Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра программного обеспечения информационных технологий

Проектирование аппаратно-программных вычислительных средств

Методическое пособие
для студентов специальности
«Программное обеспечение информационных технологий»
дневной и дистанционной форм обучения

Минск 2005

УДК 681.322(075.8)
ББК 32.973.26-02 я73
П 79

Авторы:
А.А. Иванюк, А.П. Занкович, Д.С. Петроненко, С.Б. Мусин

П 79 **Проектирование** аппаратно-программных вычислительных средств: Метод. пособие для студ. спец. «Программное обеспечение информационных технологий» дневной и дистанционной форм обуч. / А.А. Иванюк, А.П. Занкович, Д.С. Петроненко, С.Б. Мусин. – Мн.: БГУИР, 2005. – 59 с.:ил.
ISBN 985-444-754-5

В методическом пособии излагаются сведения, необходимые для проектирования цифровых систем на базе схем перепрограммируемой логики с использованием языка VHDL. Описан полный цикл проектирования цифровой системы с использованием Aldec Active-HDL, ее синтез с помощью Synplicity Synplify, а также размещение, трассировка и программирование кристалла с помощью Xilinx ISE. Приведен пример реализации обмена данными с хост-компьютером. Рассмотрен комплект макетных плат для проектирования и разработки программно-аппаратных решений на базе ПЛИС Xilinx Spartan 2E.

УДК 681.322 (075.8)
ББК 32.973.26-02 я73

ISBN 985-444-754-5

СОДЕРЖАНИЕ

1. АВТОМАТИЗИРОВАННОЕ ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ СИСТЕМ НА БАЗЕ СХЕМ ПЕРЕПРОГРАММИРУЕМОЙ ЛОГИКИ

1.1. Общие сведения о проектировании

В своем непрерывном развитии рынок микроэлектроники постоянно выдвигает все новые требования к проектируемым устройствам. Потребитель хочет получать быстродействующую, надежную и в то же время малогабаритную и потребляющую мало энергии продукцию. Два этих противоречивых требования усугубляются тем, что микроэлектронные изделия очень быстро стареют, время морального износа исчисляется иногда месяцами. Поэтому большое внимание уделяется сокращению времени выхода на рынок новых изделий.

Одним из способов решения этой проблемы стало использование программируемых логических интегральных схем (ПЛИС). Они состоят из конфигурируемых логических блоков, подобных переключателям с множеством входов и одним выходом (gates). В большинстве современных интегральных схем функции логических блоков фиксированы и не могут модифицироваться. Принципиальное отличие ПЛИС состоит в том, что и функции блоков, и конфигурация соединений между ними могут многократно меняться путем перепрограммирования. Различные типы ПЛИС отличаются количеством таких блоков, а также наличием встроенной памяти и фиксированного микропроцессорного ядра. С их помощью можно реализовывать цифровые устройства различной сложности, начиная от простейших фильтров, электронных ключей и потоковых шифраторов и заканчивая сложными многофункциональными системами на кристалле (System-on-Chip).

Еще одной из причин популярности ПЛИС в настоящее время является возможность использования при проектировании цифровых устройств высокоуровневых языков описания аппаратуры HDL (Hardware Description Language): VHDL, Verilog, SystemC, HandelC. Описание проекта в этом случае выполняется в наглядном текстовом виде, а процесс синтеза, размещения, трассировки и программирования ПЛИС − с помощью специализированных систем автоматизированного проектирования (САПР). Использование стандартизированных языков высокого уровня позволяет легко переносить описание проекта между САПР различных производителей и создавать технологически независимые описания цифровых устройств.

Проектирование интегральных схем для массового и специализированного использования сопряжено с большими временными и финансовыми затратами. Только организация производства таких систем требует от 20 000 до

100 000 дол. и нескольких недель или месяцев. Такие затраты окупаются только при выпуске изделий большими партиями. Для систем, которые производят небольшими сериями, оптимальным решением является использование ПЛИС. Высокая скорость перепрограммирования ПЛИС делает эти схемы незаменимыми помощниками проектировщиков для создания прототипов и отладки интегральных схем различной сложности.

Существует два типа проектирования цифровых устройств: восходящее и нисходящее. В первом случае проектировщик начинает описание с базовых элементов, которые используются для построения блоков более высокого уровня. Этот процесс продолжается до достижения самого верхнего, системного, уровня. Процесс нисходящего проектирования движется в обратном направлении. Сначала разрабатывается общая структура системы. При этом выделяются основные функциональные блоки и связи между ними. Затем прорабатывается структура каждого из них по отдельности путем составления из подблоков более низкого уровня. Этот процесс продолжается до тех пор, пока не будет достигнут такой уровень проектирования, на котором используются простейшие синтезируемые компоненты. Восходящее проектирование применяется, как правило, при наличии ограничений на использование определенной базы типовых элементов. Проектирование для ПЛИС в значительной степени лишено таких ограничений, поскольку отличается свободой реализации практически любых цифровых компонентов. Этим объясняется большая популярность использования нисходящего проектирования.

Полный цикл проектирования ПЛИС представлен на рис.1. Проектирование цифровой системы – многоуровневый, многошаговый и итерационный процесс, после каждого этапа возможен возврат на предыдущий или на самый первый этап с пересмотром ранее принятых решений. На любом этапе проектирования может быть выявлена ошибочность или неоптимальность выбранного ранее варианта реализации. Проектирование может считаться законченным только после верификации проекта в целом, когда завершена отладка готового устройства.

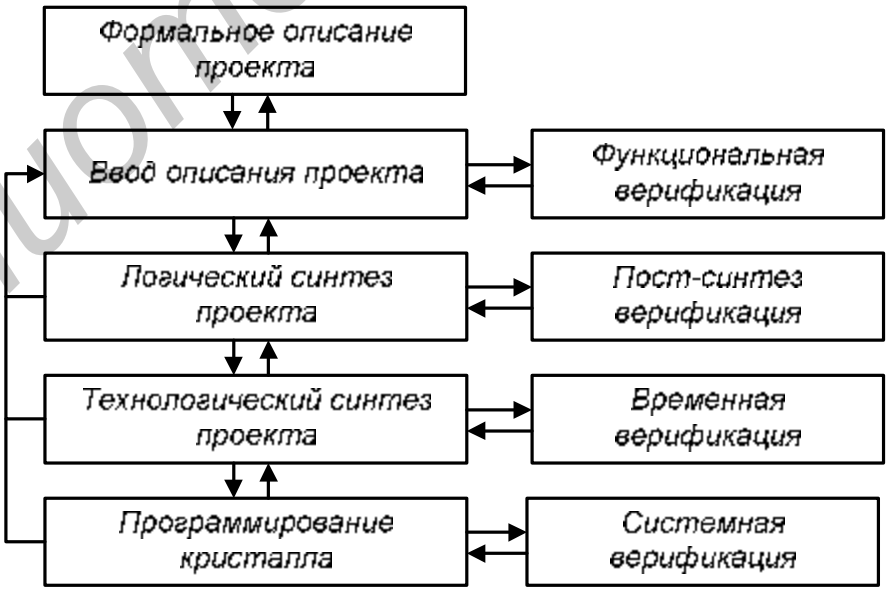


Рис. 1. Укрупненная структура полного цикла проектирования цифровой системы на базе ПЛИС

Использование при проектировании цифровых устройств программного обеспечения различных производителей сопряжено с определенными трудностями по переносу результатов работы различных САПР между друг другом. Для облегчения этой работы был создан целый ряд менеджер-проектов (Design Flow Manager), среди которых можно назвать входящие в состав таких интегрированных систем, как Altera Quartus, Xilinx ISE, Aldec Active-HDL. Внешний вид последнего представлен на рис. 2. Средствами САПР Active-HDL осуществляется ввод описания проекта, его функциональная, постсинтез и временная верификация, редактирование параметров логического и технологического синтеза. Для выполнения самого синтеза и программирования ПЛИС предоставляются интерфейсы вызова соответствующих программных средств большинства популярных производителей таких систем.

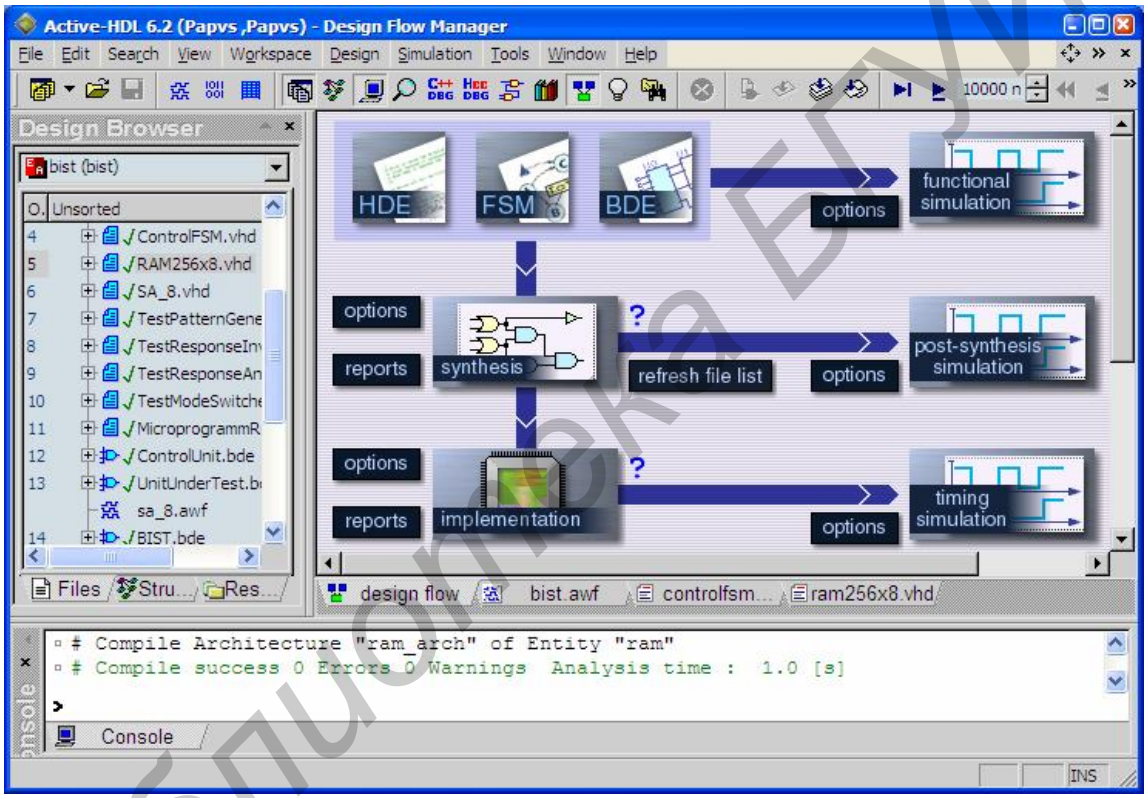


Рис. 2. Среда проектирования цифровых систем Active-HDL фирмы Aldec

Рассмотрим полный цикл проектирования цифровой системы с программным управлением на примере встроенного модуля неразрушающего самотестирования (ВСТ) для интегрированной памяти программируемой микросхемы Xilinx Spartan2E XC2S200E-200. Для моделирования описания проекта и управления всем процессом проектирования будем использовать среду Aldec Active-HDL версии 6.2. Синтез цифрового устройства выполним с помощью Synplicity Synplify 7.6 Pro, размещение, трассировку и программирование кристалла – с помощью Xilinx ISE 6.2i. Перечисленные программные средства обладают, по оценкам экспертов и по нашему субъективному мнению, наилучшими характеристиками среди ряда аналогов.

1.2. Формальное описание проекта

На этом этапе производится переход от технического задания к формализованному описанию устройства. Техническое задание, как правило, является смесью словесного и технического описаний требуемой функциональности устройства и способов взаимодействия с ним извне. Формализация технического задания приводит к выявлению основных блоков устройства и определению связей и взаимодействия между ними. Формально первый этап – этап декомпозиции задачи на отдельные подзадачи и разнесение их по различным блокам устройства. Способ и средства разбиения определяются функциональной завершенностью и обособленностью отдельных фрагментов. Так, на самом верхнем уровне иерархии система составляется из блоков самой общей структуры – устройства управления и операционных блоков. Затем, по мере выполнения декомпозиции, происходит переход от структурного описания к поведенческому.

Уровень детализации проекта зависит от его сложности. Чем более сложную функцию должно выполнять проектируемое устройство, тем сложнее оказывается его внутренняя организация и тем больше компонентов в нем можно выделить. Последовательное продвижение от блоков обобщенной функциональности к более мелким вспомогательным подблокам позволяет рассматривать компоненты как изолированно, так и во взаимосвязи друг с другом. С одной стороны, все подкомпоненты заключены в рамках и взаимодействуют внутри своего компонента. С другой стороны, они подключаются ко входам и выходам компонента, связанным со входами и выходами других компонентов.

В процессе декомпозиции системы происходит поэтапное выделение ее структуры. Разбиение на подблоки следует продолжать до тех пор, пока не будет достигнут поведенческий уровень, т.е. когда поведение блока станет очевидным и может быть легко реализовано с помощью синтезируемого подмножества языка описания аппаратуры.

Основными шагами при формальном описании цифровой системы с программным управлением являются:

1. Определение внешней организации устройства.
2. Выбор алгоритмов работы устройства.
3. Разработка системы команд.
4. Разработка общей структуры устройства.
5. Разработка архитектуры устройства управления и операционных блоков.

Внешняя организация устройства определяет входы и выходы проектируемой системы. Описанную таким образом систему можно рассматривать как «черный ящик», для которого определены функции внешних выводов, но не описаны способы их реализации. Процесс тестирования запускается после подачи высокого уровня сигнала на вход RUN (рис.3). Сигнал RST инициализирует модуль BCT, CLK – сигнал синхронизации. На

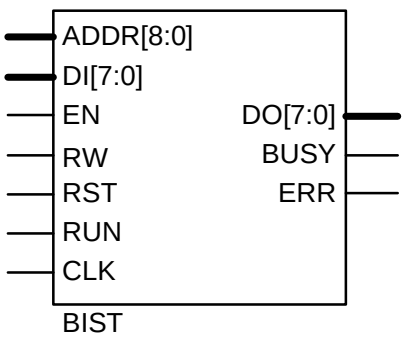


Рис. 3. Интерфейс системы неразрушающего тестирования памяти

выходе BUSY устанавливается высокий уровень сигнала в течение всего времени работы устройства, на выход ERR подается сигнал об ошибке в случае обнаружения в памяти неисправностей. Для доступа к схеме памяти в нормальном режиме используются следующие порты: ADDR – адрес памяти, DI – вход данных, DO – выход данных, RW – выбор операции чтения/записи, EN – разрешение работы с памятью. К этим портам может подключаться любое внешнее устройство и использовать память для хранения своих данных. Устройство самотестирования должно поддерживать различные разрядности шин адреса и данных.

Проектируемое устройство должно позволять выполнение периодического поиска неисправностей в модулях встроенной памяти систем на базе перепрограммируемой логики. Для поиска неисправностей необходимо использовать симметричные локально неразрушающие алгоритмы тестирования. Требуется обеспечить поддержку двух режимов работы – нормального и тестового. В нормальном режиме модуль самотестирования пропускает к схеме памяти все внешние входные сигналы, обеспечивая ее полнофункциональное использование другими компонентами системы. В тестовом режиме модуль самотестирования берет управление схемой памяти на себя, подает на ее вход тестовые воздействия и анализирует получаемые на выходах результаты. Переход в режим тестирования производится по сигналу от внешнего менеджера памяти, формируемому в моменты простоя устройства или длительного неиспользования схемы памяти.

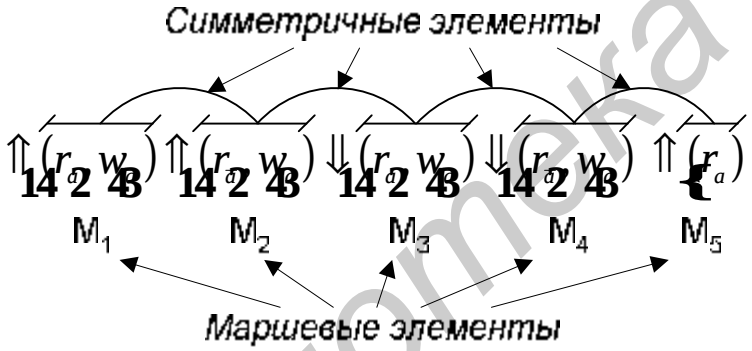


Рис. 4. Симметричный локально маршевый неразрушающий алгоритм SMarch C-

Проблема периодического тестирования, т.е. тестирования в промежутках между периодами нормального функционирования устройства, возникает при использовании интегральных схем со встроенными схемами памяти в составе аппаратуры для критических к показателям надежности приложений. Примерами таких систем могут служить телекоммуникационные устройства, маршрутизаторы, микропроцессоры систем управления и пр. При этом необходимо обеспечить сохранность информации, хранящейся в ОЗУ этих устройств. Для решения этой проблемы были предложены симметричные локально неразрушающие маршевые алгоритмы тестирования. Они характеризуются невысокой сложностью (порядка $O(N)$, где N – количество адресов ОЗУ), и высокой обнаруживающей способностью. В качестве примера такого алгоритма рассмотрим SMarch C- (рис. 4).

Алгоритм SMarch C- составляют пять маршевых элементов. Они включают набор из операций чтения или записи, применяемых ко всем ячейкам па-

мяти в определенном порядке – от старших адресов к младшим $(\Uparrow)$ или наоборот $(\Downarrow)$. Операции чтения считывают значения из памяти и помещают их в прямом (r_a) или проинвертированном $(r_{\bar{a}})$ виде во временный буфер, размер которого равен размеру слова памяти. Операции записи $(w_{\bar{a}})$ помещают проинвертированное значение из буфера в текущую ячейку памяти. Прочитанные значения дополнительно сжимаются на сигнатурных анализаторах. Нарушение симметрии прочитанных данных, обнаруживаемое с помощью симметричного сигнатурного анализа, позволяет определить наличие в памяти неисправностей.

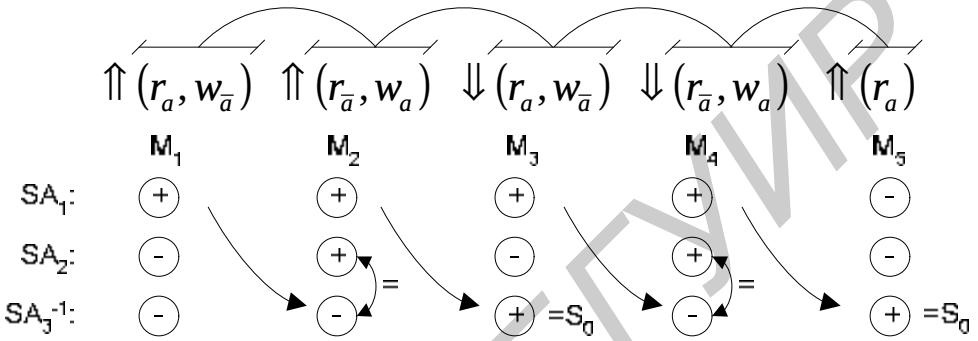


Рис. 5. Схема работы неразрушающего симметричного теста SMarch C- для проектируемой системы

В целях минимизации аппаратных затрат проектируемая ВСТ должна содержать не более трех сигнатурных анализаторов, используемых для сжатия и хранения результатов тестирования памяти. Для обеспечения возможности сжатия данных различных типов симметрии два из них (SA_1 и SA_2) задаются прямыми полиномами, а один (SA_3^{-1}) – обратным. На рис.5 представлено разложение симметричного локального маршевого теста SMarch C- для рассматриваемой ВСТ. Активные во время текущего маршевого элемента сигнатурные анализаторы отмечены знаком «+», неактивные – знаком «–». Таким образом, данные, считываемые первой операцией чтения M_1, сжимаются только сигнатурным анализатором SA_1, операцией M_2 – анализаторами SA_1 и SA_2, операцией M_3 – анализаторами SA_1 и SA_3^{-1} и т.д. Полученные сигнатуры могут передаваться между любыми двумя сигнатурными анализаторами, сравниваться между собой или с начальной нулевой сигнатурой. Так, сигнатура, полученная после первого маршевого алгоритма M_1, передается в сигнатурный анализатор SA_3^{-1}. После второго маршевого элемента M_2 сигнатуры из анализаторов SA_2 и SA_3^{-1} сравниваются между собой, после M_3 сигнатура из SA_3^{-1} сравнивается с начальной сигнатурой S_0.

Следующим шагом проектирования является разработка системы команд устройства (instruction set). От правильной и оптимальной ее разработки зависит сложность внутренней архитектуры проектируемого устройства. Вместе с тем система команд должна позволять реализовывать любой из предполагаемых для выполнения алгоритмов, т.е. обладать гибкостью и самодостаточностью. В зависимости от решаемых задач система команд может быть универсальной или специализированной. В первом случае она основана на традиционных командах пересылки данных между регистрами и памятью, выполнения арифметических и

Таблица 1

Система команд модуля BCT

Мнемоника	Код	Описание	Параметры
START_MA	000	Начало маршевого алгоритма	Нет
END_MA	001	Конец маршевого алгоритма	Нет
START_ME	010	Начало маршевого элемента	DR (1 бит) – направление адресации
READ	100	Операция чтения	MASK (3 бит) – маска разрешения работы анализаторов; IN (1 бит) – инвертирование прочитанного значения; END (1 бит) – признак последней операции в маршевом элементе
WRITE	101	Операция записи	IN (1 бит) – инвертирование записываемого значения; END (1 бит) – признак последней операции в маршевом элементе
COMPARE	110	Сравнение сигнатур	AN1 (2 бита) – номер первого анализатора; AN2 (2 бита) – номер второго анализатора*
LOAD	111	Присвоение значений анализаторов	FR (2 бита) – источник присвоения; TO (2 бита) – приемник присвоения

* если AN2 = 0, то производится сравнение с нулевой сигнатурой

логических операций, условных и безусловных переходов. Специализированная система команд оптимизирована на реализацию только определенного класса алгоритмов и включает операции более высокого уровня, являющиеся примитивными шагами заданных алгоритмов.

Примером специализированной системы команд может служить система команд, разработанная для проектируемого устройства самотестирования (табл. 1).

Команды (см. табл. 1) по функциональному назначению условно можно разделить на три группы:

1. Команды управления процессом тестирования (START_MA, END_MA, START_ME).

2. Команды, генерирующие входные воздействия для тестируемой схемы (READ, WRITE).

3. Команды, анализирующие выходные реакции тестируемой схемы (COMPARE, LOAD).

В соответствии с этим в модуле BCT будет выделено три основных функциональных компонента: устройство управления, генератор тестовых воздействий и анализатор результатов тестирования.

Для реализации маршевого алгоритма устройство самотестирования должно циклически применять операции, входящие в состав маршевых элементов, ко всем ячейкам памяти. Регулярный характер этих циклов позволяет отказаться от команд переходов. Начало каждого маршевого элемента обозначается командой START_ME. В процессе ее выполнения запоминается адрес следующей команды – адрес первой операции чтения маршевого элемента. Вместо специальной команды о конце маршевого элемента сигнализирует единичное значение в поле END каждой операции чтения или записи. После выполнения такой команды осуществляется переход по сохраненному адресу. Таким образом, на каждом такте работы проектируемого устройства при реализации маршевого элемента будет выполняться команда чтения или записи из тестируемой памяти, что позволит достичь высокой скорости тестирования. Выход из цикла операций, входящих в один маршевый элемент, осуществляется при достижении последнего адреса памяти, что определяется по единичному значению сигнала TPG_LAST_ADDR, устанавливаемому генератором адресов.

Количество команд (семь в нашем случае) определяет размер поля «Код операции» (3 бита), отводимого для каждой команды при хранении ее в памяти команд (рис. 6). Под операнды отводится еще 5 битов каждой команды. Несмотря на различное количество и размер операндов, размер поля «Операнды» фиксированный, это позволяет упростить устройство управления модуля самотестирования. Небольшой размер этого поля объясняется специализацией проектируемого устройства на циклическое выполнение простых действий, что позволяет отказаться от использования команд условного и безусловного переходов, операций с константами, адресами и хранимыми данными.

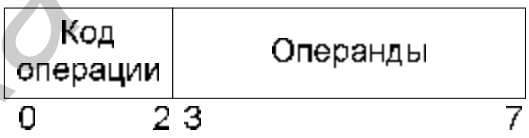


Рис. 6. Структура команды модуля BCT

С помощью предложенной системы команд для модуля BCT можно реализовать любой симметричный локально неразрушающий алгоритм с одновременным анализом не более двух симметричных элементов. Пример программы, реализующей рассмотренный выше алгоритм SMarch C-, приведен на рис. 7. Для преобразования программы из мнемонического вида (рис. 7,а) в двоичный код (рис. 7,б) был реализован простейший транслятор.

Внешняя организация и система команд определяют набор компонентов устройства, требуемый для реализации заданной функциональности. При использовании нисходящего проектирования система первоначально разбивается на функциональные блоки общего назначения без детального описания их работы. Ключевые элементы устройства самотестирования приведены на рис. 8. Ко входам тестируемой схемы памяти (RAM) подключен блок выбора режима тестирования (Test Mode Switcher, TMS). Он выполняет функции мультиплексора. В нормальном режиме работы системы он передает внешние сигналы управления памятью, в режиме тестирования – сигналы, поступающие от модуля самотестирования (Test Module, TM). Выбор режима производится с помощью входа TMS_SW (Switch). Нулевое значение на этом входе соответствует нормальному режиму, единичное – тестовому. Выходы схемы памяти возвращаются в модуль тестирования для анализа на наличие ошибок и формирования новых тестовых данных, а также передаются на внешний выход DO (Data Out).

START_MA		00000000
START_ME	1	01010000
READ	"001",0,0	10000100
WRITE	1,1	10111000
LOAD	1,3	11101110
START_ME	1	01010000
READ	"011",1,0	10001110
WRITE	0,1	10101000
COMPARE	2,3	11010110
LOAD	1,3	11101110
START_ME	0	01000000
READ	"101",0,0	10010100
WRITE	1,1	10111000
COMPARE	3,0	11011000
LOAD	1,3	11101110
START_ME	0	01000000
READ	"011",1,0	10001110
WRITE	0,1	10101000
COMPARE	2,3	11010110
LOAD	1,3	11101110
START_ME	1	01010000
READ	"100",0,1	10010001
COMPARE	3,0	11011000
END_MA		00100000
а		б

Рис 7. Программа, реализующая симметричный неразрушающий алгоритм SMarch C-:
а – в мнемоническом виде; *б* – в двоичном виде

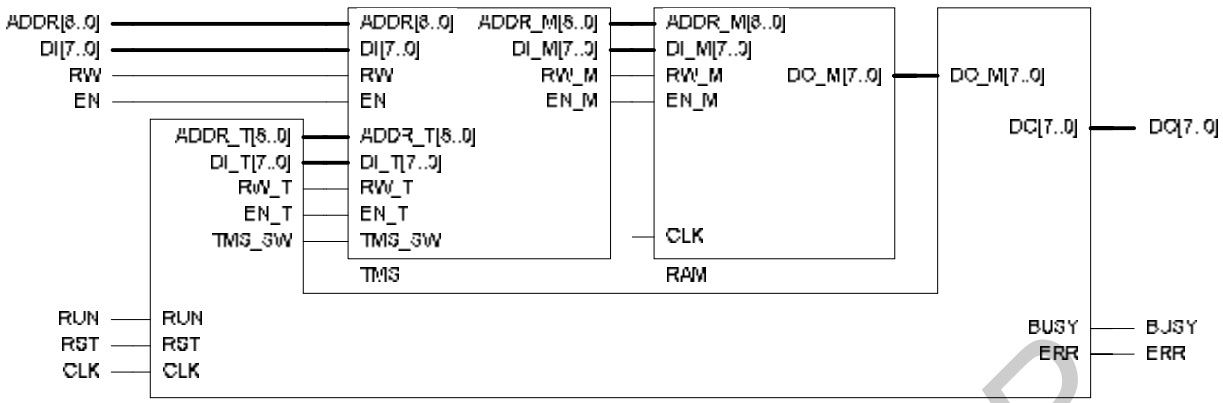


Рис. 8. Обобщенная структурная схема устройства самотестирования памяти

Продолжая цикл нисходящего проектирования, рассмотрим компоненты, составляющие блок TM (рис. 9). Данные для тестирования памяти поступают на ее входы от соответствующего генератора тестовых воздействий TPG (Test Pattern Generator), значения с выходов памяти подаются в блок анализа TRA (Test Response Analyzer). Блок управления CU (Control Unit) управляет работой всех остальных блоков.

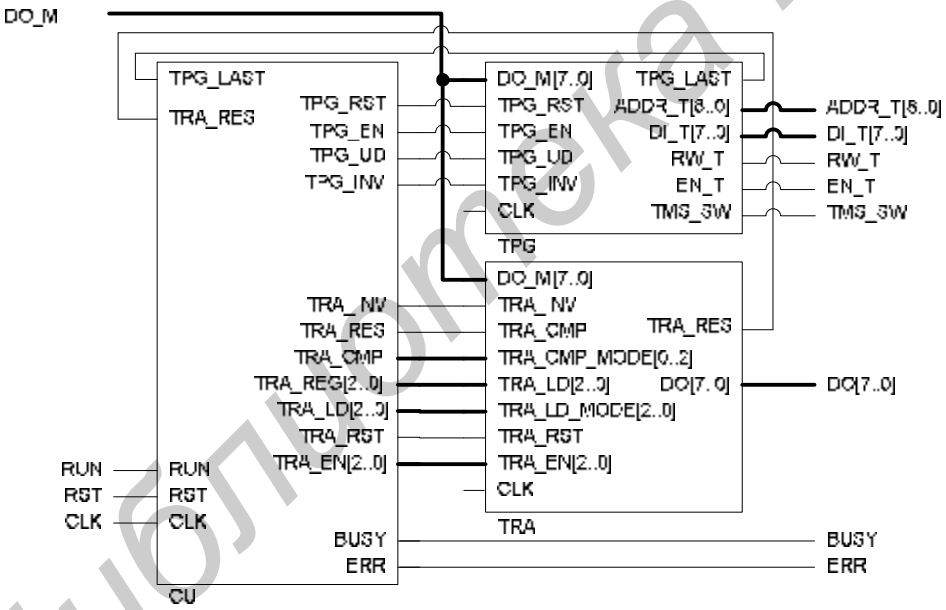


Рис. 9. Структурная схема блока TM

Блок TPG состоит из двух главных частей (рис. 10) – генератора адреса (AG, Address Generator) на основе сдвигового регистра с линейной обратной связью и инвертирующего регистра (LDATA) для хранения предыду-

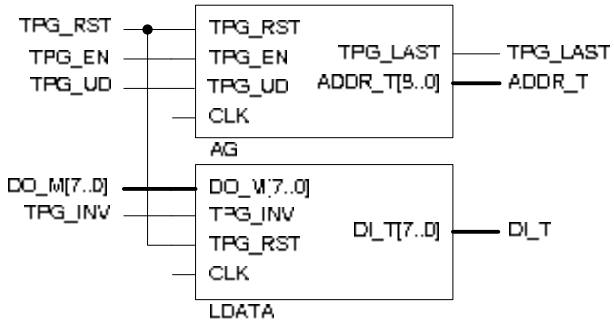


Рис. 10. Структурная схема блока TPG

щего прочитанного значения. Сигнал TPG_RST инициализирует эти регистры. TPG_EN разрешает работу генератора, TPG_UD задает направление перебора адресов («0» – от старших адресов к младшим, «1» – наоборот). TPG_INV определяет, будет ли проинвертировано значение в регистре LAST_DATA.

Задачей блока TRA является анализ получаемых в процессе тестирования схемы памяти данных, на основе которого делается вывод о наличии или отсутствии неисправностей. В его состав входят три сигнатурных анализатора. Все они объединены между собой, так что выход любого из них может быть подан на вход другого (рис. 11).

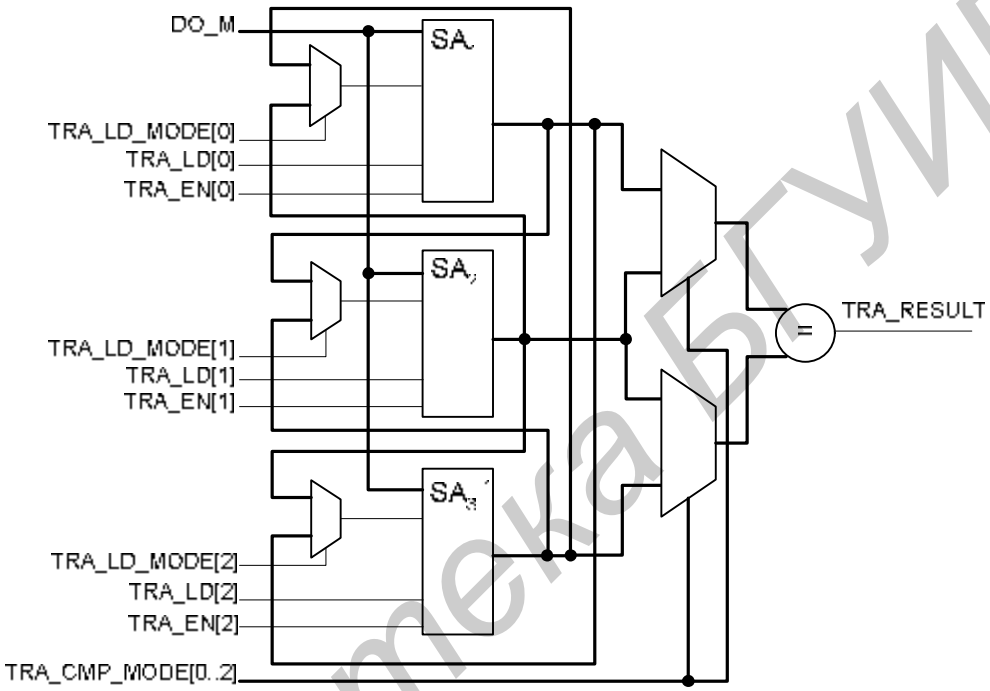


Рис. 11. Структурная схема блока TRA

Блок управления CU (Control Unit) является центральным компонентом системы. Он содержит ПЗУ, хранящее команды для выполнения. После инициализации с помощью сигнала RST выполнение программы начинается с самой первой команды. Это происходит при установке высокого уровня сигнала на входе GO. Выполнение программы продолжается до достижения команды завершения END_MA.

Эффективная работа модуля самотестирования памяти должна выполняться на максимально возможной частоте доступа к тестируемому устройству. Эта задача может быть эффективно решена при помощи конвейерной реализации блока управления CU, в котором применен трехступенчатый конвейер. Первая ступень (LD) служит для выборки и декодирования команды. На двух других рабочих ступенях (EX1 и EX2) реализуются задаваемые командой действия. При разработке системы команд для устройств с конвейерной архитектурой особое внимание должно быть уделено тому, чтобы команды обладали одинаковой сложностью, т.е. требовали для своей реализации не более заданного количества

ступеней конвейера. Это объясняется тем, что длина конвейера определяется максимальным количеством ступеней, необходимым для реализации команд. Необходимость введения двух рабочих ступеней для проектируемого устройства обусловлена двухэтапным характером команды READ: на первом этапе на вход тестируемой схемы памяти подаются управляющие воздействия; на втором – ее выходы передаются в блок анализа тестовых данных.

1.3. Ввод описания проекта

После формального описания проекта следует перейти к вводу его описания в системе автоматизированного проектирования с целью дальнейшего синтеза готового устройства. Современные САПР позволяют создавать эффективные, наглядные, управляемые и контролируемые описания проектов и их компонентов. Используемые способы пригодны как для описания проекта в целом, так и для описания его отдельных фрагментов. Один и тот же блок может быть описан с помощью различных средств. Более того, большинство САПР позволяют трансформировать один вид описания в другой. Грамотный выбор способа описания и внутренней организации или поведения разрабатываемого устройства способен существенно сократить время создания проекта, поскольку может упростить ввод его описания и тестирование. В настоящее время наиболее распространенными и универсальными способами описания являются графический и текстовый. Реже используется непосредственная трассировка схем ПЛИС в редакторе топологии, описание в виде требуемых временных диаграмм, таблиц истинности и др.

Главными достоинствами графических способов являются простота и наглядность, обусловленные привычностью проектировщиков к восприятию изображений схем. Но эти преимущества проявляются только при правильной иерархической и структурной декомпозиции проекта. В большинстве случаев графическое представление не заменяет текстовое представление, а только предваряет его.

К недостаткам графического представления можно отнести отсутствие четких стандартов соответствия текстового и графического представления. В отличие от текстовых, графические способы представления проекта обычно узко специализированы и требуют особых средств для переноса информации о проекта в другую среду. А зачастую такой перенос и вовсе не осуществим.

Современные языки описания аппаратуры допускают текстовое описание проектируемого устройства, как с точки зрения его поведения, так и точки зрения его структуры. Эти возможности позволяют использовать представление проекта в форме текстового описания алгоритмов функционирования его фрагментов в сочетании с текстовым же описанием межблочных соединений. Таким образом, использование для описания проекта текстовое HDL-описание является наиболее универсальным способом. Достоинства этого способа заключаются в его компактности и относительной простоте автоматизации любых преобразований, включая начальную генерацию проекта.

Недостатком данного подхода является функциональное «богатство» современных языков описания аппаратуры. При вводе проекта таким способом следует использовать только синтезируемые операторы перечисленных языков. Примерами несинтезируемых операторов VHDL является оператор задержки на произвольный период времени **wait for** *time*, операторы работы со строковыми данными и вещественными числами, а также работы с консолью сообщений и дисковыми файлами.

САПР Aldec Active-HDL предоставляет в распоряжение проектировщика следующие средства для ввода описания проекта:

1. Схемотехнический редактор (Block Diagram Editor, BDE) для ввода структурного описания компонентов в виде схемы подкомпонентов и связей между ними.

2. Редактор конечных автоматов (Final State Machine, FSM) для ввода поведенческого описания компонентов в виде диаграммы состояний автомата и переходов между ними.

3. Текстовый редактор (Hardware Description Editor, HDE) для ввода описания компонентов на одном из языков высокого уровня (VHDL, Verilog, HandelC).

Первые два способа относятся к графическим, третий – к текстовым.

Схемотехнический ввод удобно использовать для описания выделенных на этапе формального описания проекта блоков в среду проектирования. Основные инструменты, используемые при данном способе ввода, приведены в табл.2.

Структурные компоненты системы при нисходящем проектировании создаются из заготовок символов (Fub). Их можно рассматривать как средства для проектирования функционально изолированных объектов типа «черный ящик». На текущем уровне иерархии с помощью заготовок разрабатывается только модель взаимодействия компонентов друг с другом и внешними портами с помощью одноразрядных (Wire) или многоразрядных (Bus) соединительных линий. Если несколько компонентов используют одни и те же сигналы, такие, например, как сигналы синхронизации (CLK) и сброса (RST), то для их описания удобно пользоваться символами глобальной одноразрядной (Global Wire) или многоразрядной (Global Bus) соединительной линии. Пример описания простейшей схемы, выполненный с помощью среды Aldec Active-HDL, приведен на рис. 12.

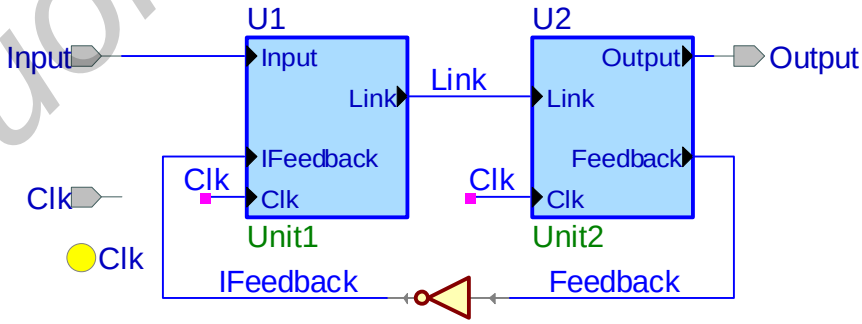


Рис. 12. Пример схемотехнического описания

Таблица 2

Инструменты редактора схемотехнического ввода Aldec Active-HDL

Инструмент	Описание
	Переводит редактор в режим выбора объектов на схемотехнической диаграмме (Select mode)
	Добавляет на схему заготовку символа (Fub)
	Открывает библиотеку готовых символов (Symbol library)
	Добавляет на схему одноразрядную соединительную линию (Wire)
	Добавляет на схему многоразрядную соединительную шину (Bus)
	Добавляет на схему одноразрядный входной порт (Input)
	Добавляет на схему одноразрядный выходной порт (Output)
	Добавляет на схему многоразрядный входной порт (Bus input)
	Добавляет на схему многоразрядный выходной порт (Bus output)
	Добавляет на схему глобальную одноразрядную линию (Global wire)
	Добавляет на схему глобальную одноразрядную шину (Global bus)
	Добавляет блок текста в заголовок генерируемого по схеме текстового HDL-описания
	Генерирует текстовое HDL-описание текущей схемы
	Просмотр сгенерированного текстового HDL-описания

Описанный в схемотехническом редакторе компонент преобразуется в текстовое VHDL-описание. При этом соединительные линии описываются в виде сигналов (**signal**), заготовки символов и сами символы объявляются в виде компонентов (**component**), а связи между ними организуются с помощью параллельных операторов **port map**. Кроме того, существует набор встроенных функциональных символов (Build-In Symbols), которые при генерации текстового описания преобразуются непосредственно в логические операторы языка. Например, для текстового описания встроенного символа инвертирования Inv используется оператор VHDL **not**. Листинг 1 получен в результате автоматического преобразования схемы, представленной на рис. 12.

Листинг 1

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

entity UnitTop is
  port(
       Clk          : in std_logic;
```

```
       Input       : in std_logic;
       Output      : out std_logic);
end UnitTop;

architecture UnitTop of UnitTop is
         component Unit1
           port (
                Clk         : in std_logic;
                IFeedback   : in std_logic;
                Input       : in std_logic;
                Link        : out std_logic);
         end component;
         component Unit2
           port (
                Clk         : in std_logic;
                Link        : in std_logic;
                Feedback    : out std_logic;
                Output      : out std_logic);
         end component;

         signal Feedback    : std_logic;
         signal IFeedback   : std_logic;
         signal Link        : std_logic;
begin
U1 : Unit1 port map(
       Clk         => Clk,
       IFeedback   => IFeedback,
       Input       => Input,
       Link        => Link);

U2 : Unit2 port map(
       Clk         => Clk,
       Feedback    => Feedback,
       Link        => Link,
       Output      => Output);

IFeedback          <= not(Feedback);

end UnitTop;
```

В процессе проработки модели, описывающей взаимодействие компонентов текущего уровня иерархии, заготовки «будущих» символов обретают все необходимые входы и выходы. После этого можно переходить к детальной разработке каждого из выделенных компонентов. Для этого достаточно выполнить команду **Push** из контекстного меню заготовки символа. В автоматически создаваемом описании символа будут содержаться все добавленные на вышестоящем шаге проектирования порты ввода/вывода. Для описания нового символа может быть выбран любой из возможных в среде Aldec Active-HDL способов: схемотехнический, текстовый или в виде диаграммы переходов конечного автомата. Однако следует иметь в виду, что описание портов символа генерируется только при первом выполнении команды **Push**. Все добавляемые впоследствии к заготовке порты требуется также добавлять к соответствующему символу вручную.

Схемотехнический редактор был использован для ввода верхнего уровня иерархии устройства BCT (рис. 13).

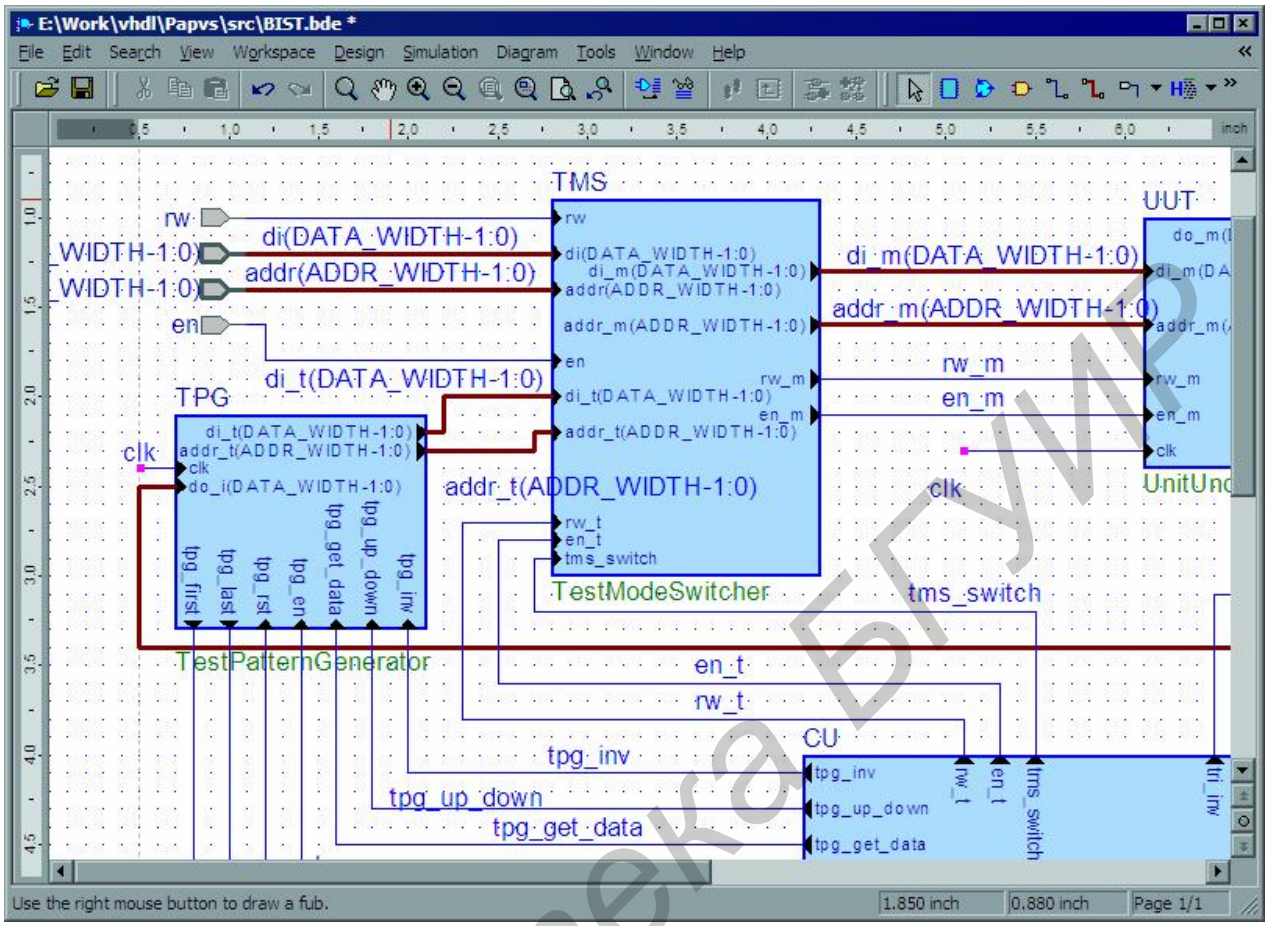


Рис. 13. Фрагмент описания устройства BCT в графическом виде

Конечные автоматы являются одними из наиболее популярных средств для описания представления последовательных компонентов системы, таких, например, как устройство управления. Редактор конечных автоматов, входящий в состав Aldec Active-HDL, позволяет вводить такие описания в наглядной графической форме в виде диаграммы состояний и переходов. Редактор позволяет свободно добавлять VHDL-код для выполнения любых действий при попадании, выходе или нахождении автомата в некотором состоянии, а также для проверки условия перехода между состояниями. На одной диаграмме могут располагаться сразу несколько работающих параллельно автоматов, разделяющих общие ресурсы и взаимодействующих с помощью сигналов. Редактор позволяет вводить как синхронные, так и асинхронные конечные автоматы. В первом случае в свойствах диаграммы требуется указать имя входного сигнала, который будет использоваться для синхронизации (CLOCK). Во втором случае − задать задержку смены состояний, вызванную распространением сигналов (Propagation Delay). Заметим, что свойством синтезируемости обладают только синхронные автоматы.

Среди прочих настраиваемых свойств автомата отметим возможность задания способа кодирования его состояний. Помимо задаваемого пользователем, возможны следующие предопределенные способы кодирования:

− двоичное (Binary) кодирование. Состояния представляются в виде последовательных двоичных чисел. Такой способ кодирования требует минимальное количество битов для реализации регистра состояния автомата;

– прямое кодирование (One-hot). Для каждого состояния в регистре состояния автомата отводиться один бит. Размер регистра состояния в этом случае равен количеству состояний;

– кодирование с помощью кода Грея (Gray). Двоичное представление каждого следующего состояния отличается от предыдущего не более чем одним битом, или сдвигом. Основным мотивом для использования такого способа кодирования является снижение вероятности ошибок при переходе между состояниями автомата.

Основные инструменты, используемые при создании диаграммы состояний и переходов, приведены в табл. 3.

Таблица 3

Основные инструменты редактора конечных автоматов

Инструмент	Описание
	Переводит редактор в режим выбора объектов на диаграмме состояний и переходов (Select mode)
	Добавляет на диаграмму новое состояние (State)
	Добавляет на диаграмму новый переход (Transition)
	Добавляет к переходу условие его выполнения (Condition)
	Добавляет к состоянию входное действие (Entry Action)
	Добавляет к состоянию действие (State Action)
	Добавляет к состоянию выходное действие (Exit Action)
	Добавляет к переходу действие (Transition Action)
	Добавляет на диаграмму входной порт (Input port)
	Добавляет на диаграмму выходной порт (Output port)
	Добавляет на диаграмму сигнал или переменную
	Добавляет на диаграмму символ сброса (RESET)
	Генерирует текстовое HDL-описание по текущей диаграмме
	Просмотр сгенерированного текстового HDL-описания

В процессе создания диаграммы состояний автомата требуется уделить особое внимание безопасности его работы. В большинстве случаев, независимо от способа кодирования, избыточность регистра состояний автомата обусловливает наличие так называемых «запрещенных» состояний, которые не описаны на

диаграмме. Переход в такие состояния может происходить при инициализации устройства или в результате сбоя. Если не предусмотреть выход из таких состояний, то при попадании в них автомат будет просто зависать. Для предотвращения таких ситуаций рекомендуется всегда добавлять к проектируемому автомату сигнал сброса, переводящий устройство в начальное состояние (Initial Sate), а также выделять одно состояние в качестве состояния по умолчанию (Trap State), переход в которое будет осуществлен при появлении в регистре состояний запрещенного кода.

В каждый момент времени автомат может находиться только в одном из состояний, переход между ними происходит на основе текущего состояния и входов автомата. В синхронном автомате все переходы осуществляются в моменты приема синхроимпульса. В зависимости от того, формируются ли выходы только на основе текущего состояния или же в их формировании принимают участие и входные сигналы, различают автоматы Мура и Мили соответственно.

Рассмотрим в качестве примера описания автомата в среде Aldec Active-HDL трехразрядный счетчик, генерирующий на выходе **data** последовательность кодов Грэя при каждом приходе синхроимпульса на вход **Clk** (рис. 14). Количество состояний (восемь) соответствует количеству возможных кодов Грэя для выбранной разрядности. Условия переходов между состояниями в данном случае отсутствуют. Переходы между состояниями приводят к изменению выхода **data**, а поскольку он зависит только от текущего состояния автомата, то проектируемый автомат является автоматом Мура.

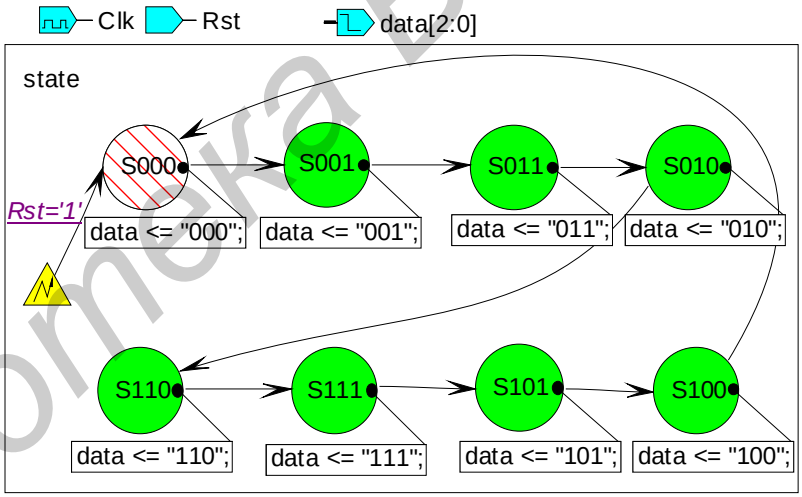


Рис. 14. Пример описания конечного автомата

В структуре аппаратно-реализуемого конечного автомата можно выделить три основные части: комбинационные блоки формирования следующего состояния и выходов автомата, а также регистр, хранящий текущее состояние (рис. 15). В соответствии с этим делением оптимальным вариантом при преобразовании диаграммы состояний и переходов конечного автомата в VHDL-описание является выделение в нем трех процессов. Такой подход позволяет гарантировать то, что средства синтеза различных производителей синтезируют автомат именно в таком виде, как его задумывал проектировщик.

Основным оператором при VHDL-описании блоков формирования следующего состояния и выходов является оператор выбора **case** (Листинг 2). Каждому состоянию соответствует один из вариантов выбора этого оператора. Состоянию по умолчанию соответствует значение выбора по умолчанию **others**.

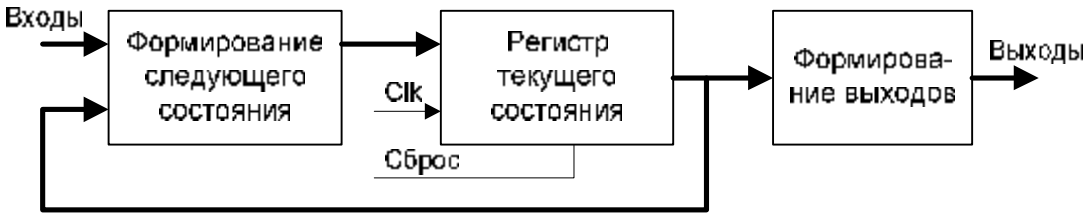


Рис. 15. Структура конечного автомата Мура

Листинг 2

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

entity FSM is
	port (
		Clk: in STD_LOGIC;
		Rst: in STD_LOGIC;
		data: out STD_LOGIC_VECTOR (2 downto 0));
end;

architecture FSM_arch of FSM is
type state_type is (S111, S110, S010, S011, S001, S000, S100, S101);
signal state, NextState_state: state_type;
signal int_data, int_data: STD_LOGIC_VECTOR (2 downto 0);
begin
-- Формирование следующего состояния
state_NextState: process (state)
begin
	NextState_state <= state;
	case state is
		when S111 =>  NextState_state <= S101;
		when S110 =>  NextState_state <= S111;
		when S010 =>  NextState_state <= S110;
		when S011 =>  NextState_state <= S010;
		when S001 =>  NextState_state <= S011;
		when S000 =>  NextState_state <= S001;
		when S100 =>  NextState_state <= S000;
		when S101 =>  NextState_state <= S100;
		when others =>
			-- Переход в состояние по умолчанию
			NextState_state <= S000;
	end case;
end process;

-- Формирование выходов
state_OutputBlock: process (int_data, state)
begin
	-- Значение по умолчанию для выходного порта
	int_data <= "000";
```

```
        case state is
                when S111 =>  int_data <= "111";
                when S110 =>  int_data <= "110";
                when S010 =>  int_data <= "010";
                when S011 =>  int_data <= "011";
                when S001 =>  int_data <= "001";
                when S000 =>  int_data <= "000";
                when S100 =>  int_data <= "100";
                when S101 =>  int_data <= "101";
                when others =>null;
        end case;
end process;

-- Синхронный регистр хранения текущего состояния
state_CurrentState: process (Clk)
begin
        if Clk'event and Clk = '1' then
                state <= NextState_state;
        end if;
end process;

-- Вывод внутренних данных на внешние порты
data <= int_data;

end FSM_arch;
```

Языковое описание устройства является наиболее универсальным на сегодняшний день способом выражения проектировщиком своего представления об устройстве и может быть использовано на различных этапах проектирования, в том числе при синтезе устройства и его моделировании. Тексты описания в большинстве языков проектирования дискретных устройств по составу синтаксических конструкций и по интерпретации результатов их исполнения (физического или модельного) очень схожи с традиционными языками программирования. Поэтому такое текстовое описание принято называть программой на языке проектирования, или, коротко, HDL-программой, а конструкции, описывающие способ формирования результатов, – операторами.

Современные языки позволяют строить модели устройств, характеризующиеся различной степенью приближения к будущей реализации, – от внешнего описания закона функционирования до детального представления проекта на уровне вентилей или макроячеек. Однако, как правило, составитель HDL-программы абстрагируется от конкретной физической реализации, выделяя прежде всего воспроизведение функционирования проектируемого изделия. После выполнения процесса верификации на основе модели строится физическое устройство.

Основным типом данных в языках описания аппаратуры являются сигналы. Их основным отличием от логических данных языков программирования являются временные характеристики. Это выражается, в частности, в том, что изменение значений сигнальной переменной может происходить не сразу после присвоения ей нового значения.

Входная информация в дискретных устройствах – цифровые сигналы, которые поступают в систему через входные контакты (порты ввода) и далее двигаются по цепям, называемым *связями*, к блокам, выполняющим те или иные преобразования, и далее на выходные контакты (порты вывода) или к другим блокам. Входным, внутренним и выходным данным проекта сопоставляются имена связей, которым соответствуют цифровые сигналы на соответствующих входных и выходных контактах проектируемой схемы или внутренних цепях проекта. Источнику сигнала, поступившего на некоторую связь, в языках сопоставляется оператор, присваивающий значения переменной, представляющей связь или сигнал на этой связи. Такие операторы называют *драйверами*.

Помимо данных сигнального типа применяется ряд служебных типов данных. Из них наиболее важными являются данные арифметического типа и строчные данные. Хотя программы могут предусматривать преобразования таких данных, эти преобразования реализуются только на этапе компиляции и, может быть, моделирования проекта. При реализации в аппаратуре используются только окончательные результаты преобразований, обычно в форме констант, задающих конфигурацию аппаратных средств.

Одним из наиболее популярных на сегодняшний день языков описания аппаратуры является VHDL. Он разработан в США в начале восьмидесятых годов по заданию Министерства обороны как язык спецификации проектов с целью единообразного понимания подсистем различными проектными группами. В 1987г. спецификация VHDL была принята в качестве стандарта ANSI/IEEE STD 1076-1987, называемого VHDL'87. Удобства и относительная универсальность конструкций этого языка достаточно быстро привели к созданию программ моделирования систем на основе их описания в терминах VHDL.

С начала девяностых годов разрабатываются прямые компиляторы VHDL-программ в аппаратные реализации различных классов. Это наряду с необходимостью более адекватного представления в языке современных тенденций в цифровой схемотехнике стало стимулом усовершенствования языка и привело к созданию расширенного стандарта ANSI/IEEE STD 1076-1993 или, кратко, VHDL'93. Сегодня трудно найти САПР дискретных устройств, которая не воспринимает описания устройств на VHDL или хотя бы на его подмножествах.

Возможно несколько подходов к текстовому описанию функционирования дискретных систем, называемых стилями проектирования. Выбор стиля во многом определяется наклонностями и опытом разработчиков, но надо иметь в виду, что часто стиль существенно влияет на порождаемую системой автоматизированного проектирования реализацию.

1. *Поведенческое описание* устройства задает отображение входов в выходы, абстрагируясь от деталей реализации устройства. Модель строится из двух VHDL-модулей: entity и architecture. Первый модуль описывает параметры настройки и порты. Второй модуль – архитектура поведенческого уровня – определяет множество процессов (**process**), реализующих функциональность устройства и работающих асинхронно по отношению друг к другу. Поведение каждого

процесса описывается последовательными операторами, которые выполняются друг за другом в соответствии с потоком управления. В поведенческом стиле создаётся большинство компонентов нижнего уровня иерархии проектирования. Пример такого описания приведен в листинге 3.

Листинг 3

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

entity RAM is
      port(
            di  : in   std_logic_vector(DATA_WIDTH-1 downto 0);
            addr: in   std_logic_vector(ADDR_WIDTH-1 downto 0);
            do  : out  std_logic_vector(DATA_WIDTH-1 downto 0);
            rw  : in   std_logic;
            en  : in   std_logic;
            clk : in   std_logic);
end RAM;

architecture ram_behave of RAM is
      subtype   word is std_logic_vector(DATA_WIDTH-1 downto 0);
      subtype   addresser is integer range (2**ADDR_WIDTH)-1 downto 0;
      type      memory_array is array (addresser) of word;
      signal    mem: memory_array;
      signal    internal_do: word;
begin
      process(clk)
            variable internal_addr: integer;
      begin
            if (clk'event and clk='1') then
                if (en = '1') then
                      internal_addr := to_integer(unsigned(addr));
                      if (rw = '1') then
                            mem(internal_addr)   <= di;
                            internal_do   <= di;
                      else
                            internal_do   <= mem(internal_addr);
                      end if;
                end if;
            end if;
      end process;

      process(internal_do) begin
            do  <= internal_do;
      end process;
end ram_behave;
```

2. *Структурное описание* устройства определяет используемые типы и состав компонентов устройства, соединения между входами и выходами компонентов. Соединения представляются сигналами. Основным признаком структурного описания является использование оператора **port map**. Примером струк-

турного описания является листинг 1, сгенерированный по структурной схеме устройства на рис. 12.

3. *Потоковое описание* устройства определяет поток данных в устройстве и его частях. По уровню абстракции потоковое описание является промежуточным между поведенческим и структурным. Оно характеризуется использованием параллельных операторов назначения сигналов. Такое описание удобно использовать для описания блоков с несложной функциональностью. В поведенческом стиле описан входящий в состав проектируемого устройства самотестирования условный инвертор выходных значений схемы памяти (листинг 4). В нем используется два потока: один просто передает значение со входа **do_m** на выход **do**; второй формирует на выходе **do_i** прямое или инверсное значение **do_m** в зависимости от значения сигнала **tri_inv**.

Листинг 4

```
entity TestResponseInverter is
        port(
                do_m  : in std_logic_vector(DATA_WIDTH-1 downto 0);
                tri_inv : in std_logic;
                do_i  : out std_logic_vector(DATA_WIDTH-1 downto 0);
                do    : out std_logic_vector(DATA_WIDTH-1 downto 0));
end TestResponseInverter;

architecture TestResponseInverter of TestResponseInverter is
begin
       do     <= do_m;
       do_i   <= not do_m when tri_inv = '1' else do_m;
end TestResponseInverter;
```

При вводе VHDL-описания имейте в виду следующие рекомендации:

− перед переходом к кодированию проекта тщательно разработайте его архитектуру – составляющие компоненты и связи между ними;

− избегайте слишком больших компонентов. Деление на подкомпоненты позволяет синтезатору эффективно выполнять логическую оптимизацию. На сегодняшний день в силу ограничений синтезирующих САПР рекомендуется не описывать компоненты, аппаратная реализация которых требует более 5000эквивалентных вентилей. Слишком большие компоненты не могут быть эффективно оптимизированы и требуют больших временных и вычислительных затрат на синтез;

− не включайте в описание устройства временные задержки. Они определяются технологией кристалла, для которого будет произведен синтез;

− имейте четкое представление, с помощью каких аппаратных ресурсов вводимое HDL-описание может быть синтезировано. Используйте только синтезируемое подмножество языков описания аппаратуры;

− используйте подпрограммы для структурирования описания и придания ему более наглядного и читабельного вида. Используйте подпрограммы для многократного использования кода;

− по возможности используйте настраиваемые параметры (**generic**) для легкой модификации и повторного использования проекта. В первую очередь это относится к параметризации размеров соединительных шин;

− не включайте одинаковый код в различные секции условных операторов. Вместо этого размесите дублирующийся код перед условным оператором, внутри которого используйте только результат выполнения кода, предварительно сохраненный во временных переменных. Аналогично не включайте в циклы код, модифицирующий инвариантный к циклам сигналы;

− имейте в виду то, что идентификаторы языка VHDL не чувствительны к регистру используемых для их задания символов. Так, идентификаторы BIST и bist обозначают один и тот же объект VHDL-описания;

− использование абстрактных типов данных улучшает наглядность описания. Например, используйте перечисленный тип данных с осмысленными именами для обозначения состояний конечного автомата;

− используйте осмысленные названия для сигналов и компонентов;

− используйте комментарии. Снабжайте каждый модуль заголовком с описанием его назначения или функциональности. Объявление каждого значимого сигнала сопровождайте комментарием о том, как он используется.

1.4. Функциональная верификация

Проверка проекта с целью обнаружения ошибок в описании и функциональности выполняется с помощью функционального моделирования (Functional Simulation). Временные характеристики будущего кристалла не учитываются. Для больших иерархических проектов возможна функциональная верификация каждого модуля.

Специфика моделирования дискретных устройств заключается в том, что процессы в них, как и вообще в реальном мире, происходят параллельно во времени, причем изменения могут происходить асинхронно и относительно независимо друг от друга. При моделировании такое поведение должно быть воспроизведено с требуемой степенью точности последовательными алгоритмами, реализуемыми в ЭВМ. Для правильного понимания принципов языкового описания и результатов моделирования следует достаточно четко представлять методы, заложенные в подсистемы моделирования современных САПР.

В основу работы систем моделирования положена дискретная событийная модель. На каждом шаге моделирования пересчитываются состояния только тех компонентов, на входе которых в данный момент происходят изменения, называемые *событиями*. Любое событие может вызвать цепочку других событий. Моделирование системы производится не для равномерно отстоящих моментов реального времени, а лишь для моментов, для которых ранее были предсказаны события.

Событийная модель, помимо совокупности таблиц, представляющих структуру моделируемого устройства, использует специфическую структуру данных – календарь событий. Он представляет собой список, каждый элемент

которого представляет запись, хранящую значение времени наступления события, имя изменяемого сигнала и предсказанное на интервал времени после наступления события значение этого сигнала. Элементы списка упорядочены по возрастанию времени появления событий. Перед началом моделирования в календарь событий заносятся все запланированные в эксперименте события на входах, т.е. изменения входных сигналов. Пересчёт значений сигналов в процессе моделирования приводит к добавлению в календарь событий новых записей. Если в календаре событий нет событий, предсказанных на время, позднее текущего, то моделирование завершается.

Событийное моделирование без учета физических задержек распространения сигналов предполагает наличие специфической бесконечно малой задержки, называемой дельта-задержкой (δ-задержка). Дельта-задержка не несёт информации о реальном времени передачи сигналов, а отражает причинно-следственные связи в объекте моделирования. Вызванные одним событием изменения приводят к возникновению других событий, которые заносятся в календарь после предыдущих событий, запланированных на это же время. Отметка времени у предсказанных событий точно такая же, как и у инициирующего их события, их разделяет только модельная дельта-задержка. Предсказанное событие не влияет на исходные данные для воспроизведения поведения остальных компонентов в циклах моделирования, предшествующих циклу отработки этого события.

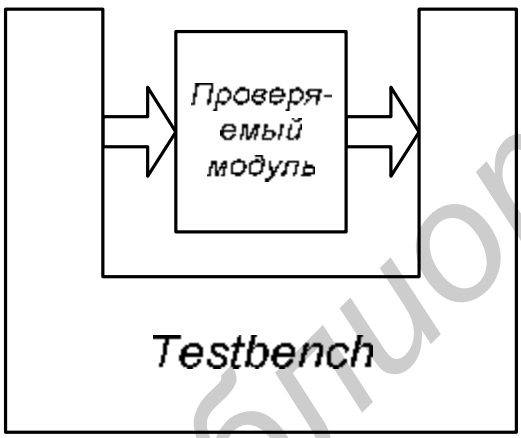


Рис. 16. Схема верификации с помощью Testbench

Выполнение функционального моделирования невозможно без подачи схемы внешних воздействий. Традиционным средством верификации VHDL-описаний цифровых устройств являются специальные тестовые модули (Testbench). Такой модуль подключается к портам тестируемой схемы (рис. 16), формирует на ее входах тестовые воздействия и анализирует функциональную правильность ее работы по выходам. Сам модуль Testbench представляет собой закрытую систему без внешних входов или выходов.

Модули Testbench также создаются с помощью VHDL. Поскольку они не входят в состав проектируемой системы, а используются только в процессе проектирования, то возможно использование всех операторов языка, включая операторы работы с файлами, консолью и операторы задержек на произвольный интервал времени. Набор верификационных данных может читаться из файла или формироваться самим тестовым модулем алгоритмически.

Пример модуля Testbench приведен в листинге 5. В процессе разработки устройства ВСТ этот модуль использовался для проверки правильности функционирования VHDL-описания схемы тестируемой памяти. Testbench читает из текстового файла (data.txt) верификационные данные, представляющие собой

текстовые записи следующего формата: <операция> <адрес> <значение>. Для операции чтения указывается значение, которое должно быть записано в ячейку с указанным адресом, для операций записи – значение, которое должно быть прочитано.

Листинг 5

```
entity ram_tb is
end ram_tb;

architecture TB_ARCHITECTURE of ram_tb is
      -- Компонент для верификации
      component ram
      port(   di  : in std_logic_vector((DATA_WIDTH-1) downto 0);
          addr: in std_logic_vector((ADDR_WIDTH-1) downto 0);
          do  : out std_logic_vector((DATA_WIDTH-1) downto 0);
          rw  : in std_logic;
          en  : in std_logic;
          clk : in std_logic);
      end component;

      -- Стимулирующие сигналы
      signal di   : std_logic_vector((DATA_WIDTH-1) downto 0);
      signal addr : std_logic_vector((ADDR_WIDTH-1) downto 0);
      signal rw   : std_logic;
      signal en   : std_logic;
      signal clk  : std_logic;
      -- Выходные сигналы
      signal do   : std_logic_vector((DATA_WIDTH-1) downto 0);
begin
      UUT : ram256x8 port map (   di => di, addr => addr, do => do,
                                  rw => rw, en => en, clk => clk);

      STIM:process
          file f          : TEXT open READ_MODE is "data.txt";
          variable l      : LINE;
          variable operation: character;
          variable addr   : integer range 0 to 2**(ADDR_WIDTH-1);
          variable data   : integer range 0 to 2**(DATA_WIDTH-1);
      begin
          en <= '1';
          while not (endfile(f)) loop
              readline(f,l);
              read(l,operation); read(l,addr); read(l,data);
              case operation is
              when 'r' =>
                  rw  <= '0'; clk <= '1';
              when 'w' =>
                  rw  <= '1'; clk <= '1';
                  di  <=
                      std_logic_vector(to_unsigned(data,DATA_WIDTH));
              when others => null;
              end case;
              wait for 100ns;
              clk <= '0';
              if (operation = 'r') then
```

```
                assert (to_integer(unsigned(do)) = data)
                    report "Прочитанное неверное значение"
                    severity WARNING;
            end if;
            wait for 100ns;
        end loop;
        wait;
    end process;
end TB_ARCHITECTURE;
```

Кроме функциональной верификации, Testbench может использоваться для сравнения идентичности работы введенного описания разрабатываемой схемы и результатов синтеза (рис. 17). Для этого на соответствующие входы обоих моделей подаются одинаковые сигналы, а выходы сравниваются компаратором.

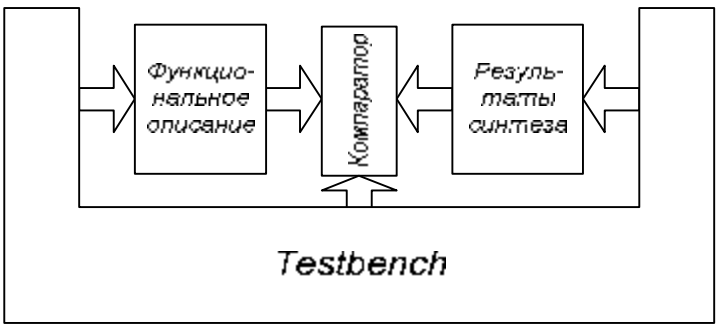


Рис. 17. Схема параллельной верификации двух моделей с помощью Testbench

Среда Aldec Active-HDL предоставляет в распо-

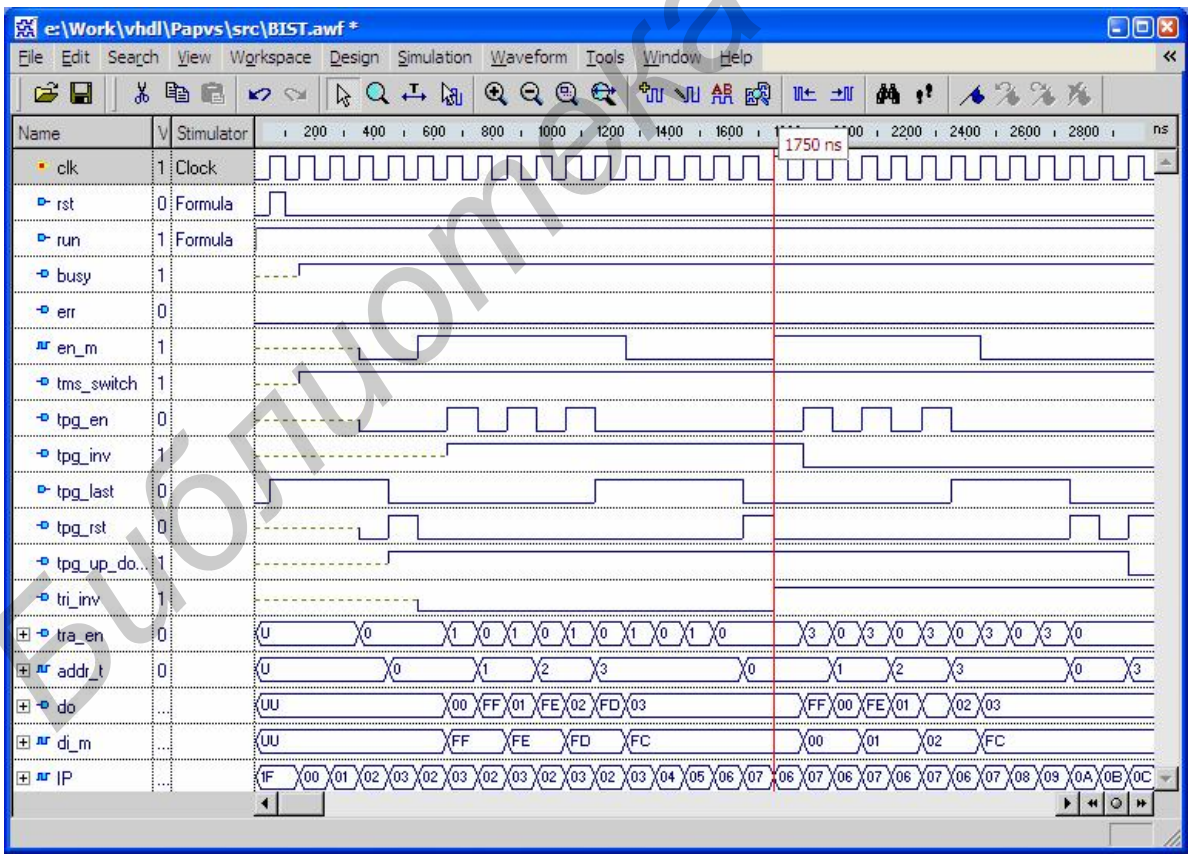


Рис. 18 Функциональная верификация устройства ВСТ с помощью редактора временных диаграмм Aldec Active-HDL

ряжение разработчиков еще один удобный инструмент функциональной верификации – редактор временных диаграмм. На рис. 18 представлен процесс верификации устройства ВСТ с его помощью. Основные инструменты редактора перечислены в табл. 4.

Временные диаграммы могут использоваться не только для просмотра результатов работы устройства, но и для подачи симулирующих импульсов на входы схемы. Для этого используется специальные инструменты – стимуляторы (табл. 5). Значения, формируемые стимуляторами, могут иметь постоянный или переменный во времени характер.

Таблица 4

Основные инструменты редактора временных диаграмм

Инструмент	Описание
	Переводит редактор в режим выбора (Select Mode)
	Переводит редактор в режим масштабирования (Zoom Mode)
	Переводит редактор в режим измерений (Measurement Mode)
	Переводит редактор в режим изменений (Edit mode)
	Увеличивает, уменьшает или вмещает диаграмму в текущее окно
	Добавляет на диаграмму сигнал (Signal)
	Добавляет к выбранному сигналу стимулятор (Stimulator)
	Сравнивает текущую диаграмму с другой диаграммой из файла
	Переводит курсор к введенной отметке времени

Таблица 5

Типы стимуляторов редактора временных диаграмм

Тип	Описание
1	2
	Фиксированные стимуляторы Value присваивают определенное значение выбранному сигналу. Значения могут быть присвоены любым сигналам и шинам и изменены в любое время в течение моделирования. Среда моделирования выполняет проверку типа сигнала чтобы удостовериться, что указанное значение допустимо для выбранного типа сигнала
	Формульные стимуляторы позволяют задать характер изменения сигнала во времени с помощью формул. Например, формула «`0 0, 1 10`» описывает сигнал, который в начальный момент времени получает значение «`0`» и изменяется в «`1`» через `10` единиц модельного времени
	Стимуляторы по «горячей» клавише Hotkey позволяют управлять значениями сигнала непосредственно с клавиатуры. Нажатие назначенной сигналу «горячей» клавиши переключает значение сигнала

Окончание табл. 5

1	2
 Clock	Синхростимуляторы Clock предназначены для моделирования формы синхросигнала. Графический редактор синхросигнала среды Active-HDL позволяет задавать такие его параметры, как частота, период, начальный сдвиг, соотношение времени нахождения в высоком и низком уровне сигнала
 Counter	Стимулятор-счетчик используется для подачи счетной последовательности значений на входные многоразрядные шины данных. Возможно задание различных типов счетных последовательностей, различных систем счисления для двоичных счетчиков, начальных значений, шага и направления счета
 Predefined	Предопределенные стимуляторы – это набор синхросигналов с предопределенной частотой, наиболее часто используемых при верификации различных проектов
 Custom	Характер изменения пользовательских стимуляторов задается разработчиком путем непосредственного редактирования временной диаграммы
 Random	Работа случайных стимуляторов основана на генераторе псевдослучайных чисел. Он возвращает целочисленные значения, распределенные в соответствии с одним из законов распределения: равномерным, нормальным, экспоненциальным, Пуассона и др.

Среда Active-HDL содержит богатые вспомогательные средства для верификации проектов с помощью графического пользовательского интерфейса (Graphical User Interface, GUI). Эти средства включают:

– проверку синтаксиса (Syntax Check) с подробным указанием места и типа ошибки;

– отладчик VHDL-описаний с возможностями пошаговой отладки (Code Tracing) и установки точки останова (Breakpoint) на любую выполняемую строчку кода или на значение сигнала;

– журнал изменений сигналов (List View), показывающий все изменения интересующих сигналов и портов схемы для каждого момента модельного времени и для дельта-задержек;

– окно просмотра текущих значений переменных и сигналов (Watch Window);

– окно просмотра текущего состояния процессов (Processes Window);

– окно стека вызовов подпрограмм (Call Stack Window).

Для упрощения моделирования дискретных устройств при их описании следует следовать следующим рекомендациям:

– использование одного процесса (**process)** вместо нескольких параллельных операторов назначения сигналов позволяет сократить количество событий, помещаемых в календарь событий, что ускоряет моделирование проекта;

− минимизируйте количество сигналов в списке чувствительности процессов. Это также способствует ускорения моделирования;

− сокращайте количество небольших по объему процессов, поскольку на активацию и дезактивацию каждого из них требуется определенное время и ресурсы системы моделирования. Если несколько регистров тактируются одним и тем же синхроимпульсом, их описание можно поместить в один процесс;

− для описания параллельных процессов вместо оператора **block** используйте оператор **process**, поскольку оператор **block** не имеет списка чувствительности и всегда активен во время моделирования;

− по возможности преобразуйте векторные типы данных, например **std_logic_vector**, в целочисленные типы, например **integer**;

− используйте в процессах переменные (**variable**) вместо сигналов (**signal**) где только возможно.

1.5. Синтез проекта

Этот этап состоит из двух частей. Сначала выполняется синтез VHDL-описания в независимые от технологии логические примитивы (RTL- или макросинтез). При этом операторы сложения синтезируются в сумматоры, операторы сравнения – в компараторы, операторы выбора – в мультиплексоры и т.д. На следующем шаге эти примитивы преобразуются в элементы той технологии, для которой выполняется синтез. Для Xilinx Spartan2E, например, все комбинационные и последовательные компоненты схемы синтезируются при помощи элементов конфигурируемых логических блоков (КЛБ) – функциональных генераторов (Loook-Up Table, LUT), логики выбора и быстрого переноса, однобитных запоминающих элементов. Для синтеза запоминающих элементов большого объема используются блоки встроенной синхронной памяти (BlockRAM).

Оба этапа можно выполнить с помощью Xilinx ISE или Synplicity Synplify. Мы рекомендуем использовать второй продукт, поскольку он обладает более мощными средствами просмотра и анализа результатов синтеза. Однако следует помнить, что все операторы VHDL моделируемы, но не все – синтезируемы. Это значит, что проект, успешно прошедший функциональную верификацию, не всегда может быть успешно синтезирован.

Рассмотрим результат синтеза описания ОЗУ из листинга 3 с помощью Synplicity Synplify 7.6 со следующими параметрами: DATA_WIDTH (ширина шины данных) = 8, ADDR_WIDTH (ширина шины адреса) = 12. Приведенная на рис. 19 схема в точности соответствует поведенческому описанию на VHDL. Массив **mem** используется для хранения информации. Его размер в точности соответствует заданным параметрам. Входной син-

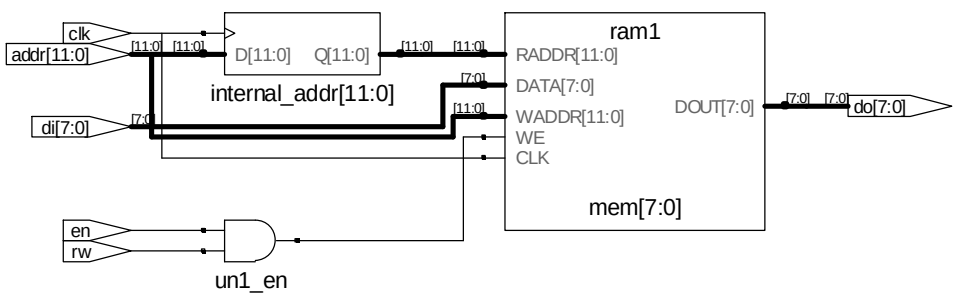


Рис. 19. Результат макросинтеза схемы памяти

хронный триггер **internal_addr** хранит текущее значение адреса. Высокий уровень сигнала на входе **en** запрещает запись в массив **mem** и вывод с защелки **do**.

Вторым этапом синтеза является представление полученного макроописания с помощью технологически-зависимых элементов. Так, для выбранной платформы Xilinx Spartan2e существует два способа реализации массива запоминающих элементов:

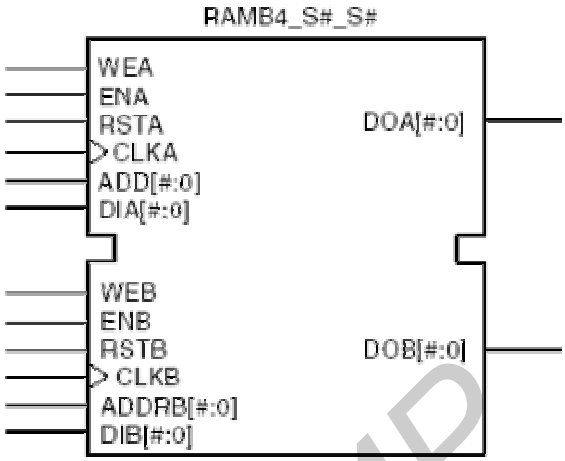


Рис. 20. Двухпортовая память BlockRAM

− с помощью входящих в состав каждого конфигурируемого логического блока триггеров;

− с помощью блоков встроенной памяти BlockRAM.

Естественно, что второй способ эффективен для реализации больших целостных массивов запоминающих элементов. Первый способ используется при синтезе массивов небольших размеров и в тех случаях, когда разработчику требуется больше памяти, чем можно синтезировать с помощью BlockRAM.

Таблица 6

Соотношение размеров шины адреса и данных BlockRAM

Размер ячейки	Количество ячеек	Шина адреса	Шина данных
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

Каждый блок памяти BlockRAM – это полностью синхронное двухпортовое 4096-битное ОЗУ с независимым управлением для каждого порта (рис. 20). Размерность шины данных для обоих портов может быть сконфигурирована независимо, что позволяет создавать преобразователи размерности шины. В табл.6 показаны возможные соотношения размерностей шин данных и адреса. Всего на кристалле Xilinx Spartan 2E XC2S200E содержится 14 блоков BlockRAM, что позволяет использовать до 56K памяти.

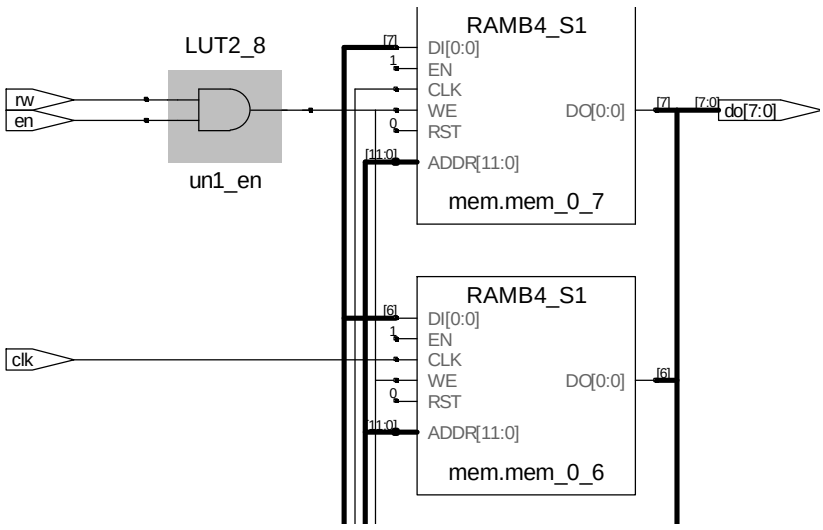


Рис.21. Технологический синтез схемы памяти

В соответствии с имеющимися технологическими ресурсами полученный при макросинтезе запоминающий массив **mem** заданной размерности 12x8 представляется в виде 8 блоков BlockRAM, сконфигурированных с размерностью 12x1 (рис.21). Таким образом, все биты слова памяти хранятся в различных блоках встроенной памяти. Входной триггер **internal_addr** удален в результате оптимизации, поскольку его функции выполняются в самом блоке BlockRAM.

Сводная информация о результатах синтеза выводится в специальный файл отчета (SRR). В нем можно получить следующую информацию:

- замечания и ошибки, возникшие в процессе работы синтезатора;
- максимальная задержка распространения сигналов в устройстве;
- критический путь с максимально задержкой распространения;
- сводный перечень ресурсов, потребовавшихся для реализации устройства.

Следует внимательно относиться ко всем замечаниям синтезатора, поскольку они, как правило, связаны с ошибками в описании устройства. Например, сообщение «FX234: Could not implement Block RAM» означает, что для синтеза запоминающегося устройства не использовались блоки встроенной памяти BlockRAM. Это может быть связано с тем, что для операций чтения и записи использовались либо различные синхросигналы, либо различные сигналы разрешения. Сообщение «CL117: Latch generated from process for signal» вызвано тем, что при условном назначении были рассмотрены не все варианты значений сигнала-условия, что привело к невозможности синтезировать комбинационную схему.

Пример сводного отчета о затраченных при синтезе ресурсах приведен в листинге 6. Основными элементами, использованными при синтезе, являются конфигурируемые функциональные генераторы (LUT – Look-Up Table), различные типы D-триггеров (D Flip-flop), мультиплексоры (MUX) и блоки памяти BlockRAM (RAMB).

Листинг 6

```
Mapping to part: xc2s200eft256-6
Cell usage:
FDC             2 uses
FDCE            5 uses
FDCP            12 uses
FDE             1 use
FDP             5 uses
FDPE            5 uses
GND             4 uses
LD              13 uses
MUXCY           1 use
MUXCY_L         20 uses
RAMB4_S1        8 uses
VCC             3 uses
XORCY           17 uses
```

Для эффективного синтеза проекта следует принимать во внимание следующие рекомендации:

– при описании комбинационной логики на основе оператора **if** убедитесь, что значения сигналам присваиваются в обеих ветках, чтобы избежать добавления синтезатором защелок;

– при описании комбинационной логики на основе оператора **case** присвойте значения по умолчанию всем выходным сигналам перед оператором **case** или используйте присвоение всем сигналам в каждой его

ветке. Это также поможет избежать добавления синтезатором непредусмотренных защелок;

− используйте оператор **case** вместо вложенной конструкции из операторов **if**, поскольку последняя реализуются с помощью приоритетных шифраторов вместо мультиплексоров;

− не используйте целочисленные типы без определения границ их изменения. По умолчания рассматривается максимальное возможное из допустимых значений. Для типа **integer**, например, в стандарте VHDL'93 по умолчанию отводится 32 бита. Это требует у синтезатора дополнительного времени для оптимизации неиспользуемой логики;

− используйте типы данных, описанные в пакетах **IEEE.STD_Logic_1164** и **IEEE.Numeric_STD**, в качестве функционального минимума;

− значения объектов, модифицируемых внутри цикла **for**, должны быть проинициализированы непосредственно перед ним;

− внутри оператора циклов **for-loop** предпочтительно использовать только присвоение значений переменным, а не сигналам.

1.6. Постсинтез и временная верификация

Постсинтез верификация необходима для проверки соответствия синтезированного проекта и его исходного описания, поскольку результаты синтеза также представляются в виде описаний на языке VHDL, верификация которых выполняется аналогично функциональной верификации.

Главной задачей временной верификации является анализ размещенного и трассированного проекта с учетом знаний о временных характеристиках выбранного кристалла и введенных пользователем временных ограничений. На основе временных параметров, использованных для синтеза компонентов, автоматически вычисляются три основных класса временных параметров:

− минимальные и максимальные задержки между источниками (входными сигналами) и приемниками (выходными сигналами), информация о которых выдается в виде матрицы задержек;

− максимально возможная производительность устройства (пропускная способность) в виде максимальной чистоты тактирования устройства;

− критический путь передачи и преобразования информации.

1.7. Имплементация проекта

На этом этапе выполняется окончательная привязка результатов технологического синтеза для конкретной платы ПЛИС. Каждый синтезированный технологический элемент связывается с элементом физического устройства. Итерационно выполняются размещение и трассировка проекта на кристалле. Целью этой операции является получение оптимального размещения в соответствии с одним из заданных пользователем критериев: по площади, скорости или балансу площади и скорости. Строится и оптимизируется карта межблочных соединений, входы и выходы схемы связываются с физическими портами ПЛИС.

Требования к физическим параметрам устройства, которые должны выполняться средством имплементации при технологическом синтезе проекта, задаются в специальном файле пользовательских настроек (User Constraints File, UCF). Например, в этом файле могут задаваться температурные и временные ограничения на работу модуля. Однако главным параметром, необходимым для работы спроектированного устройства, является привязка его портов к контактам схемы проектируемой логики (Pins assignment).

Файл пользовательских настроек представляет собой обычный текстовый файл и может редактироваться вручную или с помощью специализированного редактора Xilinx Constraint Editor (рис. 22). Для создания файла настроек выберите пункт меню `Project -> New Source`, затем тип файла «`Implementation Constraints File`» и задайте его имя.

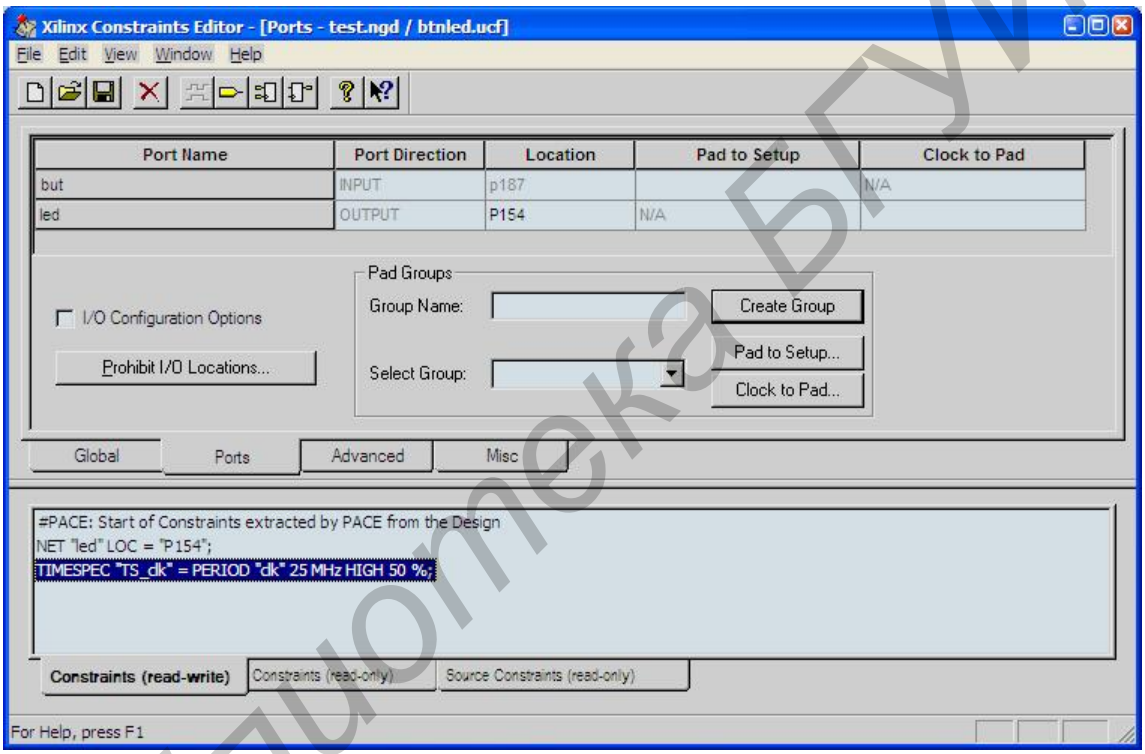


Рис. 22. Графический редактор пользовательских настроек

Формат текстовой записи, связывающей именованный порт устройства с контактом схемы перепрограммируемой логики:

net <имя_порта> **loc** = <имя_контакта>;

Например, для подключения именованного порта **clk** к контакту № 182 требуется записать:

net clk **loc** = p182;

Разрабатываемый модуль ВСТ не является самостоятельным устройством и предназначен для внедрения в системы более высокого уровня в качестве отдельного компонента. Поэтому для его системной верификации потребовалось создать модуль, выполняющий функции контроллера памяти реального устройства. В интерфейсную часть этого модуля вошли следующие порты:

− but – входной порт для сигнала от нажимаемой пользователем на плате Digilent D2-SB кнопки BUT. Используется для запуска модуля ВСТ;

− led – выходной порт, связанный с индикатором LED на плате Digilent D2-SB. Индикатор загорается в процессе работы модуля ВСТ;

− clk – входной порт для передачи синхросигнала с частотой 50MHz от встроенного генератора тактовых импульсов на модуль ВСТ.

UCF-файл для такого верификационного модуля приведен в листинге 7.

Листинг 7

```
net    but    loc  = p187;
net    led    loc  = p154;
net    clk    loc = p182;
net    clk    tnm_net      = clk;
timespec "TS_clk" = period clk 25 MHz high 50%;
```

1.8. Программирование кристалла

Полученный в результате имплементации проекта файл конфигурации загружается в ПЛИС через COM-, USB- или PCI-интерфейс. Для плат перепрограммируемой логики этот процесс производится с помощью входящей в состав Xilinx ISE утилиты iMPACT (рис. 23). Работа с ней выполняется в пошаговом режиме:

1. Выбор режима работы утилиты – программирование кристалла (Configure Devices).
2. Выбор метода конфигурирования схемы – через интерфейс граничного сканирования (Boundary Scan Mode).
3. Выбор типа интерфейса граничного сканирования – ручное задание (Enter a Boundary Scan Chain).
4. Задание файла с описанием интерфейса граничного сканирования – `<Xilinx>\spartan2e\data\xc2s200e_pq208.bsd`.
5. Задание файла описания устройства – `<имя_проекта>.bit`.
6. Непосредственное программирование кристалла (`Program...`).

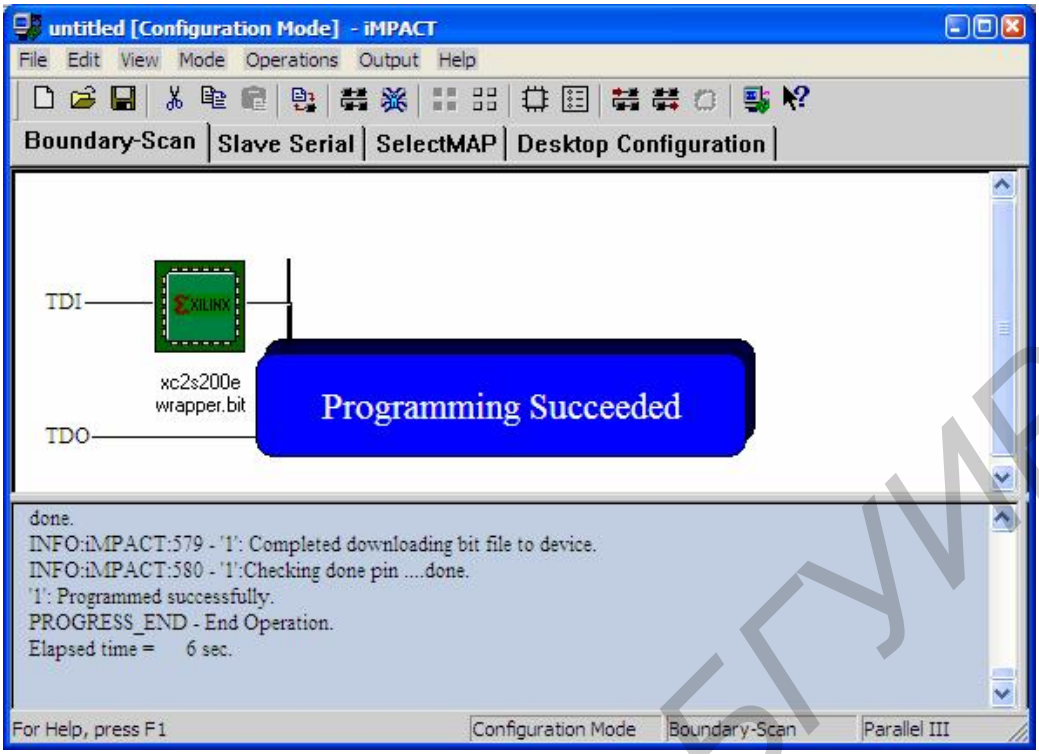


Рис.23. Программирование кристалла с помощью iMPACT

1.9. Системная верификация

Системная верификация представляет собой окончательную экспериментальную проверку правильности работы полученной цифровой системы (рис. 24). При всей тщательности проведения предыдущих этапов всегда существует вероятность того, что в проекте имеются дефекты, которые могут проявиться на этапе внедрения или даже штатного использования устройства.

Рис. 24. Системная верификация

2. РЕАЛИЗАЦИЯ ОБМЕНА ДАННЫМИ МЕЖДУ ХОСТ-КОМПЬЮТЕРОМ И ПЛАТОЙ ПЕРЕПРОГРАММИРУЕМОЙ ЛОГИКИ

Микропрограммный принцип построения используется для быстрого изменения алгоритма работы цифрового устройства в процессе его эксплуатации. В данном случае для модификации алгоритма устройство подключается к программатору (в роли которого может выступать обычный компьютер), записывающему новую программу в память устройства. Обмен данными между программатором и программируемым устройством производится с использованием заранее заданного протокола обмена (интерфейса). Можно воспользоваться стандартным интерфейсом, например JTAG, либо разработать свой интерфейс.

Рассмотрим модуль, обеспечивающий связь проекта, реализованного на базе схемы перепрограммируемой логики Xilinx Spartan 2E с ПК. Соединительный кабель подключен к разъему расширения B2 платы и разъему DB25 (параллельный порт) компьютера. Контакты 1–8 параллельного порта (рис.25) могут работать в режиме приема/передачи данных, контакты 10–13, 15 – только в режиме приема, контакты 14, 15, 17 – только в режиме передачи. Контакты 18-25 – «земля». Для доступа к контактам 2–8 используется порт 378h «Data Latch» (зависит от режима работы порта, выбранного в BIOS). Для чтения с контактов 10–13, 15 используется порт 379h «Printer Status». Для доступа к контактам 1, 14, 16, 17 – порт 37Ah «Printer Controls».

В связи с тем, что мы будем использовать один и тот же кабель как для прошивки платы через JTAG, так и для нашего интерфейса, выберем контакты, не используемые при прошивке платы. На рис.25 не заштрихованы выбранные контакты. Выбранные контакты соединяются с гнездом расширения платы B2 (рис.26). Соединим выбранные контакты параллельного порта с гнездом расширения платы B2 (табл.7). Описание разрядов используемых портов представлено в табл. 8.

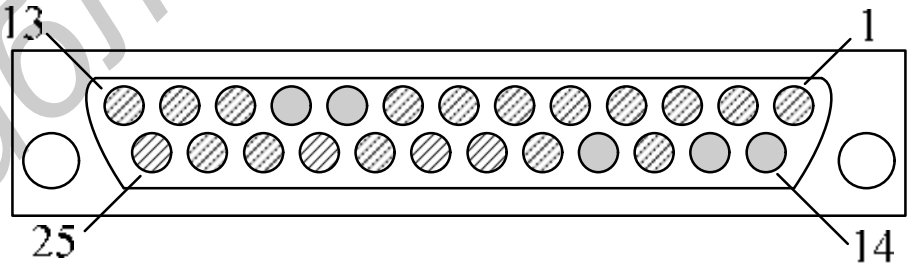


Рис. 25. DB25 Разъем параллельного порта

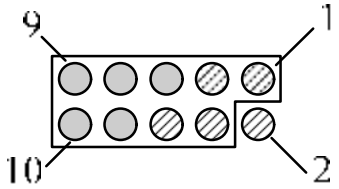


Рис.26. ПЛИС Spartan 2E, разъем B2

Таблица 7

Соответствие контактов плат DB25 и B2

Контакт DB25	Контакт B2	Описание
9	5	Строб, STB
10	7	Входная линия данных,DI0
15	10	Входная линия данных,DI1
14	9	Выходная линия данных,DO0
17	8	Выходная линия данных,DO0

Таблица 8

Разряды используемых LPT-портов

Порт LPT1	7	6	5	4	3	2	1	0
378h	STB	x	x	x	x	x	x	x
379h	x	DI0	x	x	DI1	x	x	x
37Ah	x	x	x	x	$\overline{\text{DO0}}$	x	$\overline{\text{DO1}}$	x

Протокол обмена данными между клиентом (ПК) и сервером (ПЛИС) подразумевает начало взаимодействия после того, как клиент делает запрос на чтение или запись данных. Сервер обрабатывает запрос и выдает запрашиваемые данные либо записывает данные, переданные клиентом.

Данный протокол реализуется на языке VHDL с помощью конечного автомата с тремя состояниями IDLE, WRITE, READ (листинг 8). Переход между состояниями управляется клиентом с помощью импульса *ds*. Данные (2 бита) записываются либо считываются клиентом в регистр *dbuf*. Для наглядности работы текущее состояние автомата и содержимое регистра данных выводятся на индикаторы (LED0–LED3). Временные диаграммы работы автомата приведены на рис.27.

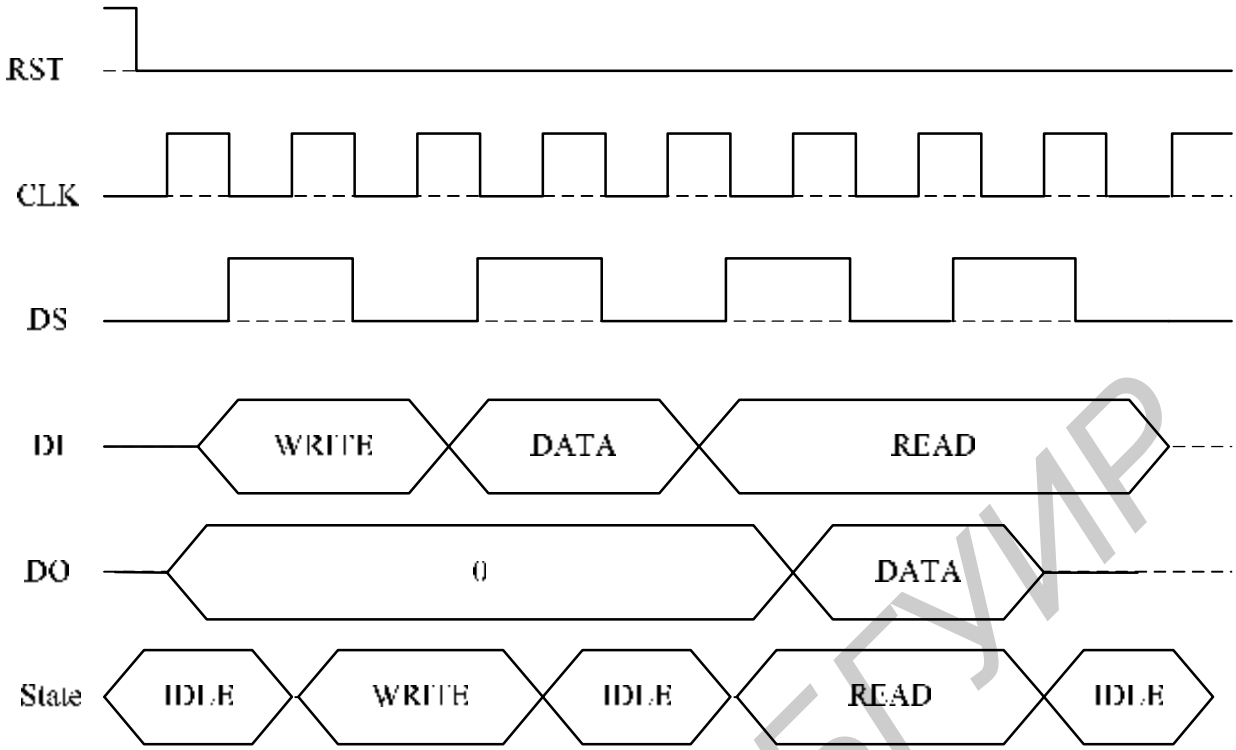


Рис. 27. Временная диаграмма работы устройства

Листинг 8

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity conn is
    port(
        ds: in std_logic; -- короткий импульс синхронизации
        di: in std_logic_vector(1 downto 0); -- вход 2 бита
        do: out std_logic_vector(1 downto 0); -- выход 2 бита

        rst : in std_logic;
        clk : in std_logic;

        l_state  : out std_logic_vector(1 downto 0); -- 11 - ожидание команды, 10
- чтение, 01 - запись
        l_data   : out std_logic_vector(1 downto 0); -- состояние буфера данных
        l_g      : out std_logic -- защелка индикатора

    );
end conn;

architecture Behavioral of conn is
signal dbuf : std_logic_vector(1 downto 0); -- регистр хранения данных
signal prev_ds : std_logic; -- регистр, в котором запоминается значение ds
-- состояния конечного автомата
```

```
type State_type is (IDLE, READ, WRITE);
signal State: State_type;
begin

    State_machine: process (clk, rst, ds)
    begin
        if(rst = '1') then
            dbuf <= (others=>'0');
            do <= (others=>'0');
            prev_ds <= '0';
            State <= IDLE;
        elsif clk'event and clk = '1' then
            -- переход между состояниями синхронизируется импульсом ds
            if(ds = '0') then
                prev_ds <= ds;
            end if;
            case State is
                when IDLE =>
                    if(ds = '1' and prev_ds = '0') then
                        if di="01" then
                            prev_ds <= '1';
                            State <= WRITE;
                        elsif di="10" then
                            prev_ds <= '1';
                            State <= READ;
                        elsif di="11" then
                            do <= "01"; -- ответ проекта
                            State <= IDLE;
                        else
                            prev_ds <= '1';
                            State <= IDLE;
                        end if;
                    end if;
                when READ =>
                    if(ds = '1' and prev_ds = '0') then
                        prev_ds <= '1';
                        State <= IDLE;
                        do <= dbuf;
                    end if;
                when WRITE =>
                    if(ds = '1' and prev_ds = '0') then
                        prev_ds <= '1';
                        State <= IDLE;
                        dbuf <= di;
                    end if;
                when others =>
                    prev_ds <= '1';
                    State <= IDLE;
            end case;
        end if;
    end process;

    -- вывод содержимого регистра данных и текущего состояния автомата на LED.
    l_data <= dbuf;
    l_state<= "01" when State = WRITE else
            "10" when State = READ else
            "11" when State = IDLE else
            "00";
    l_g <= '1';

end Behavioral;
```

При реализации клиентской части (листинг 9) использовалась библиотека WinIo (www.internals.com). Это связано с тем, что операционные системы Windows NT/2000/XP, для которых разрабатывалось клиентское ПО, не позволяют напрямую получать доступ к портам пользовательским программам. Только программы, работающие на уровне ядра ОС (драйверы), могут напрямую обращаться к портам. Библиотека WinIo содержит такой драйвер и интерфейс работы с ним.

Для использования библиотеки WinIo файлы winio.dll, winio.vxd и winio.sys должны быть помещены в один каталог с исполняемым файлом программы-программатора, а файл winio.lib – в каталог с исходным файлом. Кроме того, к проекту необходимо подключить заголовочный файл winio.h.

Для компиляции вызвать *cl.exe testport.cpp /link winio.lib*. Использование WinIo с Visual Basic, Borland C++ Builder, Visual C++ (в проекте, компилируемом в среде разработки) описано в документации (winio.chm).

Для передачи данных серверу клиент передает команду «запись» (*DIO_WriteData(DIO_CMD_WRITE)*), затем записываемые данные (*DIO_WriteData(i%4)*). Для синхронизации клиента и сервера клиент вызывает команду *DIO_Strobe()*.

Прием данных осуществляется по команде «чтение» (*DIO_ReadData(DIO_CMD_WRITE)*), после чего данные доступны для считывания командой *DIO_ReadData()*.

Листинг 9

```
#include "winio.h"

// 378h
// |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
// -----------------------------------------------
// | CTRL|  X  |  X  |  X  |  X  |  X  |  X  |  X  |

// 379h
// |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
// -----------------------------------------------
// |  X  | DI0 |  X  |  X  | DI1 |  X  |  X  |  X  |

// 37Ah
// |  7  |  6  |  5  |  4  |  3      |  2  |      1  |  0  |
// -------------------------------------------------------
// |  X  |  X  |  X  |  X  | not DO0 |  X  | not DO1 |  X  |


// Сообщения об ошибках
#define DIO_OK       0
#define DIO_E_DRIVER 1
#define DIO_E_BOARD  2
#define DIO_E_DESIGN 3

// Задержка
#define DIO_DELAY    50

// Команды
```

```
#define DIO_CMD_READ  1
#define DIO_CMD_WRITE 2

// Команда проверки связи
#define DIO_CMD_PING 3
// Ответ
#define DIO_PONG     2

// Короткий импульс (бит 7, порт 378h)
void DIO_Strobe()
{
        DWORD val;
        GetPortVal(0x378, &val, 4);
        val = 0x80; // установка бита 7
        SetPortVal(0x378, val, 4);
        Sleep(DIO_DELAY);
        val &= ~0x80; // сброс бита 7
        SetPortVal(0x378, val, 4);
}

// запись 2 битов данных (биты 1 и 3, порт 37Ah)
// бит 3 - инверсный
void DIO_WriteData(char data)
{
        DWORD val, tmp;

        GetPortVal(0x37A, &val, 4);

        val &= ~0xA; // сброс битов 1 и 3

        tmp = (((~data) << 3) & 0x8);  // data(1) -> val(3)
        tmp |= (((~data) & 0x2)); // data(0) -> val(1)
        val |= tmp;

        SetPortVal(0x37A, val, 4);
}

// чтение 2 битов данных (биты 6 и 3, порт 379h)
DWORD DIO_ReadData()
{
        DWORD val, tmp;

        GetPortVal(0x379, &val, 4);
        tmp = val >> 6;
        tmp &= 1;
        val >>= 2;
        val &= 2;
        val ^= tmp;

        return val;
}
// инициализация драйвера работы с портами
char DIO_Connect()
{
        if(!InitializeWinIo())
                return DIO_E_DRIVER;

        // проверить связь с платой
        return DIO_Ping();
}

void DIO_Disconnect()
{
```

```
	ShutdownWinIo();
}

char DIO_Ping()
{
	DIO_WriteData(DIO_PING);
	DIO_Strobe();
	if(DIO_ReadData() != DIO_PONG)
		return DIO_E_DESIGN;

	return DIO_OK;
}

int main(int argc, char* argv[])
{
	char result = DIO_Connect();

	if (result == DIO_OK)
	{
		for(int i=0; i<4; i++)
		{
			DIO_WriteData(DIO_CMD_WRITE);
			DIO_Strobe();
			DIO_WriteData(i%4);
			DIO_Strobe();
			DIO_WriteData(DIO_CMD_READ);
			DIO_Strobe();
			DIO_Strobe();
			printf("Read... %X\n", DIO_ReadData());
		}

		DIO_Disconnect();

	}
	else
	{
		printf("Error : %d\n", result);
	}

	return 0;
}
```


3. ОПИСАНИЕ КОМПЛЕКТА МАКЕТНЫХ ПЛАТ D2-SB&DIO4

Комплект макетных плат Digilent D2-SB&DIO4 состоит из двух частей (системная плата Digilent D2-SB и плата для подключения периферийных устройств Digilent DIO4). Комплект представляет собой завершенную платформу для проектирования и разработки программно-аппаратных решений на базе современных перепрограммируемых СБИС (Xilinx Spartan 2E FPGA). На рис.28 показана упрощенная схема взаимодействия компонентов системы при разработке и эксплуатации программно-аппаратных средств.

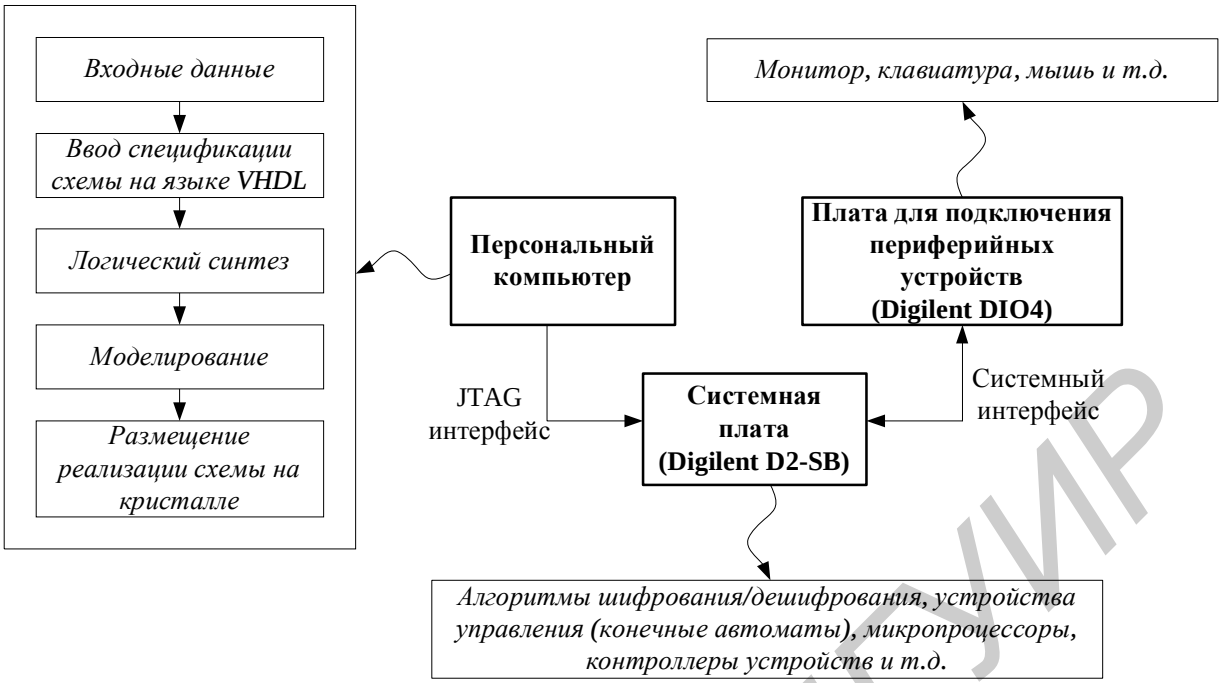


Рис. 28. Схема взаимодействия компонентов системы

Представленная система позволяет разрабатывать и верифицировать программно-аппаратные средства, начиная с момента постановки задачи и заканчивая передачей готового проекта в эксплуатацию. Процесс разработки начинается с описания цифрового устройства на языке VHDL. После описания производится передача VHDL-спецификации проектируемого устройства на этап моделирования. Отлаженная модель-спецификация после проведения логического и технологического синтеза в виде сгенерированного бинарного файла подается через интерфейс JTAG с персонального компьютера на системную плату (Digilent D2-SB) для программирования кристалла «Xilinx Spartan 2E FPGA». После проведения процедуры программирования файла конфигурации JTAG интерфейс может быть отключен, а работа системной платы продолжена в автономном режиме.

3.1. Системная плата Digilent D2-SB

Плата представляет собой минимальную систему для быстрой разработки цифровых систем на основе перепрограммируемых СБИС. Основные характеристики платы:

- Xilinx XC2S200E-200 перепрограммируемая СБИС на 200К вентилей с рабочей частотой 350МГц;
- 143 пользовательских ввода/вывода, разведенных на шесть стандартных 40-пиновых контактов;
- разъем для Flash-памяти, программируемой через JTAG интерфейс;
- генератор тактовых импульсов с частотой 50Мгц;
- JTAG порт;

– дополнительный светодиод и кнопка для контролирования базовых операций ввода/вывода.

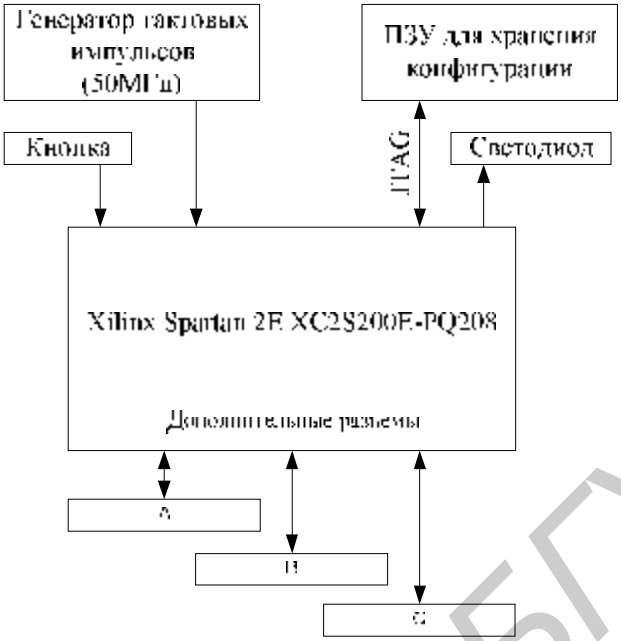


Рис. 29. Функциональная схема платы Digilent D2-SB

Макетная плата Digilent D2-SB (рис. 29) разрабатывалась как системная плата, к которой подключаются модули расширения для взаимосвязи с периферийными устройствами. Для подсоединения дополнительных модулей служат шесть внешних 40-пиновых контактов A1, A2, B1, B2, C1, C2 (рис. 30). К каждому такому контакту кроме линий питания и «земли» подсоединены 32 линии ввода/вывода от ПЛИС.

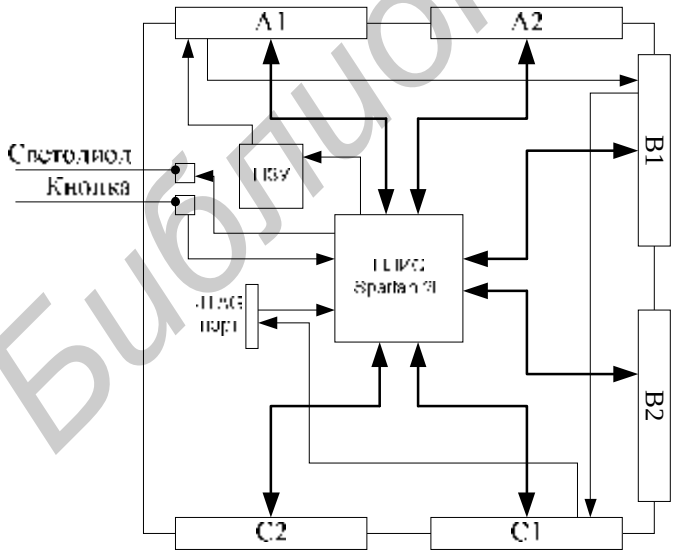


Рис. 30. Структурная схема платы Digilent D2-SB

3.2. Конфигурирование ПЛИС

Конфигурирование ПЛИС, ПЗУ, а также других программируемых устройств, которые могут быть подключены к системной плате через внешние контакты, производится при помощи цепи сканирования порт JTAG (рис. 30). На системной плате цепь сканирования JTAG проходит через перепрограммируемую СБИС, ПЗУ и четыре внешних контакта. Программирование системной платы производится через порт JTAG (расположен на системной плате), который подключен к параллельному порту персонального компьютера (рис. 28).

Конфигурирование ПЛИС системной платы (кроме цепи сканирования JTAG) может производиться при помощи ПЗУ, расположенной на макетной плате. Для этого необходимо установить ПЗУ с данными конфигурации в разъем на плате и установить перемычки в нужное положение.

3.3. Дополнительные светодиод и кнопка

На системной плате расположены светодиод и кнопка для того, чтобы можно было установить текущее состояние системы, а также иметь возможность воздействия на систему, если к плате не подключены дополнительные устройства (рис. 31). Например, состояние светодиода (горит/не горит) может быть установлено путем подачи сигнала от ПЛИС при успешной загрузке проекта на кристалл. Для реализации функции асинхронного сброса может использоваться кнопка, устанавливающая сигнал «reset» в активное состояние в нажатом положении.

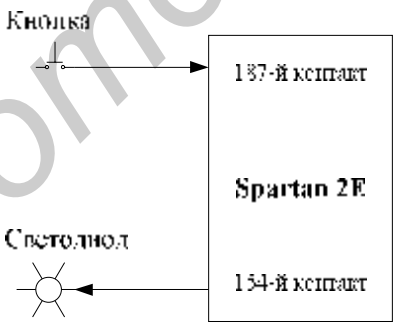


Рис. 31. Подключение кнопки и светодиода к выводам ПЛИС

3.4. Плата для подключения периферийных устройств Digilent DIO4

В комплект макетных плат Digilent D2-SB&DIO4, кроме системной платы, входит дополнительная плата для подключения периферийных устройств (периферийная плата). Основное назначение периферийной платы – обеспечение подключения наиболее распространенных цифровых периферийных устройств к системной плате. Кроме того, периферийная плата предоставляет возможность наблюдения и контроля над процессом работы проекта, загруженного в ПЛИС.

Основные характеристики периферийной платы:

- четыре 7-сегментных индикатора;

- 8 светодиодов;
- 4 кнопки;
- 8 переключателей;
- 3-битный VGA порт;
- PS/2 порт.

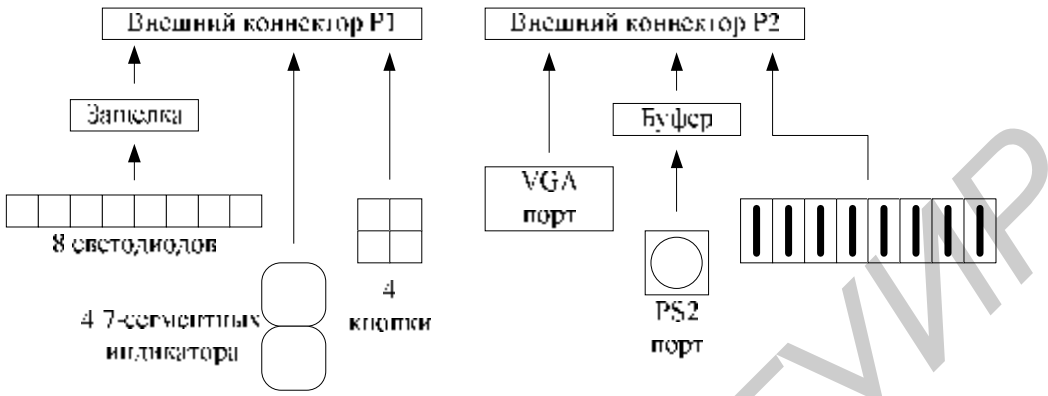


Рис. 32. Функциональная схема периферийной платы Digilent DIO4

3.5. 7-сегментные индикаторы

Периферийная плата Digilent DIO4 (рис.32) имеет четыре 7-сегментных индикатора. Таким образом, например при помощи индикаторов, можно выводить четырехзначные числа или организовать бегущую строку из цифр и некоторых букв.

Все сегменты индикаторов подключены к соответствующим анодам и катодам, на которые можно подавать логические «0» или «1». Все сегменты одного индикатора подключены к одному общему аноду. Все одноименные сегменты четырех индикаторов подключены к общему катоду. Таким образом, для управления свечением индикаторов необходимо четыре анода и семь катодов. Схема соединения сегментов индикаторов с катодами и анодами показана на рис. 33.

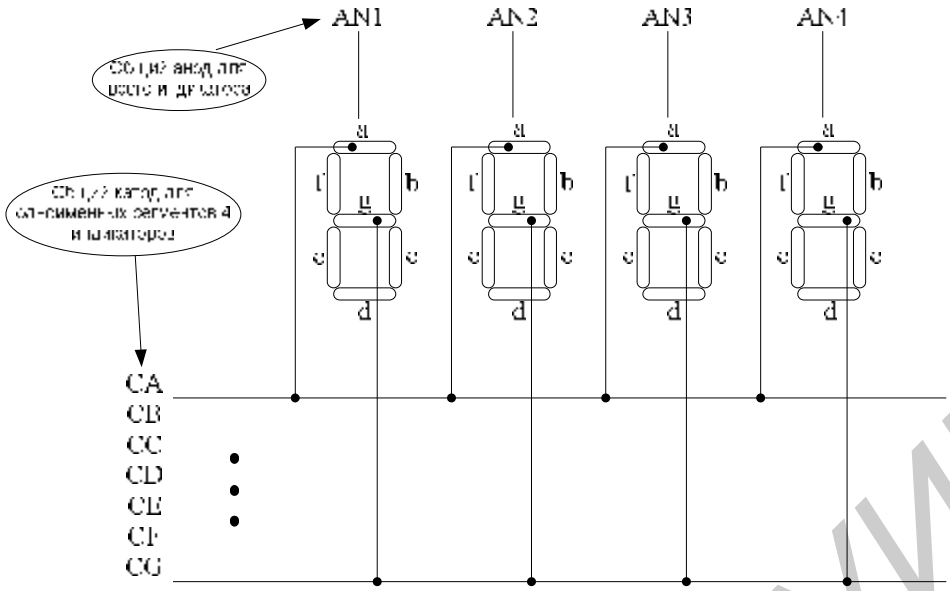
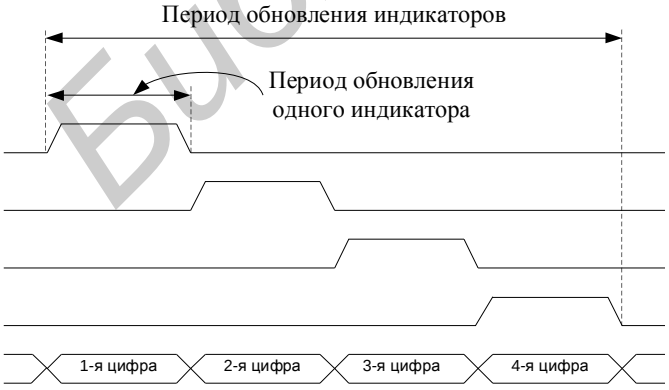


Рис. 33. Схема подключения индикаторов к внешней цепи

При управлении свечением индикаторов следует принимать во внимание тот факт, что сегмент индикатора будет светиться при подаче на его анод и катод логических нулей. То есть если мы хотим, чтобы горел сегмент *a* первого индикатора, мы должны подать логические «0» на линии AN1 и CA.

Как видно из рис. 33, если мы высветим цифру 5 на первом индикаторе и будем подавать логический «0» на линию AN3, то третий индикатор тоже будет высвечивать цифру 5. Соответственно возникает закономерный вопрос, как управлять напряжением анодов и катодов индикаторов для того, чтобы отображать любое четырехзначное число на индикаторах. Для яркого отображения цифр значение каждого индикатора должно обновляться как минимум каждые 16мс, т.е., например, первые 4мс периода обновления на анод и катоды первого индикатора подаются соответствующие значения. В следующие 4мс значения подаются на второй индикатор и т.д. по кругу. На рис. 34,а показана временная диаграмма обновления индикаторов. Соответственно на рисунке 34,б показаны значения сегментов индикаторов для отображения десятичных цифр от 0 до 9.



а

Отображаемая цифра	Значение сигналов на сегментах						
	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0

б

Рис. 34. Принцип работы 7-сегментного индикатора:

а – временная диаграмма процесса обновления индикаторов;
б – значения сегментов индикаторов при отображении цифр

3.6. Светодиоды, кнопки и переключатели

На периферийной плате расположено восемь отдельных светодиодов. Значения управляющих сигналов (LED1, … LED8) подаются на светодиоды от системной платы через защелки, управляемые по уровню сигналом LDG. Таким образом, для того чтобы включить светодиоды, необходимо подать логические «1» на соответствующие входы при установленном в логическую «1» сигнале LDG.

Кроме светодиодов на плате расположены четыре кнопки, которые устанавливают сигналы (BTN#) в логическую «1» в нажатом состоянии и в логический «0» − в отжатом. Также для генерирования логических значений могут использоваться восемь дополнительных переключателей (SW#).

Литература

1. Spartan-IIE 1.8V FPGA Family: Complete Data Sheet.
2. Digilent D2-SB System Board: Reference Manual.
3. Digilent DIIO4 Peripheral Board: Reference Manual.
4. Xilinx Libraries Guide.
5. Xilinx ISE 6 In-Depth Tutorial.
6. Грушвицкий Р.И. Проектирование систем на микросхемах программируемой логики. – СПб.: БХВ-Петербург, 2002. – 608 с.
7. Угрюмов Е.П. Цифровая схемотехника. – СПб.: БХВ-Петербург, 2000. – 528 с.
8. Douglas J Smith. HDL Chip Design. 3ed edition. Doone Publications, Madison, AL, USA. ISBN 0-9651934-3-8. 1997.

Приложение 1 Соответствие контактов системной и периферийной плат

Номер контакта на системной плате	Номер контакта на периферийной плате
1	39
2	40
3	37
4	38
5	35
6	36
7	33
8	34
9	31
10	32
11	29
12	30
13	27
14	28
15	25
16	26
17	23
18	24
19	21
20	22
21	19
22	20
23	17
24	18
25	15
26	16
27	13
28	14
29	11
30	12
31	9
32	10
33	7
34	8
35	5
36	6
37	3
38	4
39	1
40	2

Приложение 2

Соответствие внешних контактов системной платы D2-SB контактам ПЛИС Spartan IIE

№	A1		A2		B1		B2		C1		C2	
	Сигнал	Контакт ПЛИС	Сигнал	Контакт ПЛИС	Сигнал	Контакт ПЛИС	Сигнал	Контакт ПЛИС	Сигнал	Контакт ПЛИС	Сигнал	Контакт ПЛИС
1	GND		GND		GND		GND		GND		GND	
2	VU		VU		VU		VU		VU		VU	
3	VCC33		VCC33		VCC33		VCC33		VCC33		VCC33	
4	ADR0	112	PAI01	162	ADR0	112	PBI01	71	ADR0	112	PCI01	23
5	DB0	111	PAI02	161	DB0	111	PBI02	70	DB0	111	PCI02	22
6	ADR1	110	PAI03	160	ADR1	110	PBI03	69	ADR1	110	PCI03	21
7	DB1	109	PAI04	152	DB1	109	PBI04	68	DB1	109	PCI04	20
8	ADR2	108	PAI05	151	ADR2	108	PBI05	64	ADR2	108	PCI05	18
9	DB2	102	PAI06	150	DB2	102	PBI06	63	DB2	102	PCI06	17
10	ADR3	101	PAI07	149	ADR3	101	PBI07	62	ADR3	101	PCI07	16
11	DB3	100	PAI08	148	DB3	100	PBI08	61	DB3	100	PCI08	15
12	ADR4	99	PAI09	147	ADR4	99	PBI09	60	ADR4	99	PCI09	11
13	DB4	98	PAI010	146	DB4	98	PBI010	59	DB4	98	PCI010	10
14	ADR5	97	PAI011	145	ADR5	97	PBI011	58	ADR5	97	PCI011	9
15	DB5	96	PAI012	141	DB5	96	PBI012	57	DB5	96	PCI012	8
16	WE	95	PAI013	140	WE	95	PBI013	56	WE	95	PCI013	7
17	DB6	94	PAI014	139	DB6	94	PBI014	55	DB6	94	PCI014	6
18	OE	93	PAI015	138	OE	93	PBI015	49	OE	93	PCI015	5
19	DB7	89	PAI016	136	DB7	89	PBI016	48	DB7	89	PCI016	4
20	CSA	181	PAI017	135	CSB	88	PBI017	47	CSC	45	PCI017	3
21	LSBCLK	87	PAI018	134	LSBCLK	87	PBI018	46	LSBCLK	87	PCI018	206
22	MA1 DB0	180	MA2DB0	133	MB1DB0	86			MC1DB0	44	MC2DB0	205
23	MA1DB1	179	MA2DB1	132	MB1DB1	84			MC1DB1	43	MC2DB1	204
24	MA1DB2	178	MA2DB2	129	MB1DB2	83			MC1DB2	42	MC2DB2	203
25	MA1DB3	176	MA2DB3	127	MB1DB3	82			MC1DB3	41	MC2DB3	202
26	MA1DB4	175	MA2DB4	126	MB1DB4	81			MC1DB4	40	MC2DB4	201
27	MA1DB5	174	MA2DB5	125	MB1DB5	75			MC1DB5	36	MC2DB5	200
28	MA1DB6	173	MA2DB6	123	MB1DB6	74			MC1DB6	35	MC2DB6	199
29	MA1DB7	169	MA2DB7	122	MB1DB7	73			MC1DB7	34	MC2DB7	198
30	MA1ASTB	168	MA2ASTB	121					MC1ASTB	33	MC2ASTB	194
31	MA1DSTB	167	MA2DSTB	120					MC1DSTB	31	MC2DSTB	193
32	MA1WRT	166	MA2WRT	116					MC1WRT	30	MC2WRT	192
33	MA1WAIT	165	MA2WAIT	115					MC1WAIT	29	MC2WAIT	191
34	MA1RST	164	MA2RST	114					MC1RST	27	MC2RST	189
35	MA1INT	163	MA2INT	113					MC1INT	24	MC2INT	188
36	JTSELA				JTSELB				JTSELC			
37	TMS				TMS				TMS			
38	TCK				TCK				TCK			
39	TDO		GCLK0	80	TDO				TDO		GCLK1	77
40	TDI		GND		TDI				TDI		GND	

Приложение 3

Назначение контактов ПЛИС

№	Функция	№	Функция	№	Функция	№	Функция
1	2	3	4	5	6	7	8
1	GND	53	VCCO	105	VCCO	157	TDO
2	TMS	54	M2	106	PROG	158	GND
3	PC-IO17	55	PB-IO14	107	INIT	159	TDI
4	PC-IO16	56	PB-IO13	108	ADR2	160	PA-IO3
5	PC-IO15	57	PB-IO12	109	DB1	161	PA-IO2
6	PC-IO14	58	PB-IO11	110	ADR1	162	PA-IO1
7	PC-IO13	59	PB-IO10	111	DB0	163	MA1-INT
8	PC-IO12	60	PB-IO9	112	ADR0	164	MA1-RST
9	PC-IO11	61	PB-IO8	113	MA2-INT	165	MA1-WAIT
10	PC-IO10	62	PB-IO7	114	MA2-RST	166	MA1-WRT
11	PC-IO9	63	PB-IO6	115	MA2-WAIT	167	MA1 -DSTB
12	GND	64	PB-IO5	116	MA2-WRT	168	MA1 -ASTB
13	VCCO	65	GND	117	GND	169	MA1-DB7
14	VCCINTT	66	VCCO	118	VCCO	170	GND
15	PC-IO8	67	VCCINT	119	VCCINT	171	VCCO
16	PC-IO7	68	PB-IO4	120	MA2-DSTB	172	VCCINT
17	PC-IO6	69	PB-IO3	121	MA2-ASTB	173	MA1-DB6
18	PC-IO5	70	PB-IO2	122	MA2-DB7	174	MA1-DB5
19	GND	71	PB-IO1	123	MA2-DB6	175	MA1-DB4
20	PC-IO4	72	GND	124	GND	176	MA1-DB3
21	PC-IO3	73	MB1-DB7	125	MA2-DB5	177	GND
22	PC-IO2	74	MB1-DB6	126	MA2-DB4	178	MA1-DB2
23	PC-IO1	75	MB1-DB5	127	MA2-DB3	179	MA1-DB1
24	MC1-INT	76	VCCINT	128	VCCINT	180	MA1-DB0
25	GND	77	GCLK1	129	MA2-DB2	181	CSA
26	VCCO	78	VCCO	130	VCCO	182	GCLK2
27	MC1-RST	79	GND	131	GND	183	GND
28	VCCINT	80	GCLK0	132	MA2-DB1	184	VCCO
29	MC1-WAIT	81	MB1-DB4	133	MA2-DB2	185	GCLK3
30	MC1-WRT	82	MB1-DB3	134	PA-IO18	186	VCCINT
31	MC1-DSTB	83	MB1-DB2	135	PA-IO17	187	BTN
32	GND	84	MB1-DB1	136	PA-IO16	188	MC2-INT
33	MC1-ASTB	85	GND	137	GND	189	MC2-RST
34	MC1-DB7	86	MB1-DB0	138	PA-IO15	190	GND
35	MC1-DB6	87	LSBCLK	139	PA-IO14	191	MC2-WAIT
36	MC1-DB5	88	CSB	140	PA-IO13	192	MC2-WRT
37	VCCINT	89	DB7	141	PA-IO12	193	MC2-DSTB
38	VCCO	90	VCCINT	142	VCCINT	194	MC2-ASTB

Окончание прил. 3

1	2	3	4	5	6	7	8
39	GND	91	VCCO	143	VCCO	195	VCCINT
40	MC1-DB4	92	GND	144	GND	196	VCCO
41	MC1-DB3	93	OE	145	PA-IO11	197	GND
42	MC1-DB2	94	DB6	146	PA-IO10	198	MC2-DB7
43	MC1-DB1	95	WE	147	PA-IO9	199	MC2-DB6
44	MC1-DB0	96	DB5	148	PA-IO8	200	MC2-DB5
45	CSC	97	ADR5	149	PA-IO7	201	MC2-DB4
46	PB-IO18	98	DB4	150	PA-IO6	202	MC2-DB3
47	PB-IO17	99	ADR4	151	PA-IO5	203	MC2-DB2
48	PB-IO16	100	DB3	152	PA-IO4	204	MC2-DB1
49	PB-IO15	101	ADR3	153	DIN	205	MC2-DB0
50	M1	102	DB2	154	LED	206	PC-IO18
51	GND	103	GND	155	CCLK	207	TCK
52	M0	104	DONE	156	VCCO	208	VCCO

Приложение 4

Назначение внешних контактов платы расширения DIO4

P1	Сигнал	Направление	P2	Сигнал	Направление
1	nc		1	nc	
2	nc		2	nc	
3	nc		3	nc	
4	nc		4	nc	
5	nc		5	nc	
6	nc		6	nc	
7	nc		7	nc	
8	nc		8	nc	
9	nc		9	nc	
10	nc		10	nc	
11	nc		11	nc	
12	nc		12	nc	
13	AN3	in	13	vs	in
14	AN4	in	14	HS	in
15	AN1	in	15	GRN	in
16	AN2	in	16	RED	in
17	BTN4	out	17	PS2D	bidi
18	BTN5	out	18	BLU	in
19	nc		19	BTN2	out
20	BTN3	out	20	PS2C	bidi
21	LED8	in	21	DP	in
22	LEDG	in	22	BTN1	out
23	LED7	in	23	CG	in
24	nc		24	SW8	out
25	LED6	in	25	CF	in
26	nc		26	SW7	out
27	LED5	in	27	CE	in
28	nc		28	SW6	out
29	LED4	in	29	CD	in
30	nc		30	SW5	out
31	LED3	in	31	CC	in
32	nc		32	SW4	out
33	LED2	in	33	CB	in
34	nc		34	SW3	out
35	LED1	in	35	CA	in
36	nc		36	SW2	out
37	VCC33		37	VCC33	
38	nc		38	SW1	
39	GND		39	GND	
40	VU		40	VU	

Св. план 2004, резерв

Учебное издание

Иванюк Александр Александрович,
Занкович Артем Петрович,
Петроненко Денис Сергеевич,
Мусин Сергей Борисович

Проектирование аппаратно-программных вычислительных средств

Методическое пособие
для студентов специальности
«Программное обеспечение информационных технологий»
дневной и дистанционной форм обучения

Редактор Н.В. Гриневич
Корректор Е.Н. Батурчик
Компьютерная верстка М.В. Шишло

Подписано в печать 09.12.2004. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 3,6.
Уч.-изд. л. 3,3. Тираж 100 экз. Заказ 582.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия на осуществление издательской деятельности №02330/0056964 от 01.04.2004.
Лицензия на осуществление полиграфической деятельности №02330/0133108 от 30.04.2004.
220013, Минск, П. Бровки, 6