JTAG-тестирование

(часть 4)

Иосиф Каршенбойм (Санкт-Петербург)

Данная статья представляет собой обзор материалов, посвященных тестированию плат и устройств по интерфейсу JTAG. В третьей части статьи продолжается рассмотрение команд тестирования, выдаваемых по интерфейсу JTAG, и структуры BSDL-файла, а также даётся описание «привязки» битов регистра сканирования к выводам микросхемы.

Описание шин и сигналов на выводах микроконтроллера BlackFin

Рассмотрим назначение выводов микроконтроллера BlackFin BF537. Как видно из табл. 8, микропроцессор имеет множество шин и одиночных выводов, которым соответствуют буферы с различной нагрузочной способностью. Более подробную информацию об этом можно получить непосредственно из технического описания на микроконтроллер. Разумеется, при составлении тестов необходимо учитывать нагрузочную способность выходных буферов микросхем.

Разделим выводы микроконтроллера на несколько групп: «только вход», «только выход», «выход с третьим состоянием», «двунаправленная шина».

Сначала посмотрим, как описываются выводы «только вход». Фрагмент файла – ячейка 227 – описывает часть регистра, работающую с выводом сигнала TEST:

```
--num cell port function safe
(ccell disval rslt )
" 227 ( BC_2 , TEST,
        input , X ) , « &
```

Вывод сигнала TEST однонаправленный и используется только как вход. Для осуществления этой функции задействована ячейка BC_2. Безопасное состояние вывода не определено – «X».

Посмотрим, как описываются выводы сигналов «только выход». Для примера возьмём два сигнала BG_B и BGH_B. Фрагмент файла, приведённый ниже, описывает часть регистра, работающего с сигналами BG_B и BGH_B, а именно ячейки 0 и 1:

```
--num cell port function safe
(ccell disval rslt)
" 0 ( BC_1 , BG_B , output2
, X ) , « &
" 1 ( BC_1 , BGH_B , output2
, X ) , « &
```

Выводы сигналов BG_B и BGH_B однонаправленные и используются только как выход. Для таких сигналов задействованы ячейки BC_1. Безопасное состояние выводов не определено – «X». Поскольку для выводов BG_B и BGH_B не предусмотрено третьего состояния, то и никакие дополнительные сигналы управления им не требуются.

Теперь посмотрим, как описываются «выходы с третьим состоянием». Для примера возьмём фрагмент файла – ячейки 15...17, где описывается часть регистра, работающего с шиной адреса:

```
--num cell port function safe
(ccell disval rslt)
" 15 ( BC_1 , ADDR(6),
output3 , X , 17 ,
        0 , Z ) , « &
" 16 ( BC_1 , ADDR(5),
 output3 , X , 17,
        0 , Z ) , « &
" 17 ( BC_1 , * ,
control , 0 ) , « &
```

Шина адреса однонаправленная и переключается в третье состояние одним сигналом управления. Для этого в каждом разряде регистра используются ячейки только одного типа – BC_1. Ячейки типа BC_1 используют и для вывода сигналов адреса, и для записи в регистр сигнала управления. Ячейки BC_1 выполняют функцию output3 или функцию control. Для шины адреса безопасное состояние не определено – «X». Для ячеек типа BC_1 сигнал управления берется из ячейки, соответствующей 17-му биту регистра, причём состояние запрета определяет значение «0», и вся шина при этом будет находиться в состоянии «Z».

Теперь рассмотрим, как описаны ячейки, работающие с портами флагов. Порты флагов имеют независимое управление, причём выводы этих портов могут работать как на выход, так и на вход:

```
--num cell port function safe
(ccell disval rslt)
" 47 ( BC_2 , , input
, X ) , « &
" 48 ( BC_1 , PJ2 , output3
, X , 49 , 0
         , Z ) , « &
" 49 ( BC_1 , * , control
, 0 ) , « &
```

Этот фрагмент файла представляет собой ячейки 47...49, где описывается часть регистра, работающего с портом PJ. Порт PJ – набор двунаправленных сигналов с независимым для каждого сигнала управлением. Каждый вывод порта использует ячейки BC_1 и BC_2. Ячейки типа BC_1 используются для вывода сигналов данных и для записи в регистр сигнала управления. Ячейки типа BC_2 используются для ввода сигналов данных. Соответственно ячейки BC_1 выполняют функцию output3 или функцию control, а ячейки BC_2 – функцию input. Для вывода флага PJ2 безопасное состояние не определено – «X», и выходная ячейка типа BC_1 берёт сигнал управления из ячейки, соответствующей 49-му биту регистра, причём состоянию запрета соответствует значение «0», а сигнал PJ2 при этом будет находиться в состоянии «Z».

Аналогично описывается и двунаправленная шина данных. Фрагмент файла – ячейки 256...260, описывает часть регистра, работающую с шиной данных:

```
--num cell port function safe
(ccell disval rslt )
" 256 ( BC_2 , DATA(1) ,
input , X ) , « &
" 257 ( BC_1 , DATA(1) ,
output3 , X , 260 ,
       0 , Z ) , « &
" 258 ( BC_2 , DATA(0) ,
input , X ) , « &
" 259 ( BC_1 , DATA(0) ,
output3 , X , 260 ,
       0 , Z ) , « &
" 260 ( BC_1 , * , control
, 0 ) «;
```

Шина данных двунаправленная и, в отличие от порта флагов с независимым управлением, переключается только одним сигналом управления. Для передачи информации в обе стороны в каждом разряде данных используются ячейки BC_1 и BC_2. Ячейки типа BC_1 используют для вывода сигналов данных и для записи в регистр сигнала управления. Ячейки типа BC_2 используют для ввода сигналов данных. Соответственно ячейки BC_1 выполняют функцию output3 или функцию control, а ячейки BC_2 – функцию input. Для шины данных безопасное состояние не определено – «X». Для ячеек типа BC_1 сигнал управления берется из ячейки, соответствующей 260-му биту регистра, причём состоянию запрета соответствует значение «0», и вся шина при этом будет находиться в состоянии «Z».

Подведём итог

Вот общее количество выводов: 122 входа и выхода, + 2 выхода типа buffer и 56 выводов linkage. Из 122 выводов 5 выводов заняты непосредственно портом JTAG.

Из 124 выводов, которые могут быть протестированы, имеем: 35 выходов + 2 выхода типа buffer + 8 входов + 94 двунаправленных вывода.

Непосредственно не проверяются следующие выводы: EMU_B, CLKIN, XTAL, CLKBUF, RTXI, RTXO.

Также не проверяются выводы, по которым подводится питание: VDD_INT, VDD_EXT, VDD_RTC, VROUT и GND.

Тестами можно охватить только 68% от общего числа выводов. Остальные выводы могут быть оценены только косвенно.

И ещё раз JTAG в картинках

Параграф о картинках уже был в предыдущих частях статьи, так зачем же мы к нему возвращаемся снова? Возвращаемся потому, что теперь мы вооружены новыми знаниями и можем адекватно оценить предлагаемые «картинки». К настоящему моменту мы уже знаем, как работает узел TAP-контроллера, знаем, как именно микросхемы соединяются в цепочку, знаем, куда и какой бит нужно посылать. А главное, мы знаем, какой эффект получим от автоматического тестирования платы, особенно

Таблица 8. Описание шин и выводов микроконтроллера BlackFin BF537

Название	Тип	Описание
Интерфейс памяти		
ADDR19-1	O	Шина адреса для асинхронного доступа к памяти
DATA15-0	I/O	Шина данных для синхронного и асинхронного доступа к памяти
ABE1-0/SDQM1-0	O	Сигналы разрешения выбора байта для синхронного и асинхронного доступа к памяти
BR	I	Запрос на доступ к шине
BG	O	Разрешение на доступ к шине
BGH	O	Выход ожидания разрешения на доступ к шине
Сигналы управления асинхронной памяти – Asynchronous Memory Control		
AMS3-0	O	Сигнал выбора банка
ARDY	I	Аппаратный сигнал готовности
AOE	O	Разрешение выхода
ARE	O	Разрешение чтения
AWE	O	Разрешение записи
Сигналы управления синхронной памяти – Synchronous Memory Control		
SRAS	O	Строб адреса строки
SCAS	O	Строб адреса колонки
SWE	O	Разрешение записи
SCKE	O	Разрешение тактовой частоты
CLKOUT	O	Выход тактовой частоты
SA10	O	Вывод A10
SMS	O	Выбор банка
Порты		
PF0..15	I/O	Port F: GPIO/UART1-0/Timer7-0/SPI/External DMA Request
PG0..15	I/O	Port G: GPIO/PPI/SPORT1
PH0..15	I/O	Port H: GPIO/10/100 Ethernet MAC (On ADSP-BF534, these pins are GPIO only)
PJ0..15	*	Port J: SPORT0/TWI/SPISelect/CAN
Часы реального времени – Real Time Clock		
RTXI	I	Вход резонатора для RTC
RTXO	O	Выход резонатора для RTC
Порт JTAG		
TCK	I	Синхрочастота JTAG
TDO	O	Выход данных JTAG
TDI	I	Вход данных JTAG
TMS	I	Сигнал Mode Select JTAG
TRST	I	Сигнал Reset JTAG
EMU	O	Сигнал Emulation Output
Тактирование		
CLKIN	I	Вход Clock/Crystal Input
XTAL	O	Выход Crystal Output
CLKBUF	O	Выход Buffered XTAL Output
Управление режимом работы		
RESET	I	Сброс
NMI	I	Немаскируемое прерывание
BMODE2-0	I	Сигналы управления загрузкой Boot Mode Strap 2-0
Регулятор напряжения		
VROUT0	O	Вход управления для FET
VROUT1	O	Вход управления для FET
Питание		
VDDEXT	P	Питание I/O
VDDINT	P	Внутреннее питание (регулируется от 2.25 В до 3.6 В)
VDDRTC	P	Питание для часов реального времени
GND	G	Земля

I – вход *O – выход* *P – питание* *G – земля*

Рис. 16. Фрагмент современной печатной платы с установленными на ней микросхемами

если эта плата похожа на ту, которая показана на рис. 16.

На рис. 17 приведён фрагмент программы Boundary Scan Coach, о которой мы уже упоминали. Напомним, что функциональная часть микросхемы – Core Logic – показана коричневым цветом. JTAG-регистры – зелёным цветом, выводы микросхемы – серым цветом. Голубым цветом выделен контроллер JTAG-порта. Красным цветом показан путь прохождения данных по регистру граничного сканирования. Программа Boundary Scan Coach содержит несколько частей. Наибольший интерес, с точки зрения граничного сканирования, представляет собой часть № 3 этой программы, в которой описано, как производится сканирование устройства, содержащего две микросхемы. По цепи JTAG-тестирования микросхемы включены последовательно. Данные сдвигаются из порта в первую микросхему, а из неё во вторую. Выходные данные принимаются портом. Пользователь может задавать различные режимы работы выводов, например, подавать воздействия с одной микросхемы на другую. В режиме линзы можно посмотреть состояние выводов так, как это показано на рис. 15, при описании структуры ячеек ввода-вывода.

От теории к практике!

Теперь остаётся удостовериться, что всё написанное в этой статье применимо на практике. Как проверить, что считывается на входе микросхемы? Как выдать необходимые сигналы на вывод? Для этого предлагается три пути.

Путь № 1, или классический, – быстро садимся и учим языки, при помощи которых описываются тестовые воздействия. Потом неизвестно где, находим РАБОТАЮЩИЕ компиляторы этих языков. Далее понятно... С этим вариантом читатель сможет разобраться сам, если имеет достаточно времени и сил.

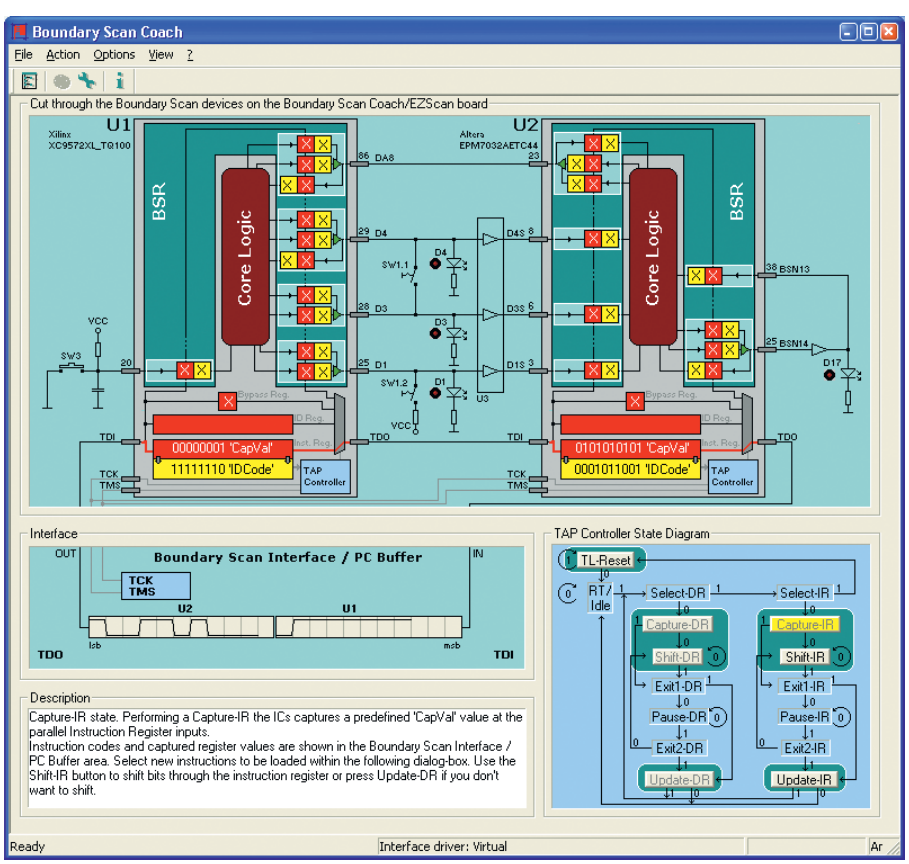

Рис. 17. Окно программы Boundary Scan Coach. Часть № 3

Путь № 2, или обычный, – с помощью C++ пишем тест сами. Например, так, как в [11, 12] или в [18].

Путь № 3, необычный, – ищем в Сети и устанавливаем Universal Scan [5].

Рассмотрим два последних пути решения проблемы. А какой из них лучше – об этом судить читателю.

Путь № 2 – переходим к симуляции аппаратной части интерфейса JTAG

До сих пор большинство программных инструментов и обучающих программ были ориентированы только на обучение. Давайте ознакомимся с теми программами и инструментами, которые могут пригодиться в реальной жизни при работе с «железом».

Поскольку уже описана работа с интерфейсом, познакомимся с ещё одним программным симулятором его работы. На этот раз речь пойдёт о симуляции работы порта и TAP-контроллера в ModelSim. Читатели могут взять этот проект с сайта автора, на странице, посвящённой JTAG. Для чего делался этот проект? Дело в том, что изготовители FPGA постепенно «разворачиваются» навстречу пользователю микросхем. В острой конкурентной борьбе им приходится давать потребителю всё больше и больше сервиса. Когда-то встроенный логический анализатор был делом любителей-одиночек. Но, как выяснилось, инструмент этот оказался очень полезен, особенно для больших проектов и для «толстых» FPGA. А чего только не сделает производитель, чтобы продать микросхем побольше да подороже. Вот так и пришлось изготовителям вводить в состав своих программных средств встроенные логические анализаторы. Естественно, что для связи с анализаторами был применён интерфейс JTAG. Но при этом всё, что было с ним связано, поставлялось в зашифрованном виде. Однако «почин был подхвачен» массами пользователей (абсолютно ясно, как именно был подхвачен), и фирмам пришлось раскрыть адресацию и сами компоненты, при помощи которых встроенные логические анализаторы подключались к проекту пользователя. Так что теперь у разработчиков FPGA есть возможность использовать знания технологии JTAG и для отладки проектов в FPGA. Ну, а там где FPGA, там и ModelSim, и Verilog.

Итак, представим себе хост, JTAG-порт и связь с проектом пользователя.

Теперь рассмотрим рис. 18, где эти компоненты представлены в виде блок-схем.

Начнём с самого понятного разработчику – а именно с той части, которая выделена голубым цветом и представляет собой логику отладочного блока пользователя, которая находится в FPGA, там же, где и проект пользователя. С этой логикой в общих чертах всё понятно. Её можно «написать» самому, отсимулировать и затем скомпилировать. А вот JTAG-порт – это аппаратная часть микросхемы, и детального описания его работы нет. Ну а хост – это немного программ и аппаратный LPT- или USB-порт. А ведь инженер-аппаратчик привык видеть всё в виде осциллограмм или их симуляций. Вот поэтому автор предлагает воспользоваться одним небольшим проектом, сделанным им для статьи. Это симуляция описанных выше устройств, выполненная в среде ModelSim и написанная на языке Verilog. Цель проекта – помочь читателю увидеть последовательность смены состояний автомата TAP-контроллера в виде осциллограмм.

Основой проекта является именно узел TAP-контроллера. Узел TAP-контроллера вместе с регистрами и другими дополнительными узлами образуют модель JTAG-порта. Для выдачи в JTAG-порт последовательности импульсов в проекте используется блок tester. Этот узел выполнен как task.

Чтобы сформировать последовательность импульсов, выдаваемых на узел tester, можно пройтись по диаграмме автомата в «ручном режиме». Кроме этого, для того чтобы упростить формирование последовательностей импульсов, выдаваемых узлом tester, можно воспользоваться ещё одним программным инструментом, также доступным на сайте автора. Этот программный инструмент как раз и является примером решения задачи путём № 2. Программа специально написана автором на BCB6 для учебных проектов к статьям. Исходные тексты программы доступны на сайте автора. Одним из результатов работы этого программного инструмента как раз и будут строки для тестовой последовательности импульсов.

Поскольку программа обучающая, то она работает по сокращённой диаграмме переходов. Для упрощения

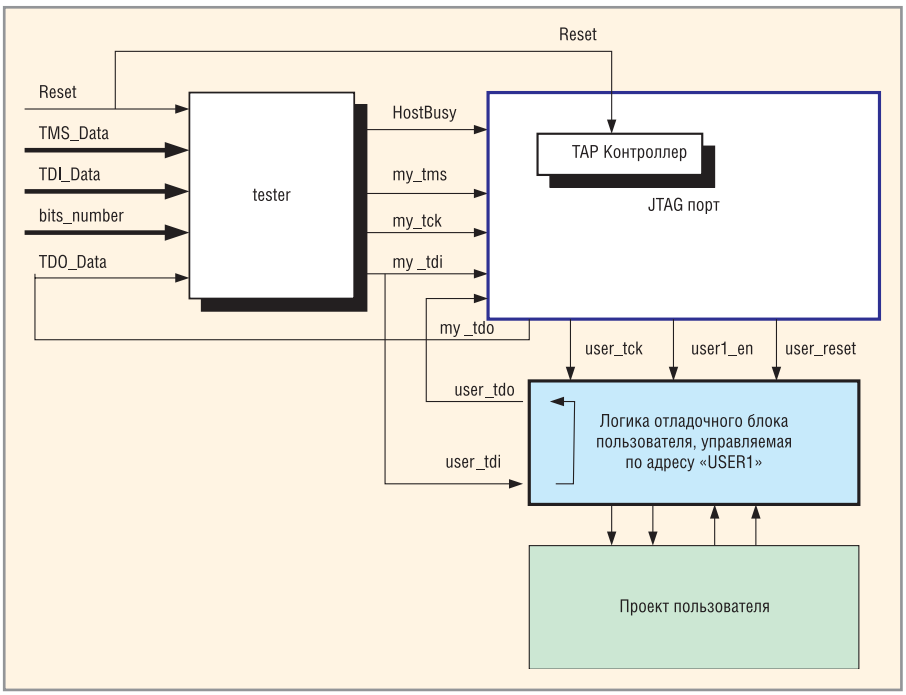

Рис. 18. Блок-схема проекта симуляции работы хоста, JTAG-порта и логики пользователя, выполненная в среде ModelSim и написанная на языке Verilog

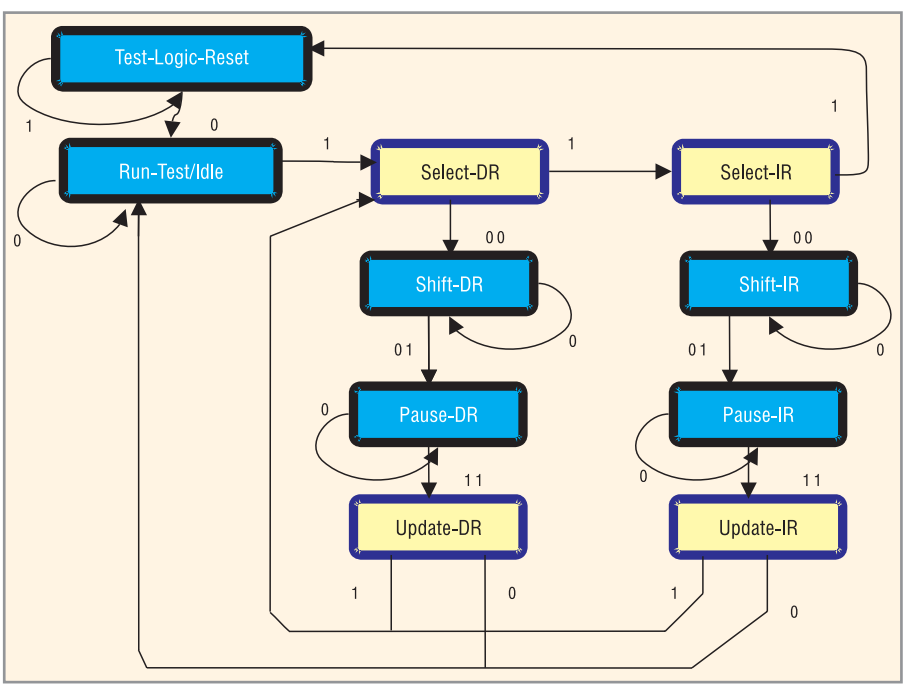

Рис. 19. Упрощённая диаграмма переходов

программы, работающей с JTAG-портом, было сокращено число состояний автомата, которыми можно пользоваться. Поэтому у нас есть возможность представить диаграмму более компактной. Такая диаграмма переходов будет выглядеть так, как показано на рис. 19. Именно этой диаграммой переходов и пользовался автор, когда разрабатывал свой программный инструмент для работы с JTAG-портом от LPT-порта хоста. Более подробно об этом можно прочитать в [18].

На рис. 20 показано основное окно программы, в котором пользователь формирует исполняемые тесты. В левой части окна расположена таблица, в которую помещаются команды и их параметры, а в правой части – органы управления, необходимые для работы с этим окном. Тесты можно исполнять как в шаговом, так и в автоматическом режиме – по таймеру. Для работы в автоматическом режиме предусмотрена возможность останова по точкам, задаваемым пользователем. Более подроб-

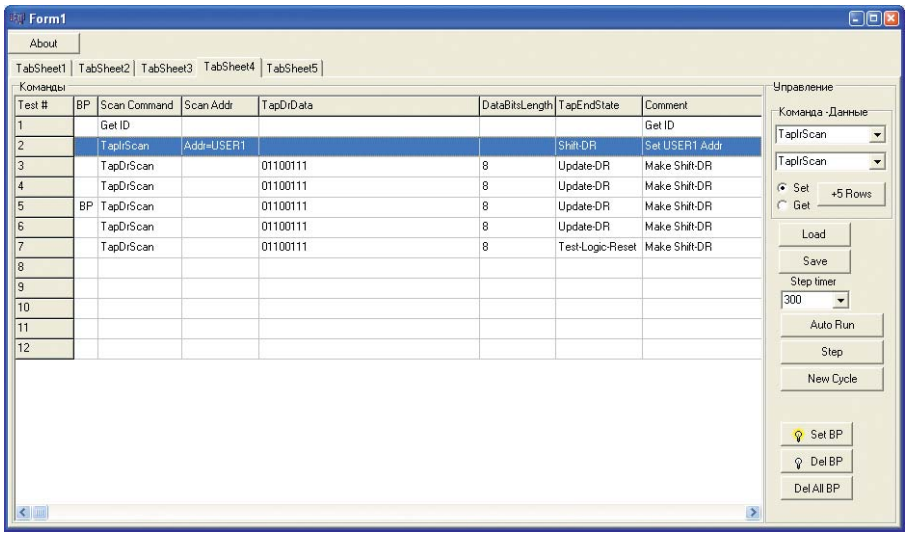

Рис. 20. Окно программы, в котором пользователь формирует исполняемые тесты

но об этой программе можно прочитать в [18].

Загрузим или сформируем тест чтения ID из микросхемы, установим мониторинг «на уровне битов» и запустим тест на выполнение. Трассировка теста распечатается в окне мониторинга даже при выключенном аппаратном адаптере.

Именно эту трассировку можно использовать при задании параметров для task. Ниже приведён фрагмент кода для симуляции режима чтения ID из микросхемы. Закомментированные строки, выделенные зеленым цветом, – это часть трассировки, взятая как результат работы программного инструмента. Строки, напечатанные чёрным цветом, – это то, что будет выполнять симулятор. Для упрощения чтение ID выполняется не за 32 такта, а только за восемь.

```
// Get ID
//переход в TAP_TLR, tms - 0x1F,
tdi - 0x00, N - 0x05
tester(32'h00FF, 32'h0000, 8,
wTDO, RST); - это сделано,
чтобы увидеть
     сброс «user_reset"

//переход в TAP_SI, tms - 0x06,
tdi - 0x00, N - 0x05
tester(32'h0006, 32'h0000, 5,
wTDO, RST);

//запись команды в последнем
байте, tms - 0x00, tdi - 0x09,
N - 0x05, RdWriteTMS_SetTDI
strScanIn - 10101
tester(32'h0000, 32'h0009, 5,
wTDO, RST);

//запись последнего бита команды
и переход в TapEndState,
tms - 0x0B, tdi - 0x00,
N - 0x05, RdWriteTMS_SetTDI
strScanIn - 11111
tester(32'h000B, 32'h0000, 5,
wTDO, RST);

//cur_state - 0x03
//по переходам - 0x02
//String IrScan - 110101
//прием данных в байтах,
tms - 0x00, tdi - 0x00,
     N - 0x08, RdWriteTMS_SetTDI
     strScanIn - 00100111
tester(32'h0000, 32'h0000, 8,
wTDO, RST);
//прием данных в байтах,
tms - 0x00, tdi - 0x00,
     N - 0x08, RdWriteTMS_SetTDI
     strScanIn - 00000001
//прием данных в байтах,
tms - 0x00, tdi - 0x00,
     N - 0x08, RdWriteTMS_SetTDI
     strScanIn - 10000010
//прием данных в байтах,
tms - 0x00, tdi - 0x00,
     N - 0x08, RdWriteTMS_SetTDI
     strScanIn - 00000011
//прием последнего бита
 и переход в TapEndState,
     tms - 0x03, tdi - 0x00,
     N - 0x03, RdWriteTMS_SetTDI
     strScanIn - 110
tester(32'h0003, 32'h0000, 3,
wTDO, RST);
//cur_state - 0x08
//по переходам - 0x01
```

Теперь более подробно можно рассмотреть структуру самого JTAG-порта, блок-схема которого приведена на рис. 21.

В состав порта входят TAP-контроллер, регистр инструкций, декодер адреса инструкций, регистры Bypass, IDCODE, BoundaryScan и другие регистры. Также в состав порта входят два мультиплексора выходных данных и вспомогательный узел, формирующий сброс. На регистры и TAP-контроллер поступают тактовые импульсы TCK. Кроме того, на TAP-контроллер поступает сигнал TMS. Входные данные – TDI поступают на регистры.

Если сигнал TMS удерживать на входе более пяти тактов синхрочастоты, то TAP-контроллер выполнит сброс своей логики и будет находиться в состоянии Test Logic Reset. Импульсы TCK и сигнал TMS переведут TAP-контроллер в другие состояния. При получении инструкции в состоянии Shift_IR, TAP-контроллер пройдёт через состояние Update_IR. Во время сдвига инструкции из JTAG-порта с выхода TDO будут считываться данные. При этом полученная инструкция будет запомнена и её код поступит на дешифратор инструкций. Дешифратор инструкций выберет, какой из регистров скоммутируется на выход TDO. В этот же регистр загрузятся данные при работе TAP-контроллера в состоянии Shift_DR. Результат работы симулятора при чтении ID представлен на рис. 22. Жёлтыми цифрами в верхней части рисунка обозначены периоды времени, соответствующие task'ам, которые в вышеприведённом листинге показаны чёрным цветом. Жёлтым шрифтом на поле симуляции обозначены состояния TAP-контроллера. Последовательность переходов из одного состояния в другое показана жёлтыми стрелками.

Рассмотрим фазы сдвига инструкции, которые на рис. 22 показаны голубыми цифрами. Импульсы, отмеченные цифрами 1 – 5, соответствуют переходам в состояние Shift_IR. Само состояние Shift_IR обозначено цифрой 6. По окончании этого состояния автомат переходит через состояния 6 в состояния, обозначенные как 7 – 11 к состоянию Shift_DR. Состояние Shift_DR обозначено как состояние 12. По окончании работы с данными происходит переход автомата через состояние 13 в состояние Run_Test/Idle.

Голубой цифрой 14 показано формирование импульса СБРОС, кото-

рый вырабатывает счётчик числа импульсов на входе TMS. При достижении заданного уровня, а именно 5 импульсов, счётчик останавливается и выдаёт сигнал СБРОС. Этот сигнал используется логикой сопряжения JTAG-порта с логикой пользователя.

Голубой цифрой 15 показано формирование импульса разрешения выборки, который вырабатывается в соответствии с тем адресом, который был загружен в TAP-контроллер по команде IR-Scan. Часть этих сигналов также используются логикой сопряжения JTAG-порта с логикой пользователя.

Для упрощения симуляции предлагаемый автором проект не содержит «логики пользователя». К «логике пользователя» автор относит те сдвиговые регистры и регистры хранения информации, которые необходимы пользователю для работы его отладочного узла. Кроме собственно узлов, преобразующих последовательный интерфейс в параллельный, пользователю ещё понадобятся дополнительные узлы привязки асинхронного (с точки зрения системной синхрочастоты в проекте пользователя) проекта пользователя к системной синхрочастоте, на которой работает весь остальной проект пользователя.

Ну и последнее, что необходимо сказать об этой диаграмме, так это то, что именно она и есть ответ на «JTAG-ребус», потому что на диаграмме «JTAG-ребус» также показано, как производилось чтение регистра ID из микросхемы.

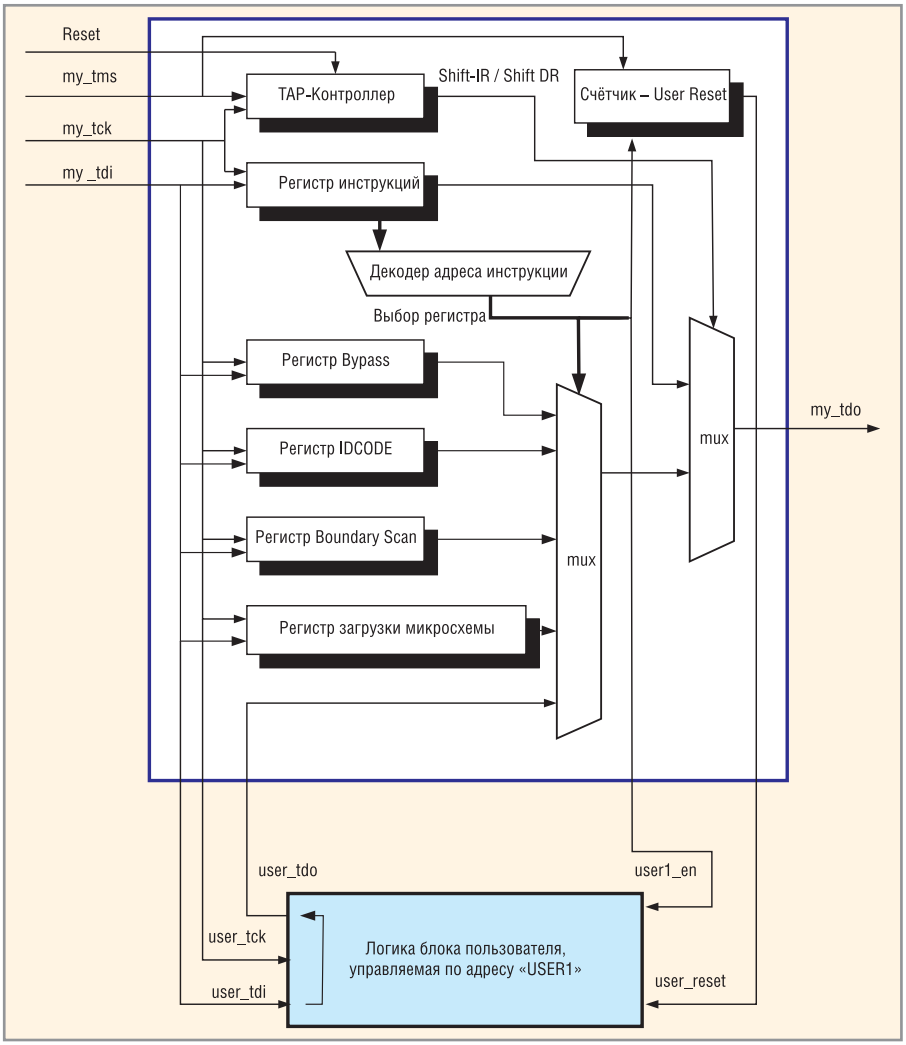

Рис. 21. Блок-схема JTAG-порта и логики пользователя

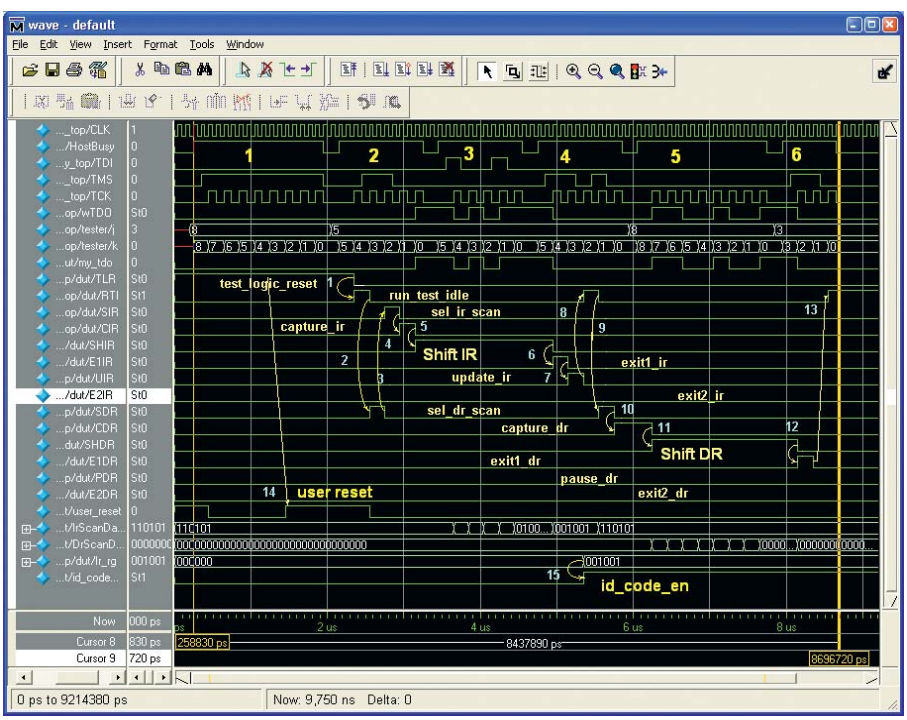

Рис. 22. Симуляция работы JTAG-порта при чтении ID из микросхемы

Путь №3 – программа Universal Scan

Программа Universal Scan позволяет сканировать устройство пользователя, управлять выходами и читать информацию на входах, а также программировать Flash-память.

Чтобы начать тестировать плату при помощи программы Universal Scan, необходимо подключить байтбластер Altera или Xilinx к проверяемой плате и подать питание. Затем следует запустить программу Universal Scan (см. рис. 23), выбрать из меню тип байтбластера и установить его на поле. Далее нужно выбрать тип микросхемы и также установить символ микросхемы на поле. Если микросхем несколько, то последнее действие необходимо повторить. После этого надо нажать кнопку «Сканировать». Вот, собственно, и все. В результате сканирования выводы микросхем раскрасятся в соответствии с теми напряжениями, которые на них присутствуют.

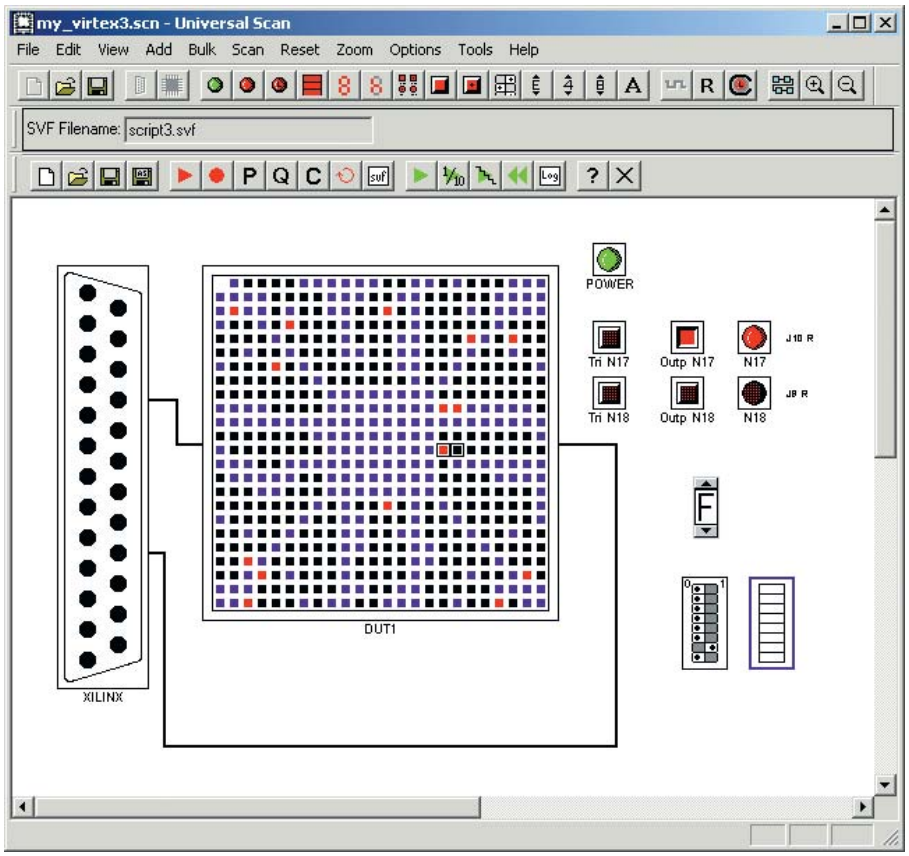

Рис. 23. Основное окно программы Universal Scan

Для указания типа микросхемы никаких особых усилий прикладывать не надо. Программа снабжена библиотекой микросхем, что позволяет иметь на рабочем поле графические символы, соответствующие выбранным микросхемам. Если же требуемой микросхемы в библиотеке нет, то программа предоставляет пользователю возможность самому сформировать необходимый символ, для чего просит предоставить ей BSDL-файл, на основе которого сама формирует изображение микросхемы. Далее начинается работа по проверке микросхемы или платы. Хотим задавать воздействие на определённый вывод? Пожалуйста! Помечаем вывод маленьким квадратиком (если присмотреться, то он виден на рис. 23) и открываем окно управления выводом (см. рис. 24). В этом окне расположены органы для управления состоянием вывода.

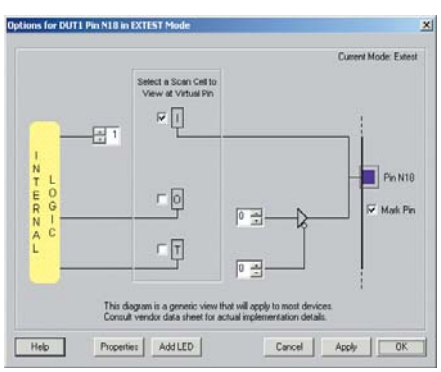

Рис. 24. Окно управления выводом программы Universal Scan

Посмотрите внимательно на сигнал разрешения управлением выхода для XC2V250_FG456, сравните его с тем, что нарисовано в Datasheet на эту микросхему. Правда состоит в том, что работает цепь именно так, как показано в Universal Scan, а не в Datasheet. Почему? Да потому, что Universal Scan работает так, написано в BSDL-файле.

Ну, а если мы хотим работать с несколькими выводами одновременно, то оказывается, гораздо проще ассоциировать с каждым из этих выводов виртуальный светодиод и виртуальные кнопки. Кнопки управляют сигналами разрешения выхода и самими сигналами выходов. Кроме того, есть наборы элементов управления – DIP-switch, цифровые переключатели и т.д. То же самое относится к элементам индикации. Кроме отдельных светодиодов, которые индицируют состояние напряжения питания или состояние одного вывода, существует множество различных виртуальных элементов индикации, таких как DIP-Leds, цифровые знакоместа и пр. Каждый индикатор может отображать данные в прямом или в инверсном коде. Также и каждая кнопка может быть запрограммирована на работу в инверсном коде. Все эти действия выполняются с верхней панелью функциональных кнопок, показанной на рис. 24. А вот нижняя панель функциональных кнопок предназначена для того, чтобы записывать действия оператора в файл SVF-формата, а затем воспроизводить эти действия в автоматическом режиме. Тут есть полная свобода для обучению языку SVF, так как всё происходит очень наглядно: записал – отредактировал – воспроизвёл. Набор тестовых воздействий постепенно превращается в тест проверки устройства.

Программа также снабжена видеороликами, где показываются действия оператора по тестированию и программированию проверяемых изделий.

Недостатком же программы Universal Scan можно назвать лишь её «нежелание» работать по интерфейсу USB с фирменными аппаратными адаптерами Altera и Xilinx. А тот USB-адаптер, который предлагается, – в свою очередь, несовместим с программными инструментами фирм Altera и Xilinx.

Очень коротко о SVF-файлах

Формат SVF-файлов дан в [15]. Этот документ описывает синтаксис для файлов SVF – последовательного векторного формата. Файлы SVF используются для описания операций высокого уровня, выполняемых по интерфейсу IEEE 1149.1. Формат SVF был разработан для того, чтобы упростить использование последовательных векторов при испытаниях тестируемого изделия. Файл SVF определён как ASCII-файл, который состоит из набора инструкций SVF и данных. Максимальное число символов в строке – не более 256, хотя одна инструкция SVF может быть записана на нескольких строках. Каждая инструкция состоит из команды и связанных параметров и завершается разделителем в виде точки с запятой. Текст в SVF-файле не зависит от регистра. Комментарии могут быть вставлены в SVF-файл после восклицательного знака – «!» или пары наклонных черт вправо – «//». Символы «//» или

«!» указывают на то, что остаток строки является комментарием.

Данные, передаваемые или принимаемые по командам тестирования в пределах каждой инструкции, записываются в шестнадцатеричном виде и всегда заключаются в круглые скобки. Естественно, что данные, предназначенные для сдвига, не всегда имеют длину, кратную 16. В таком случае нули в старшем байте при анализе отбрасываются. Порядок записи бит «старший-младший» соответствует соглашениям стандарта IEEE 1149.1.

Вот фрагмент сканирования микросхемы, выполненный на основании BSDL-файла – xc2v250.bsd:

```
!----STEP 2:
SIR 6 TDI(00) TDO(11) MASK(03);
SDR 732 TDI(969B6D…..DB6DB6D);
SDR 732 TDI(969B6D0….6DB6DB6D)
TDO(169B6D02B6DB…..DB6DB6D)
MASK(80000…080000….0000000);
```

Что делается в этом фрагменте кода? Заремаркированная строка сообщает, что выполняется шаг тестирования № 2. Далее указывается, что длина регистра инструкций составляет 6 бит, потом написано то, что требуется выдать по проводу TDI, то, что необходимо получить на TDO, и маску, по которой проверяются полученные данные.

Следующая строка указывает, что длина регистра данных (т.е. то, что мы называли регистром граничного сканирования) составляет 732 бит. Следующие строки описывают, как и в случае сканирования команды, то, что требуется выдать по проводу TDI, то, что необходимо получить на TDO, и маску, по которой проверяем полученные данные.

SVF-файл позволяет задавать воздействия на исследуемую микросхему, проверять результаты воздействия, формировать воздействия для группы микросхем, объединённых в цепочку, задавать временные параметры сканирования, а также программировать порты ввода-вывода.

Ну а где же «про тестирование»?

И вот здесь читатель вправе спросить: «А где же про само тестирование, ведь именно это и было обещано?» Итак, как же выполняются процедуры тестирования? Давайте рассмотрим их шаг за шагом. Рассматривать тестирование будем на примере проверки смонтированной печатной платы (PCB). Пусть на этой плате есть несколько микросхем, объединённых в цепочку, которые имеют порт JTAG.

Первым делом определим порядок подключения микросхем в цепи JTAG, затем, используя BSDL-файлы этих микросхем, произведём следующие действия:

- выделим коды команд тестирования, длину кода для каждой команды и т.д. так, как это описывалось выше;
- выделим фрагменты, соответствующие регистру граничного сканирования и рассмотрим безопасный режим для каждой из микросхем;
- определим биты управления и их состояние для безопасного режима;
- определим, на каких входах этих микросхем входные данные будут определены и в каких ситуациях.

Более подробно рассмотрим последнее действие. Если мы тестируем изделие, значит, на него есть документация в виде схемы, а ещё лучше, если на плату есть проект для программного инструмента типа PCAD, Power PCB или любого другого аналогичного инструмента. При помощи программного инструмента можно легко выделить список связей из проекта. По списку связей и по схеме необходимо определить, какая информация будет читаться на входах микросхем. Если вход микросхемы подключен к какому-либо выходу, не имеющему третьего состояния, то на таком входе всегда должна присутствовать информация, определяемая состоянием указанного выхода. Если информация на тестируемом входе отличается от той, которую мы ожидаем увидеть, то это значит, что в цепи имеет место неисправность. То же самое относится к случаю, когда один выход тестируемой микросхемы, не имеющий третьего состояния, подключен к одному или к нескольким входам другой микросхемы. Состояние выхода будет всегда определено, и, соответственно, будет определено состояние тех входов, на которые поступает сигнал с тестируемого выхода. То же самое можно сказать и о входах, на которые поданы сигналы опорного напряжения или которые соединены с общим проводом. Это будет справедливо и для выходов, которые через какие-либо компоненты подключены к «плюсу» или «на землю». Примером таких цепей могут служить выходы с подключенными к ним светодиодами.

Совсем другое дело, если в проверяемой цепи присутствует выход микросхемы, имеющей третье состояние. Сначала рассмотрим цепь, состоящую из выходного буфера с третьим состоянием и подключенными к ней одним или несколькими входами. Если к этой цепи не подключены опорные резисторы, то будет иметь место следующее:

- если управление третьим состоянием выключено и выход находится в активном состоянии, то из цепи читается информация, полностью соответствующая состоянию выхода;
- если управление третьим состоянием включено и выход находится в неактивном состоянии, то из цепи читается либо информация, соответствующая состоянию подключенного к этой цепи входа, либо неопределённая информация.

Во втором случае на информацию, которая считывается из цепи, может влиять ещё незакончившийся процесс перезарядки конденсаторов, когда цепь хранит состояние, предшествующее тому, в котором производилось измерение.

Кроме того, в тестируемую цепь возможно проникновение помехи от соседних цепей или от источника питания. Процесс перезарядки конденсаторов, вызываемый последовательностью выполнения тестов, всегда определён. В том случае, когда информация определяется наводкой, результат чтения из цепи носит нерегулярный характер. Если же в цепь включен опорный резистор или имеется компонент, который хранит последний по времени уровень сигнала – keeper, то информация, читаемая из такой цепи, становится определённой.

Итак, надо определить, в каких битах регистра сканирования будет достоверная информация при текущем состоянии схемы. Чтобы упростить определение неисправных цепей, введём такие понятия, как «выданная в регистр сканирования строка», «принятая из регистра сканирования строка», «строка маски» и «строка паттерна». Если данные, которые мы выдаём в регистр сканирования, представить как строку из набо-

ра нулей и единиц, то вот её мы и назовем «выданная в регистр сканирования строка». Принятые данные будем трактовать как «принятая из регистра сканирования строка». Тогда термин «строка маски» будет обозначать строку данных, состоящую также из нулей и единиц. На тех позициях, которые нас не интересуют в данном тесте, мы поместим нули, а там, где мы хотим увидеть и проанализировать результат чтения из регистра граничного сканирования, мы поместим единицы. Значение, которое мы хотим получить при тестировании, назовем «строкой паттерна».

Тестирование производится следующим образом. Выдаём строку данных в регистр сканирования, получаем из регистра данные. Затем на принятые данные накладываем маску и полученный результат сравниваем с паттерном. Как видите, всё очень просто.

Но тогда почему же все средства тестирования, а особенно программные инструменты, так дороги? Ответ тоже прост, ибо, как говориться в одной рекламе: «Все дело в волшебных пузырьках»... Во-первых, хорошие программы сами работают со списком связей и сами разделяют цепи на группы, в соответствии с которыми будут выданы тестирующие воздействия. Точно так же эти программы сами формируют цепочки для регистров сканирования в том случае, когда в цепи включено несколько микросхем. Ну и, наконец, эти программы сами определяют порядок выдачи тестовых воздействий.

Рекламная пауза окончена, и теперь мы более подробно продолжим рассматривать тестовые воздействия. Как нам уже известно, команды Sample-Preload и Extest переключают мультиплексор регистра граничного сканирования на чтение внешних по отношению к микросхеме данных, а команда Intest – внутренних данных, т.е. данных из ядра микросхемы (Core Logic). И ещё одно важное замечание. Команда Intest читает данные, которые уже сформированы в ядре микросхемы, поэтому здесь достаточно одного цикла чтения. Команда Sample-Preload записывает новые данные в регистр граничного сканирования, но не выводит записанные в регистр граничного сканирования данные из микросхемы. Читаются же внешние данные, поступающие на входы микросхемы. Здесь также достаточно одного цикла чтения. А вот команда Extest, как только она будет выдана в цикле записи команды, тут же выводит из микросхемы записанные в регистр граничного сканирования данные, и они появляются на выходах. Если после выдачи команды Extest мы произведём запись данных, то эти новые данные появятся на выходах только после завершения цикла чтения-записи. В этом случае данные, которые появятся на выходах микросхемы после выполнения команды Extest, могут изменить состояние проверяемого изделия. Поэтому в таком случае достоверно считанными данными будут те, которые мы получим после второго чтения.

Мы рассмотрели, какие команды и в какой регистр нужно помещать, чтобы данные с выхода одной микросхемы прочитать на входе другой. Но и это ещё не всё! Вспомните, что длина регистра граничного сканирования у микроконтроллера или у микросхемы FPGA может быть от нескольких сотен до тысячи и боле бит! А теперь представьте, что у вас в цепочке находится хотя бы пять таких микросхем. Тогда для каждого теста нам надо выдать $5 \times 1000 \times 2$ циклов записи по JTAG – всего 10 000 циклов. Представим, что мы хотим выполнить тест «бегающая единица». И ещё представим, что эти пять микросхем расположены, например, в трёх физически разделённых узлах на плате. То есть цепи от одной группы микросхем никак не пересекаются с цепями от другой группы и, следовательно, не могут иметь общих неисправностей, например замыканий. Тогда для такой платы будет нецелесообразно «гонять» тесты по 10 000 циклов. Разделим также и тесты на три группы. Для тех микросхем, которые не относятся к определённой группе, применим режим Clamp или HighZ. Вспомним, что в режиме Clamp данные на выходах микросхемы будут зафиксированы, а в режиме HighZ выходы будут находиться в третьем состоянии. Таким образом, в те микросхемы, куда мы подавали команду Clamp, надо выдать данные только один раз в начале теста, а потом данные будут «защёлкнуты» и останутся неизменными при последующих тестах.

При такой методике тестирования суммарная длина выдаваемой цепочки сократится в три раза. Соответственно, сократится и время обработки результатов тестирования.

И последнее, что необходимо упомянуть, – это сама методика тестирования изделия. Нет нужды описывать простейшие тесты – «шахматные коды», «счётчик», «бегающая единица», – они общеизвестны.

Сколько именно тестов надо выполнить, в какой последовательности и насколько достоверным при этом будет результат – выходит за рамки данной статьи.

Технологии JTAG-тестирования ещё развиваются. Кроме технологии тестирования для дискретных сигналов, описанных в статье, появились технологии для смешанных аналого-цифровых сигналов. Такие технологии применяются в том случае, если в изделии используются стандарты сигналов типа LVDS. Также появились и технологии для сканирования аналоговых сигналов. Такие технологии призваны обеспечить тестирование, например, цепей питания. Так же, как и технологии сканирования, развиваются и технологии внутрисхемного программирования.

Заключение

Кроме тех программ и путей решения проблем тестирования, которые обсуждались выше, существует достаточно большое количество методов, позволяющих проверять изделия. В рамках журнальной статьи нет возможности описать или хотя бы перечислить их все. Если же сравнивать ручное и автоматическое тестирование, то разница между ними будет в том, что при автоматическом тестировании можно задать больше тестов, выбрать различные форматы работы выводов микросхемы и т.д. Для автоматического тестирования лишний час при проверке десятков тысяч цепей на каждой плате никак не сказывается на общих расходах фирмы. Ведь сегодня простенький компьютер, занимающийся тестированием, гораздо дешевле, чем регулировщик, задействовать таких компьютеров можно довольно много, а использовать их можно «в три смены».

В последнее время в российских периодических изданиях всё больше компаний рекламируют свои возможности по изготовлению печатных плат и контрактной сборке. А это значит, что проблемы автоматического тести-

рования станут более актуальными и для российских разработчиков.

Литература

1. ADI http://www.analog.com.
2. ООО «Элтех» www.eltech.spb.ru.
3. Boundary Scan Coach. GOEPEL Electronic. http://www.goepel.com.
4. Design and Java Applets Support for an Asynchronous-Mode Learning of Digital Test. http://www.pld.ttu.ee/applets.
5. Universal Scan. Ricreations, Inc. http://www.universalscan.com.
6. Платунов А.Е., Постников Н.П., Чистяков А.Г. Механизмы граничного сканирования в неоднородных микропроцессорных системах. Chip News. http://lmt.cs.ifmo.ru/article_chip_news.html.
7. Рустинов В., Городецкий А. «Разделяй и властвуй» – принцип граничного сканирования. Chip News. http://chip-news.gaw.ru/html.cgi/arhiv/01_06/stat-3.htm.
8. Каршенбойм И.Виртуальные кнопки и светодиоды, или Неизвестное обо всём известном JTAG_сканировании. Компоненты и технологии. 2005. № 6.
9. http://www.national.com/appinfo/scan/index.html.
10. IEEE 1149.1 (JTAG) Boundary-Scan Testing for Stratix II Devices. Ch. 9. Altera. www.altera.com.
11 Kuznetsov D. JTAG Boundary-Scan Test. Introduction. http://www.orc.ru/~dkuzn/j_intro.htm.
12. http://jtagtools.sourceforge.net/download.html.
13. IEEE Standard Test Access Port and Boundary-Scan Architecture. IEEE Std 1149.1-2001.
14. Boundary-Scan Test and In-System Programming Software. Corelis. http://www.corelis.com/products/Test_Software.htm.
15. Serial Vector Format Specification. ASSET InterTech, Inc. Copyright 1997-1999. Texas Instruments Inc. Copyright 1994. www.asset-intertech.com/support/svf.pdf.
16. www.amontec.com. JTAG Interface: Common Pinouts amt_ann003 (v1.1) Application Note.
17. EIA/JEP106, JEDEC Publication 106, Standard Manufacturer's Identification Code.
18. Каршенбойм И. Микропроцессор своими руками/4. Как отладить встроенный в FPGA микроконтроллер? Компоненты и технологии. 2006. № 11.