Виртуальные кнопки и светодиоды,
или Неизвестное обо всем известном JTAG-сканировании

Иосиф КАРШЕНБОЙМ
iosifk@narod.ru

Здравствуй, уважаемый читатель!
Да, именно так хочется начать эту статью. Почти два года прошло с тех пор, когда была опубликована моя последняя статья. Спасибо всем тем, кто читает мои статьи, а тем более спасибо тем, кто мне пишет. И что бы не писали о гонорарах известных детских писательниц, за то, что написано у нас, такого не получишь. Поэтому писатели, которые печатаются на этих страницах, пишут в основном не для себя, а для читателей. Спасибо тем читателям, которые предлагают всем «миром» сделать открытые проекты микроконтроллеров. Есть среди них опытные «коты», как, например, романтик Евгений из Москвы, есть и совсем неопытные, такие как Александр из израильского университета Ben-Gurion University Бен-Гур?. Что касается открытых российских проектов, то тут, я думаю, должны сказать свое слово наши университеты. Тем же, кто руководствуется тезисом «а как бы мне от вас чего-нибудь бесплатно», лучше обратиться на opencores. Там вы получите то, что хотите. А если работать не будет, не обессудьте — это же opencores!
Итак, спасибо всем. А теперь начинается новая история.

Сижу я тихо и никого не трогаю...

Вдруг прибегает руководитель проекта (честное слово, все так и было!) и заявляет, что самый главный хочет, чтобы я… (дальше пропускаю)… программу JTAG-тестирования платы… (дальше пропускаю)… 49 микросхем xc2v250_fg456, примерно 9 тыс. цепей… за две недели. Ну и, конечно, потом чтобы и остальные платы тоже. И здесь я тоже дальше пропускаю… В начале статьи приведена фотография с микросхемами, а то что автору надо было тестировать выглядит примерно также, только микросхем много больше.

Вот таким образом и пришлось познакомиться с JTAG-тестированием. Нельзя сказать, чтобы слово JTAG было совсем неизвестно, но вот в самом тестировании нашлось очень много того, чего не было написано в документации и в статьях на эту тему. На этих страницах будет написано только то, что автору хотелось бы и самому увидеть и прочитать в начале разработки программы JTAG-тестирования.

Сначала надо написать о JTAG то, что пишут все

Здесь надо бы очень коротко повторить избитые фразы о том, что такое JTAG и что такое JTAG-сканирование, и чем это отличается от загрузки FPGA или от отладки микропроцессоров.
Для начала отошлю читателей к списку литературы [1–3]. Теперь тоже самое, но более простыми словами. Желание сделать процесс проверки разрабатываемых и выпускаемых изделий более простым привел к тому, что в состав микросхем, плат и устройств стали вводить специальные узлы, предназначенные только для целей автоматической проверки изделий. Данные узлы подключались к компьютеру по последовательному интерфейсу. Ну а когда появился достаточно простой и хорошо стандартизованный интерфейс, то к нему стали добавлять различные функции, определяемые типом самого изделия. Именно так появилась загрузка «прошивки» через JTAG, отладка микроконтроллеров и все прочее. Потребность в более сложных функциях тестирования и отладки породила, в свою очередь, специализированные языки описания процессов тестирования и программирования.

Проверка, проверка и еще раз проверка...

Представим, что изделие, которое мы хотим проверить — это песочные часы. Перевернул такие часы — и процесс тестирования пошел. А если изделие более сложное? Достоверность получаемых при проверке данных будет значительно меньше. И чем сложнее само тестируемое изделие, тем менее достоверными будут результаты тестирования. Где же выход? Выход из этого тупика был найден уже очень давно. Сложное изделие или сложный процесс при проверке заменяется на менее сложное изделие или процесс и по нему определяются результаты тестирования. Пример — песочные часы, о которых написано выше. Сложный

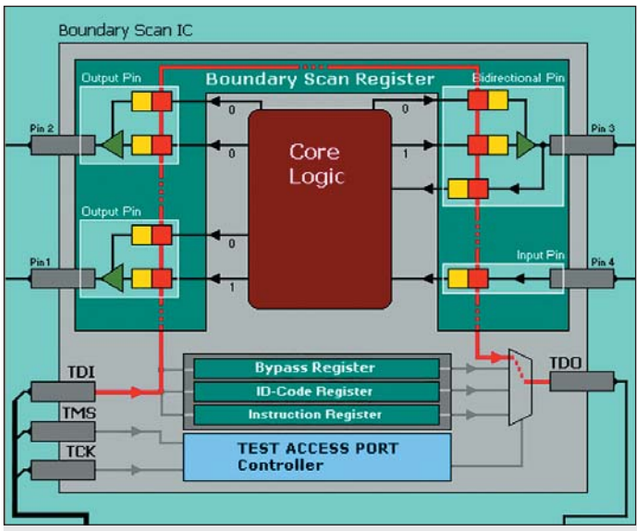

Рис. 1. Структура узлов JTAG-сканирования (BSR.bmp)

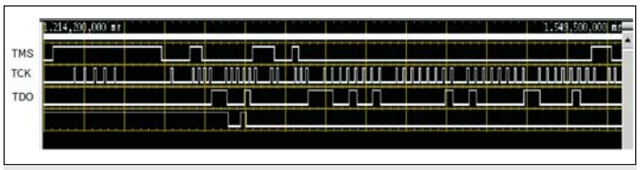

Рис. 3. Вид диаграммы на экране логического анализатора

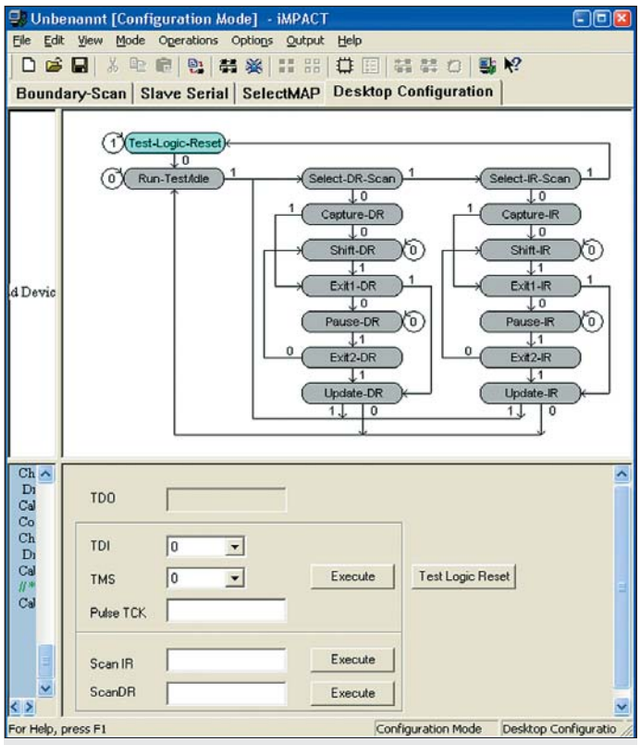

Рис. 2. Структура управляющего автомата JTAG-сканирования на примере программы iMPACT

процесс определения временного интервала заменяется на очень простой, а следовательно, более надежный, процесс перемещения песка из одного сосуда в другой. И, если кому-то уже стало скучно, то давайте вспомним, что такое сторожевой таймер (watchdog timer) и для чего он используется. Сложный процесс — выполняемая процессором программа — сравнивается при проверке с образцовым процессом, который выполняется в таймере. И, поскольку таймер гораздо надежнее, и выполняемый им процесс проще, чем процессор + память команд + память данных + + выполняемая программа и пр., то и результат получается более-менее достоверный.

Теперь, представим, что мы хотим тестировать микросхему. Что же надо добавить в микросхему, если мы хотим ее тестировать? Ответ очевиден — надо добавить то, что будет наиболее простым, и, следовательно, наиболее надежным. Заглянем в литературу [1] и увидим, что мы на правильном пути, ибо в микросхему добавлен простой регистр. Число выводов должно быть минимально по тем же соображениям. Следовательно, регистр должен быть сдвиговым. Ну и, конечно, блок управления. Далее в этой статье описание процесса тестирования приводится по отношению к микросхеме, так как это тот «атом», из которого состоят все устройства. Что касается проверки плат в целом или устройств в целом, то по отношению к JTAG-порту они ведут себя точно так же, как и отдельная микросхема.

Где взять учебник?

Где же взять учебник, если вас этому не учили. К сожалению, учебников по этой теме очень мало, и то, что приводится здесь, нашлось в сети с большим трудом. Жаль, что поисковый сервер не сообщил ни одного адреса в российском Интернете, хотя эта тема вполне достойна внимания российского студенчества.

Первое и, можно сказать, не стареющее издание — программа scanedu.exe от TI.

Второе — Boundary Scan Coach от GOEPEL Electronic [4].

Третье — страницы, посвященные JTAG-тестированию, на сайте Таллиннского университета [5].

Четвертое — Universal Scan. Данная программа может считаться неплохим учебником и будет подробно рассмотрена далее.

Вот и весь перечень. Программы-учебники позволяют просимулировать сканирование микросхем, показывают содержимое регистров, входные и выходные переменные.

Конечно, в сети существует бесчисленное число статей, но кроме общих фраз читать в них особо нечего.

JTAG в картинках

Теперь посмотрим на JTAG в картинках. На рис. 1 приведен фрагмент программы Boundary Scan Coach — типовая структура узлов JTAG-сканирования, встроенных в микросхему. Функциональная часть микросхемы — Core Logic — показана коричневым цветом. JTAG-регистры — зеленым цветом, выводы микросхемы — серым цветом. Голубым цветом выделен контроллер JTAG-порта. Красным цветом показан путь прохождения данных по регистру граничного сканирования. Названия сигналов и регистров, а также выполняемые ими функции достаточно широко описаны, поэтому будем считать, что все это читателю уже известно. Далее автор попытается описать только то, что не было опубликовано в предыдущих изданиях. Особое внимание будет обращено на ошибки, которые были обнаружены в технической документации фирм-производителей.

Рассмотрение работы JTAG начнем с классификации. Интерфейс JTAG — последовательный. Он предназначен для записи и чтения данных и команд управления. Команды управления интерфейсом записываются в регистр косвенного адреса. Данные записываются и считываются из того регистра, на который указывает регистр косвенного адреса. Итак, есть раздельные циклы записи адреса, и отдельные циклы чтения-записи данных. Чтение и запись данных производится одновременно. Данные в микросхему выдаются на вывод TDI, а принимаются из микросхемы с вывода TDO. Управление интерфейсом осуществляется путем воздействия на автомат контроллера JTAG-порта. Сигналы управления на контроллер JTAG-порта приходят по входу TMS. Диаграмма переходов автомата управления JTAG-портом является необходимым призна-

ком статьи о работе JTAG. Но, поскольку было заявлено, что автор не будет повторять предыдущие статьи, то на рис. 2 приведен фрагмент программы iMPACT, входящей в состав пакета проектирования Xilinx ISE. Данную программу так же можно использовать как для целей обучения, так и для проверки изделий, содержащих микросхемы с JTAG-портами.

Лучшее описание работы управляющего автомата JTAG-сканирования приведено в описании работы микросхем фирмы Altera [7].

То, как это выглядит в жизни на экране логического анализатора, приведено на рис. 3. К сожалению, нет возможности тщательно описать эту диаграмму — так как это делает Altera. Приводится она здесь только для того, чтобы читатель сам смог убедиться, что без программных средств расшифровывать такие диаграммы довольно затруднительно.

Далее необходимо отметить следующее. Понятно, что записывается в микросхему то, что мы туда посылаем. А вот что считывается из микросхемы?

Если мы посылаем команду, то считываем, можно сказать, регистр состояния микросхемы. Например, для микросхемы серии Virtex2 XC2V250_FG456 в BSDL-файле читаем (подробнее о BSDL-файле будет написано далее):

```
attribute INSTRUCTION_CAPTURE of XC2V250_FG456: entity is
-- Bit 5 is 1 when DONE is released (part of startup sequence)
-- Bit 4 is 1 if house-cleaning is complete
-- Bit 3 is ISC_Enabled
-- Bit 2 is ISC_Done
        «XXXX01»;
```

Если мы посылаем данные, то считываем то, что находится в тех регистрах, на которые указывает регистр косвенного адреса.

Если речь идет о регистрах «USERCODE» или «IDCODE» то, конечно, мы прочтем те данные, которые там уже хранятся. А вот что будет читаться из регистра граничного сканирования?

Сначала вспомним сам термин «граничное сканирование». Но ведь у границы есть две стороны — внешняя (заграница) и внутренняя. Так с какой же стороны границы мы читаем данные? А с той, которая определена выданной перед чтением командой. Команды Sample-Preload и Extest переключает мультиплексор регистра граничного сканирования на чтение внешних по отношению к микросхеме данных, а команда Intest — внутренних данных, то есть данных из ядра микросхемы (Core Logic).

И еще одно важное замечание. Команда Intest читает те 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 произведем запись данных, то эти новые данные появятся на выходах только после завершения цикла чтения-записи данных. В этом случае данные, которые появятся на выходах микросхемы после выполнения команды Extest, могут изменить состояние проверяемого изделия. Поэтому в данном случае достоверно считанными данными будут те, которые мы получим после второго чтения.

Можно также выполнять не два цикла Extest + + Данные и Extest + Данные, а два цикла Sample-Preload + Данные и Extest + Данные. В таком случае в цикле Sample-Preload данные на выдачу будут записаны в выходной регистр, но на выходе не появятся. Далее, как только будет выдана команда Extest, на выходах микросхемы появятся нужные данные, а во время выполнения чикла записи-чтения команды Extest в регистр граничного сканирования запишутся достоверные данные.

Выводы:

1. Можно проверить «внутреннюю начинку» микросхемы, эмулируя для нее внешние воздействия.
2. Можно выдать в «окружающий мир» сигналы из микросхемы, эмулируя работу ядра микросхемы.
3. Можно организовать проверку самого «окружающего мира», например межсоединений между микросхемами, от микросхем к разъемам и т. д.

Теперь заглянем внутрь регистра граничного сканирования

BSDL-файл! Как красиво звучит! Жаль только, что Xilinx к ним (а следовательно, и к нам) так наплевательски относится. Только после третьего письма в техподдержку они втихаря заменили файл на своем сайте. Осторожно, дорогой читатель! Тот файл, который вы взяли сегодня с сайта этой фирмы, может быть завтра заменен на другой. Это не важно, что файл помечен как проверенный. Важно только то, что фирма не сообщает о замене файлов.

Еще одно хорошее место для поисков BSDL-файлов — это тот пакет программ, с которым вы работаете, так как обычно среди прочих папок там находится и скромная папка с названием «BSDL».

Теперь несколько слов о структуре BSDL-файла. Это VHDL-подобный файл, описывающий то, как сформирован регистр сканирования.

Первая часть — порты.

Термины «in bit, out bit, inout bit» вполне понятны, они описывают порты, которые подключены к регистру сканирования как входы, как выходы или как двунаправленные входы-выходы. Термины «linkage bit, linkage bit_vector» описывают порты, которые не подключены к регистру сканирования.

Вот так выглядят выделенные выводы:

```
NOCONNECT: linkage bit_vector (1 to 124);
TCK: in bit;
TDI: in bit;
TDO: out bit;
TMS: in bit;
```

А так выглядят выводы типа IO, то есть двунаправленные входы-выходы:

```
IO_A5: inout bit; -- PAD6
IO_A6: inout bit; -- PAD10
IO_A10: inout bit; -- PAD18
```

Здесь «IO» — тип вывода, «A5» — контакт микросхемы, и в примечании «PAD6» — технологический пад чипа, с которого сделана разварка на вывод A5.

И далее, сначала для микросхем фирмы Xilinx:

```
attribute BOUNDARY_LENGTH of XC2V250_FG456 : entity is 732;
```

То есть длина регистра — 732 бита. А вот как описываются сами ячейки регистра:

```
attribute BOUNDARY_REGISTER of XC2V250_FG456: entity is
-- cellnum (type, port, function, safe[, ccell, disval, disrslt])
        « 0 (BC_2, *, controlr, 1),» &
        « 1 (BC_2, IO_B19, output3, X, 0, 1, PULL0),» & -- PAD48
        « 2 (BC_2, IO_B19, input, X),» & -- PAD48
```

Здесь BC_2 — это тип ячейки (ячейка с двунаправленным выходом, присоединенная к контакту микросхемы B19). Бит 0 — управление выходом, бит 1 — выход, бит 2 — вход. Для сравнения посмотрим на ячейку для микросхем фирмы Altera:

```
attribute BOUNDARY_REGISTER of EPF10K20R240: entity is
 --BSC group 0 for untestable Family-specific pin
«0 (BC_4, *, internal, X),» &
«1 (BC_4, *, internal, 1),» &
«2 (BC_4, *, internal, X),» & …

 --BSC group 2 for I/O pin 6
«6 (BC_1, IO6, input, X),» &
«7 (BC_1, *, control, 1),» &
«8 (BC_1, IO6, output3, X, 7, 1, Z),» & …
```

И, как мы видим, все тоже, но несколько в другом порядке.

Остальное в BSDL-файле более-менее понятно. Есть только еще одно тонкое место. Поскольку регистр Boundary Scan — сдвиговый, то очень важно то, в какую сторону производится сдвиг. Для JTAG-технологии все, что есть, сдвигается вправо. Поэтому в документации есть легкая путаница с младшими и старшими битами. И если в одном месте данные для сдвига изображены вот так:

```
attribute INSTRUCTION_OPCODE of XC2V250_FG456: entity is
        «EXTEST (000000),» &
        «SAMPLE (000001),» & …
```

то это значит, что команда SAMPLE выдается на TDI как 1,0,0,0,0,0. Ну а в другом месте документации можно встретить совершенно разные высказывания по этому поводу.

Теперь еще раз посмотрим на рис. 1. Это часть скриншота программы Boundary Scan Coach. Теперь мы можем проверить свои знания на этом симуляторе. На рис. 4 показана

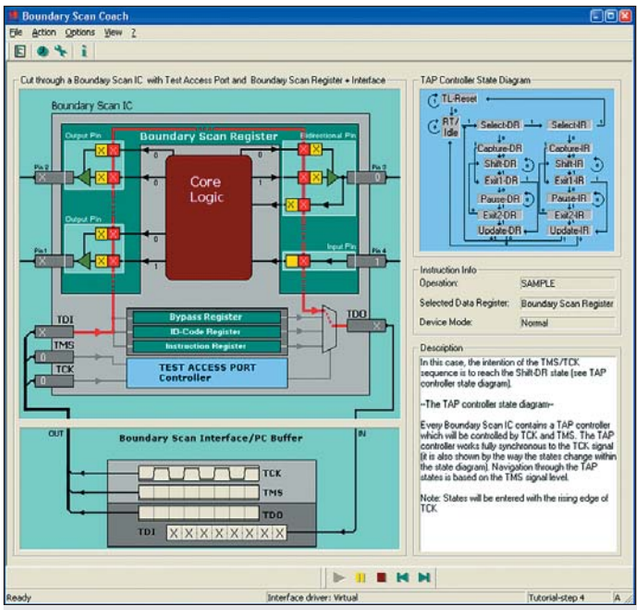

Рис. 4. Окно программы Boundary Scan Coach, часть № 2

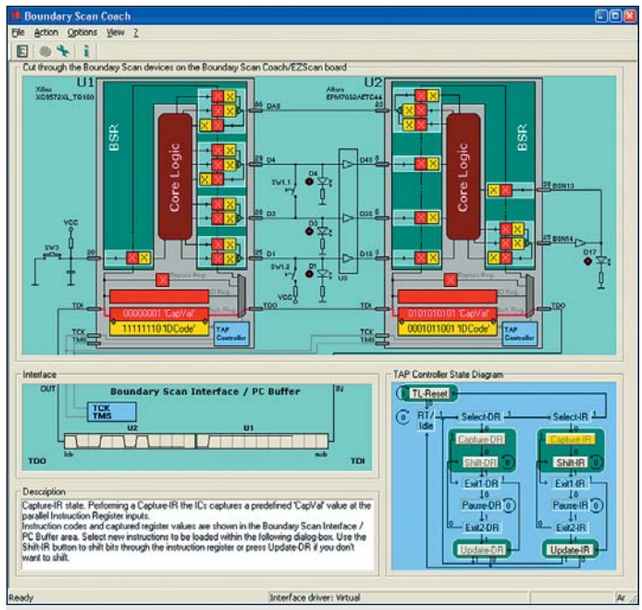

Рис. 5. Окно программы Boundary Scan Coach, часть № 3

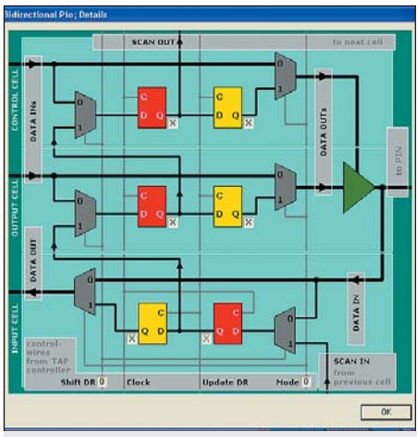

Рис. 6. Окно линзы, показывающее состояние выводов в программе Boundary Scan Coach

часть № 2 этой программы. Здесь пользователь может установить данные в регистр граничного сканирования и посмотреть на результат, полученный после записи команд и данных.

Но наибольший интерес с точки зрения граничного сканирования представляет собой часть № 3 этой программы. Окно программы Boundary Scan Coach, часть № 3, приведено на рис. 5. В этой части программы представлено то, как производится сканирование устройства, содержащего две микросхемы. По цепи JTAG-тестирования микросхемы включены последовательно. Данные сдвигаются из порта в первую микросхему, а из нее во вторую. Пользователь может задавать различные режимы работы выводов, например, выдавать воздействия из одной микросхемы в другую. В режиме линзы можно посмотреть состояние выводов, так, как это показано на рис. 6.

От теории к практике!

Осталось лишь убедиться в том, что все описанное в этой статье применимо к тому изделию, которое вы хотите проверять. Вроде уже и теоретических знаний много, да вот на практике как все это применить? Как проверить, что считывается на входе микросхемы? Как выдать необходимые сигналы на вывод?

Путь № 1, классический. Быстро садимся и учим языки, при помощи которых описываются тестовые воздействия. Потом (неизвестно где) находим работающие компиляторы этих языков. Далее понятно…

Путь № 2, обычный, наш. Берем C++, и пишем [8, 9].

Путь № 3, необычный. Ищем в сети и устанавливаем Universal Scan [10].

Что же позволяет эта программа?

1. Позволяет сканировать устройство пользователя, управлять выходами и читать информацию на входах.
2. Программировать Flash-память.

В этой статье будет рассмотрено только JTAG-тестирование.

Вот какие действия необходимо выполнить, чтобы начать тестировать плату. Подключить Altera или Xilinx байтбластер к проверяемой плате, подать питание. Запустить программу Universal Scan (рис. 7). Выбрать из меню тип байтбластера и установить его на поле. Выбрать тип микросхемы и так же установить символ микросхемы на поле. Если микросхем несколько, то последний пункт необходимо повторить. Далее надо нажать кнопку «Сканировать». Вот собственно и все. В результате сканирования выводы микросхем раскрасятся в соответствии с теми напряжениями, которые на них присутствуют.

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 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 определенный вывод — пожалуйста. Помечаем вывод маленьким квадратиком (если присмотреться, то он виден на рис. 7) и открываем окно управления выводом (рис. 8). На этом окне расположены органы для управления состоянием вывода.

Здесь будет небольшое отступление. Посмотрите внимательно на сигнал разрешения управлением выхода для XC2V250_FG456, сравните его с тем, что нарисовано в Datasheet. Правда состоит в том, что цепь работает именно так, как показано в Universal Scan, а не в Datasheet.

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 и с элементами индикации. Кроме отдельных светодиодов, которые индицируют состояние напряжения питания или состояние одного вывода, существует множество различных виртуальных элементов индикации, таких как DIP-Leds, ци-

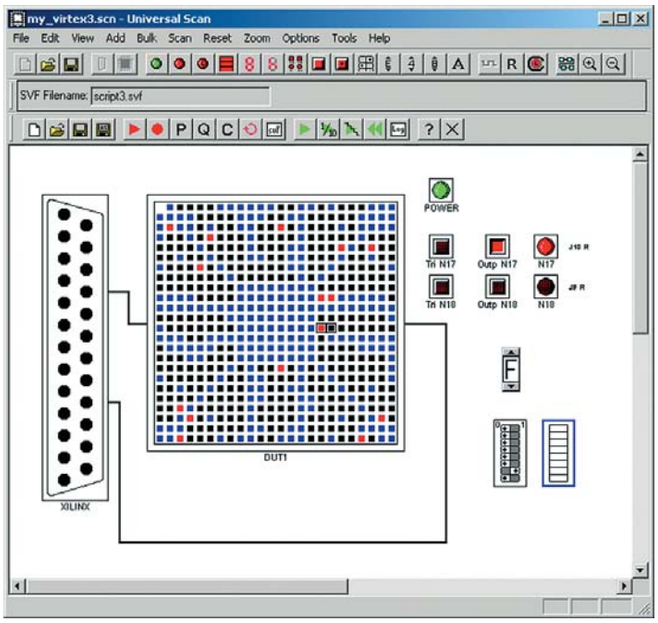

Рис. 7. Основное окно программы Universal Scan

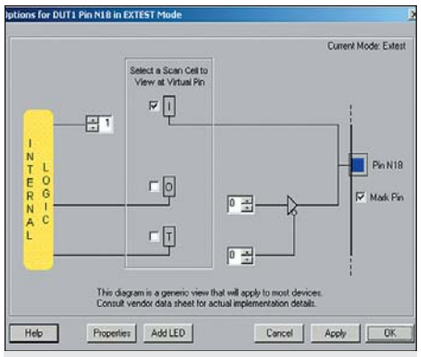

Рис. 8. Программа Universal Scan, окно управления выводом

фровые знакомства и пр. Каждый индикатор может отображать данные в прямом или в инверсном коде. Так же и каждая кнопка может быть запрограммирована на работу в инверсном коде. Все эти действия выполняются с верхней панелью функциональных кнопок, показанной на рис. 8. А вот нижняя панель функциональных кнопок предназначена для того, чтобы записывать действия оператора в файл SVF-формата, а затем воспроизводить действия оператора в автоматическом режиме. Тут есть полная свобода к обучению языку SVF, так как все происходит очень наглядно. Записал — отредактировал — воспроизвел. Набор тестовых воздействий постепенно превращается в тест проверки устройства.
Программа также снабжена видео-роликами, где показываются действия оператора по тестированию и программированию проверяемых изделий.

Очень коротко о SVF-файлах

Формат SVF-файлов описан в документации [12]. Вот фрагмент сканирования микросхемы, выполненный на основании BSDL-файла xc2v250.bsd.

```
!----STEP 2:
    SIR 6 TDI(00) TDO(11) MASK(03);
    SDR 732
TDI(969B6D02DB40B6D02DB40B6D02DB4036D02DB40B6D02DB40
B6DB6DBB6DB6DB6DB6D76DB6DB6DB6DB6DB6DB6DB6DBADB6
DB40B6D02DB40B6D02DB40B6D02DB40B6D02DB40B6D02DB6DBE
B6DB6DB4DB6DB6DA6DB6DB6DB6DB6DB6D);
    SDR 732
TDI(969B6D02DB40B6D02DB40B6D02DB4036D02DB40B6D02DB40
B6DB6DBB6DB6DB6DB6D76DB6DB6DB6DB6DB6DB6DB6DBADB6
DB40B6D02DB40B6D02DB40B6D02DB40B6D02DB40B6D02DB6DBE
B6DB6DB4DB6DB6DA6DB6DB6DB6DB6DB6D)
TDO(169B6D02DB40B6D02DB40B6D02DB4036D02DB40B6D02DB40
B6DB6DBB6DB6DB6DB6DF6DB6DB6DB6DB6DB6DB6DB6DBADB6
DB40B6D02DB40B6D02DB40B6D02DB40B6D02DB40B6D02DB6DBE
B6DB6DB4DB6DB6DB6DB6DB6DB6DB6DB6D)
MASK(800000000000000000000000000000000000000000000000
000000000000800000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000
000000000);
```

Что делается в этом фрагменте кода? Сначала указывается, что длина регистра инструкций — 6 бит, потом написано то, что мы хотим выдать по проводу TDI, то, что хотим получить на TDO, и маска, по которой проверяем полученные данные.
В следующей строке указывается, что длина регистра данных (то, что мы называли регистром граничного сканирования) — 732 бита. Следующие строки описывают, как и в случае сканирования команды, то, что хотим выдать по проводу TDI, то, что хотим 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, объединенных в цепочку. В нем также задаются временные параметры сканирования. Можно программировать и порты ввода-вывода.

Ну и совсем коротко о DLP-2232M

Маленькая и очень дешевая платка DLP-2232M (рис. 9) — контроллер USB-JTAG.

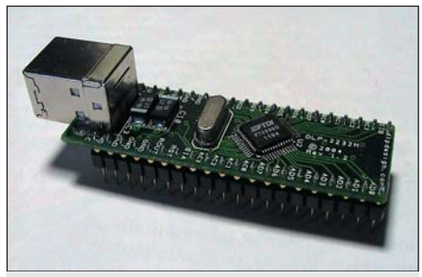

Рис. 9. Плата DLP-2232M — контроллер USB-JTAG

Легко покупается со склада. Фирма DLP Design, Inc. (**www.dlpdesign.com**) успела сделать специальный драйвер для программирования по JTAG. Основа DLP-2232M — микросхема FT2232 фирмы Future Technology Devices International (**www.ftdichip.com**). В режиме работы JTAG-контроллера достигается скорость 5,6 Мбит/с. Кроме сигналов JTAG-интерфейса так же доступны 8 линий портов, которые могут назначаться как входы или как выходы. DLP-2232M подключается к USB-порту компьютера. Никаких особых знаний для программирования USB не требуется, это все «скрыто» в поставляемом драйвере. Необходимо только «приготовить» передаваемые данные в массиве, указать на него драйверу, сообщить, сколько бит мы хотим передать и в какое состояние по диаграмме управления JTAG-портом (рис. 2) мы хотим перейти.
Ниже приведен пример записи в JTAG-порт через USB.

```
// Write buffer to device under test
Status = JTAG_Write(ftHandle,                    //
                    true,                        // команда или данные
                    8,                           // число
                                                    записываемых бит
                    &WriteDataBuffer,            // буфер на запись
                    65535,                       // объем буфера
                    TEST_LOGIC_STATE);           // в какое состояние
                                                    хотим перейти
```

Правда, и здесь не все обошлось совсем гладко. Не удалось заставить сдвигать данные длиной в 1 бит. Но, тем не менее, еще раз хочется сказать, что работать с такой платой довольно удобно.
И еще немного. В библиотеке Xilinx Unified Libraries присутствуют элементы BSCAN_SPARTAN2, BSCAN_VIRTEX2. Таким образом, есть возможность получить доступ к проекту пользователя через тот же порт интерфейса JTAG, который используется и для загрузки микросхемы. А это значит, что можно использовать

USB-порт компьютера не только для JTAG-тестирования, но и для отладки и трассировки проекта пользователя.

Несколько слов о программах тестирования изделий

В качестве примера программы тестирования можно привести пакет программ фирмы Corelis [11]. Данные программы позволяют автоматически сформировать файл тестовых воздействий, проанализировать, насколько полно покрывается тестированием проверяемое изделие, выполнить тестовую программу и сообщить пользователю о результатах тестирования. Кроме автоматического режима тестирования существует ручной интерактивный режим. Результаты тестирования могут быть представлены как в виде таблиц с сообщениями об ошибках, так и в более наглядном виде. Результаты тестирования могут быть визуализированы в виде фрагмента PCB с указанием возможной точки проявления неисправности. Возможен режим, когда тестовые воздействия визуализируются примерно так же, как на экране логического анализатора.
Тесты для отдельных плат могут быть объединены в тесты для всего проверяемого изделия. Кроме программ тестирования в пакет входят программы In-System Programming. Они позволяют программировать Flash-микросхемы непосредственно в устройстве.

Заключение

Кроме тех программ, которые представлены в статье, существует достаточно большое количество программ, позволяющих тестировать изделия. На страницах журнала нет возможности описать или хотя бы перечислить их все. Могу сказать только об одной из них, а именно о той, которую хотел получить наш «самый главный». Правда, надо честно сказать, что в две недели уложиться так и не удалось, но зато через пару месяцев благодаря этой программе удалось «отловить» перемычку с сигнального провода на землю. Перемычка эта находилась где-то внутри платы между слоями и давала сопротивление в 2,5 Ом. Поэтому после загрузки FPGA, когда стандарт выводов становится другой, она была совершенно не «видна» для «ручных» методов тестирования. Разница здесь в том, что при автоматическом тестировании можно задать больше тестов, выбрать различные форматы выводов и т. д., так как для автоматического тестирования лишний час или два при проверке десятков тысяч цепей на каждой плате никак не сказываются на общей производительности фирмы.
И еще хочется добавить. В последнее время в российских изданиях появляется все больше предложений об изготовлении печатных плат и о контрактной сборке. А это значит, что проблемы автоматического тестирования станут более актуальными и для российских разработчиков.

Литература

1. IEEE Standard Test Access Port and Boundary-Scan Architecture. IEEE Std 1149.1-2001.
2. Платунов А. Е., Постников Н. П., Чистяков А. Г. Механизмы граничного сканирования в неоднородных микропроцессорных системах // ChipNews. 2000. № 10. **http://lmt.cs.ifmo.ru/article_chip_news.html**.
3. Рустинов В., Городецкий А. «Разделяй и властвуй» — принцип граничного сканирования // ChipNews. 2001. № 6. **http://chipnews.gaw.ru/html.cgi/arhiv/01_06/stat-3.htm**.
4. Boundary Scan Coach. GOEPEL Electronic. **http://www.goepel.com/**.
5. Design and Java Applets Support for an Asynchronous-Mode Learning of Digital Test. **http://www.pld.ttu.ee/applets/**.
6. **http://www.national.com/appinfo/scan/index.html**.
7. Chapter 9. IEEE 1149.1 (JTAG) Boundary-Scan Testing for Stratix II Devices. Altera. **www.altera.com**.
8. Dmitry Kuznetsov. JTAG Boundary-Scan Test — introduction. **http://www.orc.ru/~dkuzn/j_intro.htm**.
9. **http://jtagtools.sourceforge.net/download.html**
10. **http://www.universalscan.com**.
11. Boundary-Scan Test and In-System Programming Software. Corelis. **http://www.corelis.com/products/Test_Software.htm**.
12. Serial Vector Format Specification. ASSET InterTech, Inc. Texas Instruments, Inc. **www.asset-intertech.com/support/svf.pdf**.
13. Xilinx Unified Libraries. libguide.pdf.