRN	WIRE
GLOBAL	SOFT	X

Символы языка AHDL

Символ	Описание
_	Подчеркивание (Underscore)
-	Тире (Dash)
/	Косая черта (Slash)
--	Двойное тире
%	Процент (Percent)
( )	Открывающая и закрывающая скобки (Left and Right Parentheses)
[ ]	Открывающая и закрывающая прямоугольные скобки (Left and Right Brackets)
'...'	Открывающая и закрывающая кавычки (Quotation marks)
"..."	Двойные кавычки (Double Quotation)
.	Точка (Period)
..	Двойная точка
;	Точка с запятой (Semicolon)
,	Запятая (Comma)
:	Двоеточие (Colon)
=	Равенство (Equals)
=>	Стрелка (Arrow)
+	Плюс (Plus)
-	Минус (Minus)
==	Двойное равенство
!	Восклицательный знак (Exclamation point)
!=	Восклицательный знак и равенство
>	Больше (Greater than)
>=	Больше или равно
<	Меньше (Less than)
<=	Меньше или равно
&	Амперсанд (Ampersand)
!&	Восклицательный знак и амперсанд
$	Доллар (Dollar sign)
!$	Восклицательный знак и доллар
#	Фунт (Pound sign)
!#	Восклицательный знак и фунт
?	Вопросительный знак (Question mark)

5.6.4. Константы

Константы в AHDL можно использовать, давая имя числу или текстовой строке. Константы определяются с помощью оператора *Constant*. В нем константу можно определить с помощью арифметического выражения. Оператора *Constant* имеет следующие правила синтаксиса и семантики:

- Оператор начинается с ключевого слова CONSTANT, за которым следует символическое имя константы, затем символ '=', затем число или арифметическое выражение. Оператор завершается символом ';'.
- В арифметическое выражение оператора *Constant* могут входить имена констант, определенных ранее.
- Компилятор из выражения оператора *Constant* формирует число, логика для этого выражения не создается.
- Использование циклических ссылок на другие имена констант недопустимы.

- Использовать имя константы можно только после ее определения.

Примеры

```
CONSTANT UPPER_LIMIT = 130;
CONSTANT FOO = 1;
CONSTANT FOO_PLUS_ONE = FOO + 1;
```

5.6.5. Переменные

Переменные описываются в необязательном разделе *Variable* .

- Раздел начинается с ключевого слова VARIABLE , за которым следуют символические имена, разделенные запятыми, далее символ двоеточия ':' и тип переменной.
- Допустимыми типами переменных являются NODE, TRI_STATE_NODE, имена примитивов, мегафункций, макрофункций, машины состояний.
- Раздел завершается символом ';'.

Объявление переменной типа NODE удобно для хранения значений промежуточных выражений, ей можно присвоить значение по умолчанию.

5.6.6. Порты

Порт – это вход или выход логической функции. Порты могут находиться в двух местах:

- Порт, являющийся входом или выходом текущего файла, объявляется в разделе Subdesign. Формат объявления следующий:

<имя порта>:<тип порта> [= <значение по умолчанию>]

Имеются следующие типы портов:

INPUT	MACHINE INPUT
OUTPUT	MACHINE OUTPUT
BIDIR	

- Порт, являющийся входом или выходом файла более низкого уровня или примитива, используется в разделе Logic.

5.6.7. Цепи

Цепь, объявленная с помощью слова Node в разделе *Variable*, может использоваться для хранения значения промежуточного выражения. Такие объявления полезны при повторном использовании этого выражения. При этом выражение можно заменить именем цепи, которое легче читается.

Цепи бывают типа Node и Tri_State_Node. Эти типы отличаются тем, что Node связывают сигналы функцией Проводное-И (значения по умолчанию в операторах *Defaults* равно VCC) или Проводное-ИЛИ (значения по умолчанию в операторах *Defaults* равно GND), а Tri_State_Node привязывают сигналы к одной и той же цепи.

5.6.8. Группы, шины (группы цепей)

Такие однотипные элементы, как символические имена, порты, цепи можно объединять в группы. Имя группы можно определить именем с одним диапазоном, двумя диапазонами или именем в последовательном формате.

```
a[1..5]
d[1..3][1..6]
(f, g, h)
```

Шина (группа цепей) может иметь до 256 бит, трактуется как коллекция цепей и работает как одно целое.

В булевых уравнения шина может приравниваться булеву выражению, другой шине, одной цепи, VCC, GND, 1, 0. Оператор *Option* можно использовать для указания, будет ли младший бит наиболее значимым (MSB), наименее значимым (LSB) или каким-либо другим.

После определения шины скобки [] являются коротким определением всего диапазона. Равносильны a[4..1] и a[], b[5..4][3..2] равносильно b[][]. Выполняются следующие правила приравнивания:

- При приравнивании шин одинакового размера приравниваются соответствующие биты.
- Если шина приравнивается цепи, VCC или GND, все ее биты получают эти значения.
- Если шина приравнивается к 1, только наименее значимый бит получает значение 1, остальные соединяются с GND.

Если приравниваются шины не одинакового размера, левая шина должна быть кратна по размеру правой. Значение правой шины мультиплицируется на левую, они выравниваются по правому биту.

5.6.9. Примитивы

Примитивы – это библиотечные функциональные блоки, применяющиеся в системе Max+PlusII для описания ресурсов ПЛИС. Примитивы используются в графических (.gdf), текстовых (.tdf) и VHDL файлах (.vhd). Символы примитивов для графического редактора доступны в составе библиотеки \maxplus\max2lib\prim.

Примитивы триггеров и защелок:

DFF		DFFE
SRFF	SRFFE	
TFF		TFFE
JKFF	JKFFE	
LATCH		

5.6.10. Арифметические и логические выражения

В языке AHDL определены два типа выражений – арифметические и логические. Арифметические выражения – это средство формирования конструкций языка AHDL, эти выражения вычисляются компилятором на этапе синтаксического анализа, для их реализации ресурсы PLD не используются.

Логические выражения определяют собственно функционирование проектируемого устройства, правила их формирования и интерпретации связаны с особенностями элементов цифровых устройств.

Арифметические выражения используются для указания:

- выражений в разделе Define Statement;
- значений констант в разделе Constante Statement;
- границ диапазонов изменения индексов групп;
- границ диапазона переменной в операторе FOR GENERATE;
- выражения в операторах IF GENERATE и ASSERT.

Результат арифметического выражения должен быть целым положительным числом, если это не так, то он округляется до большего целого. Существуют две функции для явного задания правил округления: CETL – округление до большего целого; FLOOR - округление до меньшего целого. Арифметические и логические операции, операции сравнения (компараторы) имеют следующие приоритеты при вычислении арифметических выражений:

Операция/ Компаратор	Пример	Описание	Приоритет
+ (унарный)	+1	плюс	1
- (унарный)	-1	минус	1
! NOT	!a	отрицание	1
^	a ^ 2	степень	1
MOD	4 MOD 2	модуль	2
DIV	4 DIV 2	деление	2
*	a * 2	умножение	2
LOG2	LOG2(4-3)	логарифм по основанию 2	2
+	1+1	сложение	3
-	1-1	вычитание	3
==(numeric)	5 == 5	равенство чисел	4

Операция/ Компаратор	Пример	Описание	Приоритет
== (string)	"a" == "b"	равенство строк	4
!=	5 != 4	не равно	4
>	5 > 4	больше	4
>=	5 >= 5	больше или равно	4
<	a < b+2	Іменьше	4
<=	a <= b+2	меньше или равно	4
&	a & b	AND	5
AND	a AND b		
!&	1 !& 0	NAND	5
NAND	1 NAND 0		
$	1 $ 1	XOR	6
XOR	1 XOR 1		
!$	1 !$ 1	XNOR	6
XNOR	1 XNOR 1		
#	a # b	OR	7
OR	a OR b		
!#	a !# b	NOR	7
NOR	a NOR b		
?	(5<4) ? 3:4	Условная операция	8

Приоритеты операций могут изменяться при помощи скобок.

Булевы выражения и уравнения. Булевы выражения – это операнды (числа, цепи, группы), разделенные знаками арифметических и логических операций, компараторами (операторами сравнения), сгруппированные с помощью скобок. Эти выражения используются в булевых уравнениях и операторах CASE и IF_THEN.

Булево выражение может иметь один из следующих видов:

- операнд
  Пример: a, b[5..1], 7, VCC
- ссылка на логическую функцию
  Пример: out[15..0] = 16dmux(q[3..0]);
- Префиксный унарный оператор (! или -), примененный к булеву выражению
  Пример: !c
- Два булевых выражения, разделенных бинарным оператором
  Пример: d1 $ d3
- Булево выражение, заключенное в скобки
  Пример: (!foo & bar)

Результат булева выражения имеет ту же ширину, что и операнды.

Булево уравнение устанавливает цепь, шину, порт и т.п. в состояние, определяемое булевым выражением. Символ (=) в булевом уравнении указывает, что результат булева выражения справа является источником для цепи или шины слева. В булевом уравнении слева может находиться

идентификатор, имя порта или группы. Перед ним можно использовать оператор (!) инверсии. Справа в уравнении находится булево выражение, вычисляемое по правилам приоритетов (операции одного приоритета выполняются слева направо). Скобки могут изменять порядок вычислений.

Порядок следования булевых уравнений в программе не важен, логические состояния устанавливаются одновременно.

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

Здесь выходам out1 и out2 присваиваются значения, определяемые булевыми выражениями, в которых участвуют имена входных и выходных портов. Используются операции И, ИЛИ.

Это же устройство можно описать по другому, используя переменную типа NODE.

```
SUBDESIGN boole2
(
  a0, a1, b   : INPUT;
  out1, out2 : OUTPUT;
)
VARIABLE
  a_equals_2  : NODE;
BEGIN
  a_equals_2 = a1 & !a0;
  out1 =   a_equals_2
  out2 = a_equals_2 # b;
END;
```

Если имя переменной типа NODE используется в нескольких выражениях, то ее описание позволяет экономить ресурсы ПЛИС.

В булевых выражениях могут использоваться следующие логические операции:

Операция	Пример	Описание
! NOT	!music NOT music	инверсия
& AND	a & b a AND b	И
!& NAND	a[3..1] !& b[5..3] a[3..1] NAND b[5..3]	И-НЕ
$ XOR	fluo $ graf fluo XOR graf	исключающее ИЛИ
!$ XNOR	x2 !$ x4 x2 XNOR x4	инверсия исключающего ИЛИ

Операция	Пример	Описание
# OR	tris # tran tris OR tran	ИЛИ
!# NOR	c[8..5] !# d[5..4] c[8..5] NOR d[5..4]	ИЛИ-НЕ

Унарная операция NOT может применяться к одноразрядной переменной, группе переменных и к числу. При одноразрядной переменной результатом является инвертированное значение. В случае группы инвертируется каждый член группы. В случае числа инвертируется каждый разряд его двоичного представления.

Бинарные операции AND, NAND, OR, NOR, XOR, NXOR допускают следующие комбинации операндов:

- оба операнда одноразрядные (переменные, порты, VCC, GND);
- оба операнда являются группами – операция применяется поразрядно, поэтому операнды должны быть одинаковой разрядности;
- один оператор одноразрядный, а другой является группой – одноразрядный операнд тиражируется до группы, затем к двум группам поразрядно применяется операция;
- оба операнда числа – они представляются в двоичном формате и к группам двоичных разрядов применяется операция;
- один операнд число, а другой является одноразрядным, либо группой – число представляется группой двоичных разрядов и операция применяется к двум группам двоичных разрядов. Одноразрядный операнд тиражируется в группу, разрядность которой соответствует разрядности двоичного представления числа.

В булевых выражениях могут применяться следующие компараторы (операции сравнения):

Компаратор	Пример	Описание
== (логическое)	bus[15..0] ==H"B800"	Равно
!= (логическое)	a1 != a3	Не равно
> (арифметическое)	c[] > d[]	Больше
>= (арифметическое)	fiu[] >= fiu[]	больше или равно
< (арифметическое)	c < d+2	Меньше
<= (арифметическое)	e <= f-2	меньше или равно

Результатом операции сравнения является логический ноль (GND), если условие не выполнено, и логическая единица (VCC), если условие выполнено.

Из таблицы видно, что компараторы делятся на логические и арифметические. При логическом сравнении осуществляется поразрядное сравнение операндов, а при арифметическом сравнении группа разрядов интерпретируется как двоичное число без знака.

В булевых выражениях могут применяться следующие арифметические операции:

Операция	Пример	Описание
+ (унарный)	+2	Плюс
- (унарный)	-f[]	Минус
+	dd[2..0] + ca[2..0]	Сложение
-	gamma[] - sigma[]	Вычитание

Арифметические операции могут выполняться над группами и числами. Если оба операнда являются группами, то они должны иметь одинаковую разрядность. При операциях над числами они представляются в виде групп двоичных разрядов и автоматически выравниваются.

Приоритет операций в булевых выражения следующий:

Приоритет	Операция/ компаратор:	Описание
1	-	минус, дополнение до 2 (negative)
1	!	инверсия, логическое НЕ (NOT)
2	+	сложение (addition)
2	-	вычитание (subtraction)
3	==	равно (equal to)
3	!=	не равно (not equal to)
3	<	меньше (less than)
3	<=	меньше или равно (less than or equal to)
3	>	больше (greater than)
3	>=	больше или равно (greater than or equal to)
4	&	И (AND)
4	!&	И-НЕ (NAND)
5	$	ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)
5	!$	ИНВЕРСИЯ ИСКЛЮЧАЮЩЕГО ИЛИ (XNOR)
6	#	ИЛИ (OR)
6	!#	ИЛИ-НЕ (NOR)

Пример сложного булева уравнения:

```
a[ ] = ((c[ ] &  -B"001101") + e[6..1]) # (p, q, r, s, t, v);
```

Это выражение вычисляется в следующем порядке:

1. Двоичное число B"001101" дополняется до 2 и становится B"110011". Унарный (-) имеет высший приоритет.
2. Выполняется операция AND над B"110011" и группой c[]. Эта операция имеет второй приоритет из-за скобок.
3. Результат складывается с группой e[6..1].
4. Выполняется операция OR над результатом и группой (p, q, r, s, t, v).

Общий результат присваивается группе a[ ]. Для правильности уравнения разрядность группы слева должна быть кратна разрядности группы справа.

5.6.11. Комбинационная логика

Комбинационная логика – это конечные автоматы без памяти. Эта логика описывается в AHDL с помощью булевых выражений и уравнений, таблиц истинности, мега и макрофункций.

Реализация табличной логики. Таблицы истинности в языке AHDL задаются с помощью оператора *Truth Table*. Оператор имеет следующие правила синтаксиса и семантики.

- Первая строка оператора до символа (;) является заголовком таблицы. Вначале идет ключевое слово TABLE, за которым следует список входов (имена, разделенные запятыми). Далее следует символ ⇒ и список выходов.
- Последующие строки таблицы содержат списки значений входов и выходов. Списки разделены символом ⇒. Значения могут быть числами, константами VCC и GND, именами констант, группами чисел или констант. Входные значения могут быть безразличными состояниями (X). Входные и выходные значения соответствуют входам и выходам, указанным в заголовке таблицы.
- Описание таблицы завершается ключевыми словами END TABLE и (;).

При использовании этого оператора необходимо соблюдать следующие правила.

- Имена в заголовке таблицы должны представлять собой либо одиночные цепи, либо группы.
- В таблице в качестве входных значений можно употреблять символ безразличного состояния (X) для задания диапазона кодов.
- Размеры списков значений входов и выходов должны точно соответствовать размеру списков имен входов и выходов в заголовке. В противном случае в качестве выходных используются значения по умолчанию.
- При использовании символа (X) необходимо следить, чтобы диапазоны задаваемых значений не перекрывались в пределах одной таблицы.

```
SUBDESIGN 7segment
(
  i[3..0]          : INPUT;
  a, b, c, d, e, f, g : OUTPUT;
)
```

```
BEGIN
  TABLE
    i[3..0] => a, b, c, d, e, f, g;
    H"0"   => 1, 1, 1, 1, 1, 1, 0;
    H"1"   => 0, 1, 1, 0, 0, 0, 0;
    H"2"   => 1, 1, 0, 1, 1, 0, 1;
    H"3"   => 1, 1, 1, 1, 0, 0, 1;
    H"4"   => 0, 1, 1, 0, 0, 1, 1;
    H"5"   => 1, 0, 1, 1, 0, 1, 1;
    H"6"   => 1, 0, 1, 1, 1, 1, 1;
    H"7"   => 1, 1, 1, 0, 0, 0, 0;
    H"8"   => 1, 1, 1, 1, 1, 1, 1;
    H"9"   => 1, 1, 1, 1, 0, 1, 1;
    H"A"   => 1, 1, 1, 0, 1, 1, 1;
    H"B"   => 0, 0, 1, 1, 1, 1, 1;
    H"C"   => 1, 0, 0, 1, 1, 1, 0;
    H"D"   => 0, 1, 1, 1, 1, 0, 1;
    H"E"   => 1, 0, 0, 1, 1, 1, 1;
    H"F"   => 1, 0, 0, 0, 1, 1, 1;
  END TABLE;
END;
```

Это дешифратор для семисегментного индикатора.

```
SUBDESIGN decode3
(
  addr[15..0], m/io       : INPUT;
  rom, ram, print, sp[2..1] : OUTPUT;
)
BEGIN
  TABLE
    m/io, addr[15..0]          =>          rom, ram,  print,    sp[];
    1,   B"00XXXXXXXXXXXXXX"  =>    1,   0,     0,      B"00";
    1,   B"100XXXXXXXXXXXXX"  =>    0,   1,     0,      B"00";
    0,   B"0000001010101110"   =>    0,   0,     1,      B"00";
    0,   B"0000001011011110"   =>    0,   0,     0,      B"01";
    0,   B"0000001101110000"   =>    0,   0,     0,      B"10";

  END TABLE;
END;
```

А это дешифратор адреса для внешней памяти микропроцессорной системы с шестнадцатиразрядным адресом. Использование безразличных состояний позволяет проще запрограммировать устройство и использовать меньше ресурсов ПЛИС.

Реализация условной логики. Условная логика реализуется с помощью операторов *IF Then* и *Case*.

Оператор *Case* определяет список альтернативных вариантов, один из которых выполняется, если значение селектора (переменной, группы или выражения), стоящего за ключевым словом CASE, соответствует значе-

нию, стоящему за ключевым словом WHEN этого варианта. Оператор имеет следующие правила синтаксиса и семантики.

- Вначале идет ключевое слово CASE, за которым следует селектор, далее ключевое слово IS.
- Оператор завершается ключевым словосочетанием END CASE, за которым стоит символ (;).
- Тело оператора представляет собой список альтернативных вариантов, каждый из которых начинается ключевым словом WHEN. Каждый вариант представляет собой набор констант, разделенных запятыми, за которым следует символ ⇒. За этим символом следует список операторов, разделенных символом символом (;). Последний вариант может начинаться ключевым словосочетанием WHEN OTHERS.
- Если значение селектора равно одному из значений констант варианта, выполняются все операторы этого варианта. Если значение селектора не равно ни одной из констант, выполняются операторы за словосочетанием WHEN OTHERS (если оно есть).
- Если оператор CASE используется для описания конечного автомата, словосочетание WHEN OTHERS не может использоваться для выхода из неразрешенных состояний.

```
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;
```

Это описание дешифратора 2→4, который преобразует двухразрядный двоичный код в код «one hot» (четыре значения содержат каждое по одной единице). В зависимости от кода на входе активизируется одна из ветвей оператора CASE.

Оператор *IF Then* определяет списки операторов, выполняемых в зависимости от значения булева выражения. Оператор имеет следующие правила синтаксиса и семантики.

- Вначале идет ключевое слово IF, за которым следует булево выражение, далее ключевое слово THEN и список операторов, разделенных символом (;).
- Далее между ключевыми словами ELSEIF и THEN может следовать дополнительное булево выражение, а за THEN – список операторов,

выполняемых или нет в зависимости от значения этого выражения. Эта необязательная конструкция может повторяться многократно.

- Операторы за словом THEN выполняются, если булево выражение истинно, при этом последующая конструкция ELSEIF THEN игнорируется.
- За конструкцией ELSEIF THEN может следовать ключевое слово ELSE, а за ним список операторов, которые выполняются, если ни одно из булевых выражений ни приняло истинного значения.
- Значения булевых выражений за ключевыми словами IF и ELSEIF вычисляются последовательно.
- Оператор завершается словосочетанием END IF и символом (;).

```
SUBDESIGN priority
(
  low, middle, high   : INPUT;
  highest_level[1..0] : OUTPUT;
)
BEGIN
  IF high THEN
    highest_level[] = 3;
  ELSIF middle THEN
    highest_level[] = 2;
  ELSIF low THEN
    highest_level[] = 1;
  ELSE
    highest_level[] = 0;
  END IF;
END;
```

Это шифратор приоритетов. На выходах устанавливается код, соответствующий приоритету входа, на котором имеется значение VCC.

А это просто пример выбора вариантов действий:

```
IF a[] == b[] THEN
    c[8..1] = H "77";
    addr[3..1] = f[3..1].q;
    f[].d = addr[] + 1;
ELSIF g3 $ g4  THEN
    f[].d = addr[];
ELSE
    d = VCC;
END IF;
```

5.6.12. Последовательностная логика

Последовательностная логика (конечные автоматы с памятью) описывается в языке AHDL с помощью триггеров, регистров, машин состояний и библиотеки параметрических модулей (LPM).

Регистры. Регистр создается объявлением его в разделе VARIABLE или ссылкой в строке в разделе LOGIC. После определения регистра его можно подключить к другой логике при помощи его портов. Обращение к портам регистра производится по имени регистра и имени порта:

```
<имя регистра>.<имя порта>
```

В следующем примере сформирован восьмиразрядный регистр из триггеров D-типа, который защелкивает данные с входов d по положительному фронту синхросигнала при положительном активном значении на входе разрешения.

```
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;
```

Первое уравнение логической секции соединяет вход тактового сигнала проекта с тактовыми входами триггеров. Во втором уравнении подключается разрешающий сигнал, в третьем уравнении подключаются входы, а в четвертом выходы. Altera рекомендует использовать в качестве тактирующего глобальный синхросигнал.

Можно объявить выходы проекта как D-триггеры в разделе VARIABLE. Это дает экономию за счет использования триггеров на выводах микросхемы.

```
SUBDESIGN reg_out
(
  clk, load, d[7..0] : INPUT;

  q[7..0]          : OUTPUT;
)

VARIABLE
  q[7..0]          : DFFE; % выходы объявлены как регистры %
BEGIN
  q[].clk = clk;
  q[].ena = load;
  q[] = d[];
END;
```

Счетчики. Счетчики обычно реализуются на D-триггерах с использованием оператора IF. В следующем примере реализован 16-разрядный загружаемый счетчик со сбросом.

```
SUBDESIGN ahdlcnt
(
  clk, load, ena, clr, d[15..0] : INPUT;
  q[15..0]                      : OUTPUT;
)

VARIABLE
  count[15..0]                  : DFF;

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;
```

5.6.13. Машина состояний

Машина состояний в AHDL – это языковая структура, позволяющая описывать конечный автомат в виде множества внутренних состояний проекта. Переходы между состояниями синхронизуются тактовым сигналом. Условие и направление перехода определяется для каждого состояния индивидуально. Каждому состоянию можно поставить в соответствие один или несколько выходных управляющих сигналов.

Машина состояний описывается как переменная в разделе VARIABLE. Поведение машины описывается в разделе логики после ключевого слова BEGIN. Форма описания машины состояний следующая:

```
имя_переменной : MACHINE [ OF BITS <список битов> ]
                                        WITH STATES ( состояние1, состояние2, ...);
где
    состояние ::= <имя> [ = <значение> ] ,
    < значение > ::=   <число> | <имя> .

Например.

ss : MACHINE  OF BITS (q1, q2, q3)
              WITH STATES    ( s1 = B"000",
                               s2 = B"010",
```

```
                                    s3 = B"111");
```

или

```
ss: MACHINE WITH STATES (s0, s1);
```

Обязательным является перечисление списка состояний (s0, s1). Если не объявлена конструкция OF BITS, то объявление имени состояния равносильно объявлению переменной типа NODE.

Если объявлена конструкция OF BITS, то перечисленные биты должны существовать физически. В этом случае состояния рассматриваются как комбинация значений выходов этих битов (логических ячеек), аналогично константам. Кроме того, количество возможных состояний в этом случае равно 2 в степени «количество битов», поэтому желательно явно описать все состояния, включая ложные. Первое по списку состояние является состоянием после сигнала сброса.

Машина состояний имеет следующие порты.

.clk – входной тактовый сигнал;

.reset – сигнал сброса, активный уровень – «1»;

.ena – разрешение перехода, активный уровень – «1».

Поведение машины можно описать двумя способами:

- с помощью конструкции CASE;
- с помощью конструкции TABLE.

```
SUBDESIGN StateMachine1                    -- Пример описания с
                                           -- помощью конструкции CASE
(
  clk, reset, d : INPUT;     -- входные порты проекта
  q         : OUTPUT;      -- выходной порт проекта
)
VARIABLE
  ss: MACHINE WITH STATES (s0, s1);  -- переменная ss – машина с с-ниями s0, s1
BEGIN
  ss.clk = clk;         -- соединим входной сигнал clk с  тактовым входом машины
 ss.reset = reset;    -- соединим входной сигнал reset  с  входом сброс
-- Далее рассматриваем каждое состояние и анализируем условия переходов

CASE ss IS
    WHEN s0 =>        -- в состоянии s0
      q = GND;        -- устанавливаем выход q в 0
      IF d THEN       -- если на входе d высокий уровень,
      ss = s1;     -- то следующее состояние будет s1, иначе ss останется прежним
      END IF;
    WHEN s1 =>       -- в состоянии s1
      q = VCC;        -- устанавливаем выход q в 1
IF !d THEN            -- если на входе d низкий уровень,
      ss = s0;        --  следующее состояние будет s0, иначе ss останется прежним
      END IF;
  END CASE;
END;
```

Теперь рассмотрим пример описания машины состояний с помощью конструкции TABLE. Это машина состояний *с асинхронными сигналами на выходах* (так называемый автомат Милли). Формирование выходного сигнала производит комбинационная логика. После перехода к следующему состоянию, синхронизированного тактовым сигналом clk, изменение выходного сигнала происходит с задержкой, которую вносит комбинационная логика.

```
SUBDESIGN mealy
(
  clk   : INPUT;          -- входные порты проекта
  reset : INPUT;
  y     : INPUT;
  z     : OUTPUT;         -- выходной порт проекта
)
VARIABLE
  ss: MACHINE WITH STATES (s0, s1, s2, s3);
BEGIN
  ss.clk = clk;
  ss.reset = reset;

  TABLE
 -- текущее   входной     выходной  следующее
 -- состояние  сигнал      сигнал    состояние
    ss,         y   =>     z,              ss; -- заголовок таблицы
    s0,         0   =>     0,              s0; --  описание данных
    s0,         1   =>     1,              s1;
    s1,         0   =>     1,              s1;
    s1,         1   =>     0,              s2;
    s2,         0   =>     0,              s2;
    s2,         1   =>     1,              s3;
    s3,         0   =>     0,              s3;
    s3,         1   =>     1,              s0;
  END TABLE;
END;
```

Далее приведен пример описания машины состояний *с синхронными сигналами на выходах*. Формирование выходного сигнала производится синхронно с переходом к следующему состоянию по стробу clk.

```
SUBDESIGN moore2
(
  clk   : INPUT;
  reset : INPUT;
  y     : INPUT;
  z     : OUTPUT;
)
VARIABLE
  ss: MACHINE WITH STATES (s0, s1, s2, s3);
  zd: NODE;
BEGIN
  ss.clk   = clk;
  ss.reset = reset;
  z = DFF(zd, clk, VCC, VCC);    -- выходной триггер,
```

```
                                        синхронизирующий изменение сигнала
  TABLE
-- текущее   входной        следующее    выходной
-- состояние  сигнал         состояние    сигнал
    ss,        y    =>        ss,                zd;
    s0,        0    =>        s0,                0;
    s0,        1    =>        s2,                1;
    s1,        0    =>        s0,                0;
    s1,        1    =>        s2,                1;
    s2,        0    =>        s2,                1;
    s2,        1    =>        s3,                0;
    s3,        0    =>        s3,                0;
    s3,        1    =>        s1,                1;
  END TABLE;
END;
```

5.7. Комплекс учебных средств «Проектирование систем на ПЛИС»

В лаборатории «Микропроцессорные системы» МИФИ на основе многолетнего опыта использования ПЛИС разработан комплекс средств для обучения проектированию цифровых устройств на ПЛИС Altera с использованием системы Max+plusII. Комплекс включает:

- Методические материалы в виде описания лабораторного практикума;
- Программное инструментальное обеспечение в виде системы MAX+plusII фирмы Altera. Пакет работает на персональных компьютерах в среде Windows;
- Аппаратное обеспечение в виде платы лабораторного макета и устройства типа ByteBlaster;

Комплекс может быть установлен на отдельный компьютер для индивидуального обучения (достаточно PC типа 486-DX4 с оперативной памятью от 16 Мбайт), а также на локальную сеть компьютеров учебного класса.

Лабораторный стенд LabKit-8000 (рис. 5.15) реализован на базе ПЛИС типа EPF8282A широко распространенного семейства FLEX 8000 фирмы Altera. Эта ПЛИС содержит около 5000 эквивалентных вентилей, имеет 68 линий ввода-вывода. В реальных проектах может быть использовано до 2500 вентилей и свыше 200 триггеров, что вполне достаточно как для выполнения учебных заданий практикума, так и для практической работы специалистов. Кроме ПЛИС макет содержит устройства ввода и отображения данных, генератор тактовых импульсов частотой 4 МГц, звуковой пьезоизлучатель, стабилизатор напряжения питания. Дополнительные разъемы (18 линий ввода-вывода) позволяют подключать к макету

различные модули расширения. Предусмотрен также разъем для тестирования реализуемых устройств в соответствии со стандартом JTAG.

Конфигурация ПЛИС осуществляется с помощью персонального компьютера через кабель типа ByteBlaster, подключаемый к соответствующему разъему макета. При выполнении практикума используется метод последовательной пассивной загрузки конфигурации от компьютера, на котором установлена система проектирования MAX+plusII. Технология внутрисистемного программирования ПЛИС позволяет реализовать на базе данного макета широкую номенклатуру комбинационных и последовательностных цифровых устройств различной сложности, проектирование и испытание которых производится в процессе выполнения лабораторных работ практикума. На макете предусмотрена также возможность подключения конфигурационного ПЗУ с последовательной выборкой для конфигурации ПЛИС без использования компьютера.

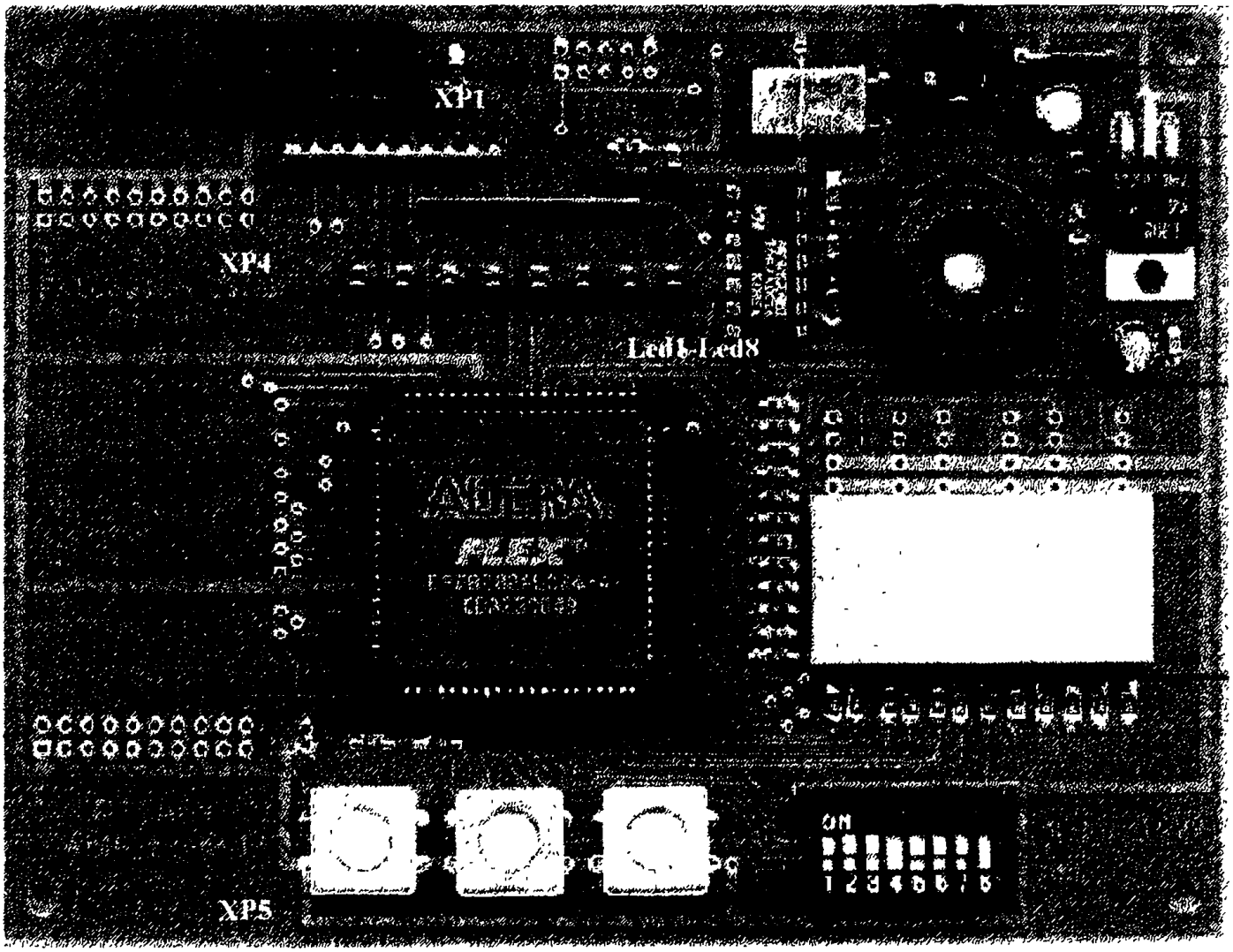


Рис. 5.15. Плата LabKit8000

Разработанный практикум включает семь лабораторных работ.

Работа 1. «Графический ввод схемы устройства и функциональная симуляция с использованием САПР MAX+plusII». В работе изучается графический редактор и осуществляется ввод в систему MAX+plusII

принципиальной схемы комбинационного устройства, описанного булевым уравнением. После этого выполняется трансляция проекта, с помощью редактора логико-временных диаграмм формируется последовательность входных тестовых векторов и осуществляется симуляция (логическое моделирование). В результате система MAX+plusII формирует диаграмму состояний для выходов устройства, анализируя которую совместно с таблицей истинности можно сделать заключение о правильности функционирования разработанного устройства.

Работа 2. «Описание схемы на языке AHDL, использование монитора иерархии проекта САПР MAX+plusII». В этой работе изучается процесс описания цифрового устройства на языке AHDL и создание иерархического проекта. С помощью редактора Text Editor создается текстовый файл описания схемы мультиплексера на языке AHDL. Для ускорения процесса используются шаблоны конструкций языка, заложенные в систему MAX+plusII. Алгоритм программы реализуется с помощью оператора IF_THEN_ELSE. На основе текстового файла создается символьный файл, который с помощью редактора Graphic Editor включается в иерархический проект вместе с устройством, созданным в первой работе. Для анализа иерархического проекта используется монитор Hierarchy Display. После этого выполняется трансляция проекта, формирование тестовых векторов и симуляция в последовательности, аналогичной предыдущей работе.

Работа 3. «Разработка комбинационных схем, программирование ПЛИС и анализ размещения схемы на кристалле». Работа посвящена проектированию комбинационных схем с использованием платы LabKit-8000, а также анализу и редактированию размещения схемы устройства на кристалле ПЛИС с помощью редактора конфигурации FloorPlan Editor системы MAX+plusII. Рассматривается функционирование схемы дешифратора для семисегментного индикатора и пример его описания на языке AHDL в виде таблицы. Перед трансляцией определяется тип микросхемы ПЛИС для реализации проекта и функции ее выводов. После трансляции с помощью приложения Programmer осуществляется загрузка кода программы через ByteBlaster в ПЛИС на плате лабораторного макета LabKit8000. Проверка работы дешифратора осуществляется вводом кода входных сигналов с переключателей и наблюдением знаков на семисегментном индикаторе. Для анализа размещения схемы устройства на кристалле используется редактор конфигурации FloorPlan Editor (рис. 5.16). С его помощью выполняется коррекция размещения выводов, после повторной трансляции работа дешифратора вновь проверяется с использованием платы LabKit-8000.

Работа 4. «Разработка последовательностных схем, временной анализ в системе MAX+plusII». В работе практически осваивается проектирование последовательностных схем на примере программируемого

счетчика/таймера - типового узла современных микроконтроллеров. Особое внимание уделяется выполнению анализа временных характеристик разработанной схемы с помощью приложения Timing Analyzer, который производится после процедуры размещения и трассировки схемы на кристалле ПЛИС. При этом исследуется влияние размещения элементов и выводов на кристалле на быстродействие проектируемого последовательностного устройства.

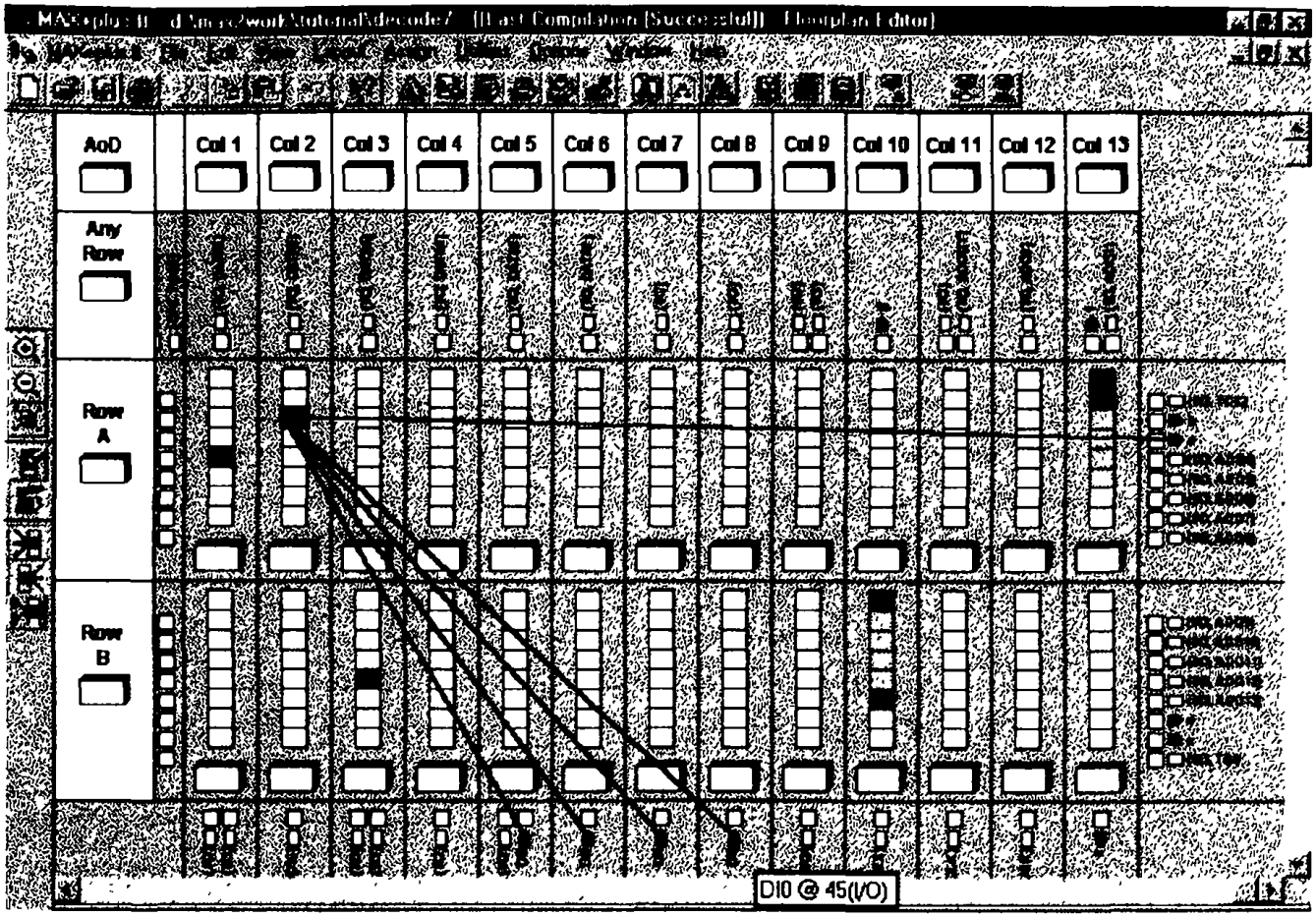


Рис. 5.16. Изображение конфигурации ПЛИС

Работа 5. «Разработка проекта цифрового устройства как машины состояний.» Рассматриваются способы описания проектируемого устройства как машины состояний в языке AHDL.

Работа 6. «Использование методов оптимизации при проектировании цифровых устройств на ПЛИС.» В работе изучаются методы и механизмы оптимизации, встроенные в систему MAX+plusII, а также приемы, позволяющие провести анализ характеристик разработанного проекта цифрового устройства.

Работа 7. «Построение конечных автоматов (на примере арифметико-логического устройства RISC-микроконтроллера)». В завершающей работе практикума, проектирование сложных конечных автоматов изучается на примере регистрового АЛУ – ядра микропроцессорных и микроконтроллерных устройств. В процессе выполнения работы проектируется схе-

ма АЛУ, реализующего набор основных арифметических и логических операций, проверяется правильность их выполнения.

При выполнении практикума особое внимание уделяется самостоятельной работе. Следует отметить, что система MAX+plusII предоставляет пользователям широкие возможности ее освоения с помощью опции Help (для ее использования требуется, естественно, знание английского языка). Методика проведения работ предполагает сначала выполнение обязательного задания, реализация которого подробно описана в разработанном учебном пособии. Затем самостоятельно выполняется разработка индивидуального проекта. Для каждой работы предлагается набор, содержащий 10-20 заданий на проектирование типовых цифровых узлов и блоков: дешифраторов, компараторов, сумматоров, умножителей, регистров, комбинационных и последовательностных схем. В устройствах, проектируемых счетчиков, генераторов кодов и других по этим заданиям, используются размещенные на макете средства ввода и вывода данных: переключатели, кнопки, светодиоды, индикаторы, генератор импульсов, пьезоизлучатель. Таким образом обеспечивается наглядность результатов при функционировании разработанной схемы.

Данный практикум используется как при обучении студентов соответствующих специальностей, так и для повышения квалификации специалистов промышленности.

При обучении студентов освоение методов реализации цифровых устройств на базе ПЛИС проводится вместе с изучением классических методов проектирования. Будущие специалисты осваивают основные способы преобразования и минимизации логических функций, учатся проектировать разнообразные устройства из набора серийно выпускаемых микросхем малой и средней степени интеграции или комплекта стандартных ячеек, используемых при разработке заказных БИС. Одновременно они учатся проектировать аналогичные устройства на базе ПЛИС, сравнивают технические и экономические характеристики устройств, реализуемых различными методами.

Для повышения квалификации специалистов промышленности, уже имеющих опыт разработки устройств на базе серийных микросхем, проводятся циклы практического освоения методики проектирования цифровых схем на базе ПЛИС. При этом теоретический раздел обучения сокращен, а основное внимание уделено практической реализации проектов на лабораторном макете. Достаточно подготовленные специалисты могут самостоятельно освоить методику проектирования цифровых устройств на базе ПЛИС Altera, используя разработанный лабораторный макет, учебное пособие и необходимое программное обеспечение.