СРАВНЕНИЕ ПРОЦЕССОВ РАЗРАБОТКИ ПРИЛОЖЕНИЙ В СИСТЕМАХ JTAG PROVISION И JTAG LIVE STUDIO. Часть 2

АЛЕКСЕЙ ИВАНОВ, консультант, JTAG, *Alexey@jtag.com*

В первой части статьи, вышедшей в предыдущем номере «Электронных компонентов», мы рассказали о разработке некоторых видов приложений периферийного сканирования в среде JTAG ProVision. Для примера была взята JT2156 (учебная плата JTAG Technologies), и описана генерация теста межсоединений, тестов SDRAM DDR II, кварцевого генератора, приемо-передатчика Ethernet, а также создание итоговой тестовой последовательности. Во второй части статьи мы рассмотрим процесс создания этих приложений для той же платы в недавно анонсированном бюджетном пакете JTAG Live Studio.

JTAG Live Studio дает возможность применять периферийное сканирование практически любой организации за счет своей доступности. В состав базового комплекта входят следующие компоненты:

- контроллер JTAG Live Controller с одним TAP-портом (см. рис. 1);
- программа Buzz, которая, если ее использовать отдельно, является бесплатной и скачивается с [1];
- программа BuzzPlus, позволяющая найти путем последовательного опроса связи отдельно выбранного вывода любого JTAG-компонента с другими JTAG-компонентами;
- программа AutoBuzz, изучающая все связи JTAG-компонентов между собой;
- программа Script (аналог JFT из ProVision). Используется для написания скриптов для тестирования кластеров, окружающих JTAG-компоненты;
- программа Clip (аналог ActiveTest из ProVision) для ручного написания тестовых векторов.

Рис. 1. **Контроллер JTAG Live с одним TAP-портом и USB-интерфейсом**

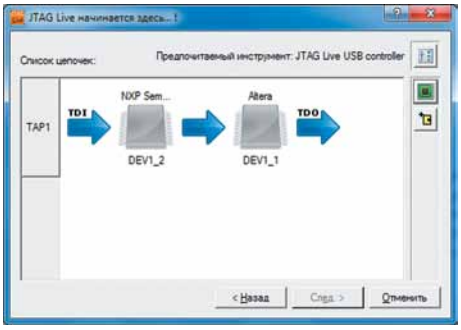

Рис. 2. **Автоматическое определение JTAG-цепочек в JTAG Live Studio**

На семинарах по периферийному сканированию автору этой статьи часто задают типичный вопрос «А как пишутся тесты?». Если семинар посвящен JTAG ProVision, автору всегда приходится поправлять, отвечая, что в ProVision тесты генерируются автоматически на основе схематики, BOM, моделей компонентов из библиотек. Поэтому фраза «написание тестов» не совсем уместна. Если мы окунемся в изучение JTAG Live Studio, то увидим, что здесь как раз большинство тестов именно «пишутся»! Но об этом — ниже.

ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

В отличие от JTAG ProVision, JTAG Live Studio не имеет автоматической генерации тестов. Соответственно, информация о межсоединениях из САПР не требуется. Как следствие, при тестировании не фигурируют названия цепей и компонентов кластеров.

Для работы потребуются только BSDL-файлы, которые, как уже было упомянуто, загружаются с сайтов производителей компонентов. Единственное, что требуется для дальнейшей работы, — это информация о цепях JTAG-цепочек. В ProVision эта информация

берется из нетлиста, а в Studio она вводится вручную или определяется автоматически при подключении контроллера к тестируемой плате (см. рис. 2).

После окончания подготовительного этапа можно приступать к созданию тестов.

Для создания тестов компонентов, окружающих компоненты с поддержкой периферийного сканирования (т.н. кластеров) необходимо также найти документацию на них (datasheet), поскольку готовые модели для автоматической генерации в JTAG Live Studio отсутствуют.

СОЗДАНИЕ ТЕСТА МЕЖСОЕДИНЕНИЙ

В JTAG Live Studio тест межсоединений компонентов с поддержкой периферийного сканирования производится на основе предварительного сканирования связей у заведомо исправной платы. Затем полученные связи сохраняются, и с этой картой сравниваются все другие платы. При обнаружении лишних или отсутствующих связей такие несоответствия могут рассматриваться как дефекты. При этом система не выдает информацию о цепях, а только выделяет цветовой маркировкой те связи, где найдено несоответствие (см. рис. 3). Для создания теста используется программа AutoBuzz в пакете Studio.

Имеется также возможность изучения связей из нетлиста, экспортированного из САПР. Но при этом следует помнить, что система JTAG Live Studio изучит только связи компонентов с поддержкой периферийного сканирования. Не следует путать этот процесс с импортированием схематики, как это происходит в ProVision. При наличии связующей логики она не учитывается автоматически, и по умолчанию ее присутствие рассматривается как обрыв. Такие недочеты необходимо править «вручную». Дальнейший процесс тестирования целиком повторяет сравнение с картой соединений заранее исправной платы за исключением того, что вместо нее использовался нетлист. Результаты тестирования выглядят как на рисунке 3.

Следует обратить внимание и на преимущество такого подхода к тестированию межсоединений: инструмент AutoBuzz позволяет изучать межсоединения JTAG-компонентов плат, на которые отсутствует информация из САПР (нетлист).

СОЗДАНИЕ ТЕСТА ОЗУ SDRAM DDR2

Напомним, что на тестируемой плате JT2156 установлена микросхема памяти SDRAM DDR2. В JTAG Live Studio для создания приложения для тестирования такого кластера необходимо создать приложение типа Script. Далее следует открыть редактор кода, в котором требуется написать код на языке Python.

Перед созданием скрипта необходимо изучить документацию на микросхему SDRAM MT47H64M8. Для создания скрипта требуется такая информация как шина адреса, шина данных, управляющие сигналы, процесс чтения и записи.

Также при подготовительном этапе требуется изучить принципиальную схему платы JT2156, чтобы понять, с какими выводами компонентов с поддержкой периферийного сканирования соединены порты ОЗУ (следует помнить, что ни выводов самой ОЗУ, ни названий цепей в проекте нет, так как нетлист фактически не импортируется).

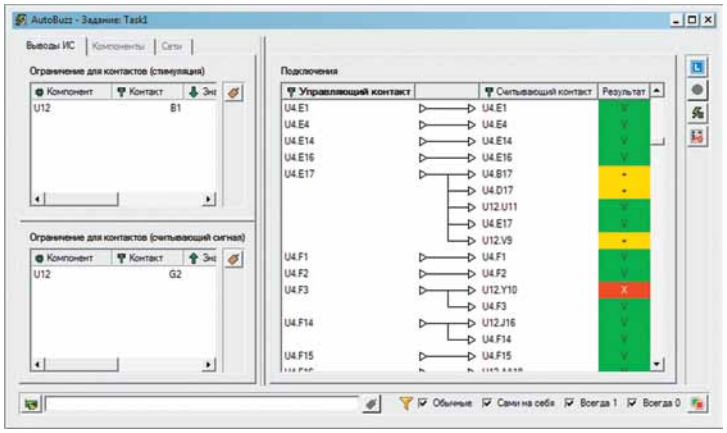

Рис. 3. **Окно программы AutoBuzz. Зеленым цветом показаны совпадающие соединения, желтым — лишние, красным — отсутствующие**

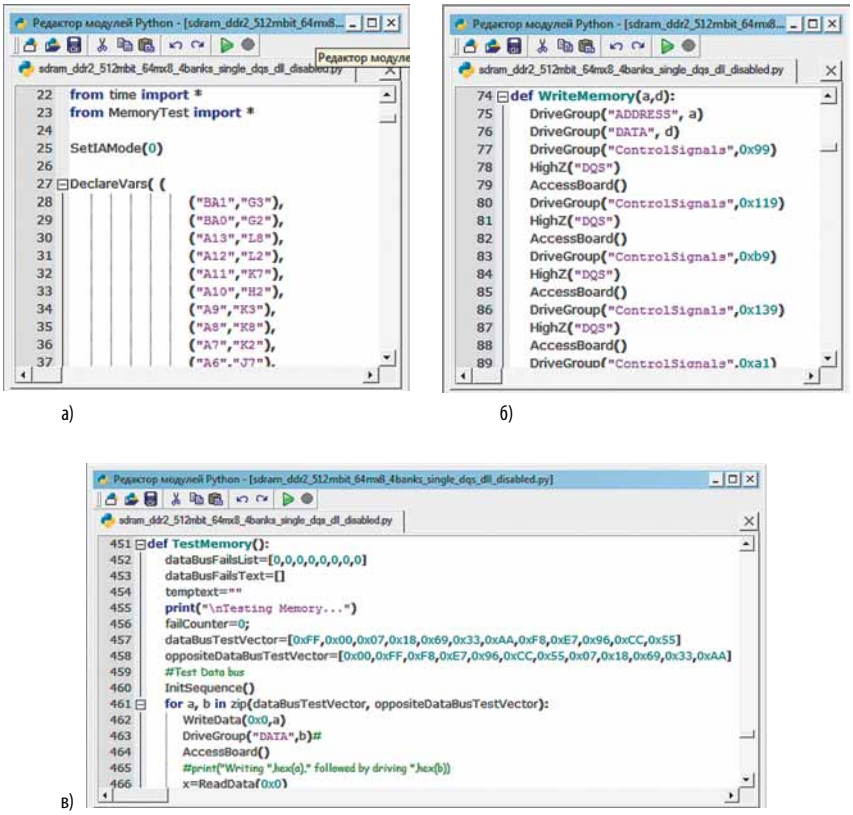

Рис. 4. **Создание теста SDRAM DDR2**: а) привязка портов микросхемы ОЗУ (переменных) к выводам JTAG-компонента; б) фрагмент процедуры записи данных в ОЗУ; в) фрагмент процедуры тестирования памяти

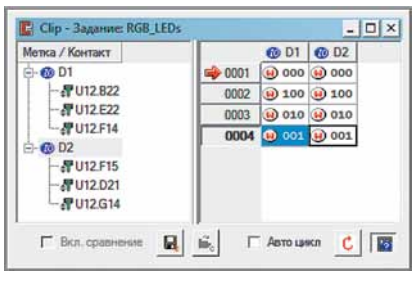

Рис. 5. **JTAG Live Clip. Тестирование светодиодов**

Первый этап заключается в создании модели (модуля) тестирования ОЗУ на языке Python. Эта модель должна содержать описание портов и процедур инициализации, записи, чтения и созданного на основе этих функций алгоритма тестирования. Эта информация, в свою очередь, подчеркивается из документации на микросхему.

Первая часть модуля должна содержать привязку выводов микросхемы ОЗУ к названиям функциональных портов (см. рис. 4а). Эту информацию можно извлечь из документации на микросхему MT47H64M8 в части, которая касается корпуса BGA. Далее необходимо объявить группы выводов, составляющих шины адреса, данных и контрольных сигналов с помощью стандартных библиотечных процедур DeclareGroup.

Теперь можно приступить к созданию основных функций, которые будут использоваться при тестировании ОЗУ. Первая из них — запись данных. Для того чтобы понять процесс записи, следует изучить документацию на MT47H64M8; наша задача — воспроизвести последовательности управляющих сигналов на выводах JTAG-компонентов, подключенных к памяти. Отрывок из описания процедур записи показан на рисунке 4б: WriteData (a, b) — это функция, которая может использоваться впоследствии для записи данных по выбранному адресу, b — это любые данные. Точно также, просматривая документацию, можно написать функции чтения из ОЗУ и ее инициализации: ReadData (a) и InitSequence ().

Далее с помощью созданных функций пишется код для, собственно, проверки памяти. Очевидно, что проверка должна заключаться в последовательной записи и чтении данных. Рисунок 4в показывает отрывок из данного скрипта.

После того как тестовый модуль на микросхему MT47H64M8 готов, можно приступить к написанию основной программы.

К программе мы также привязываем созданный ранее модуль, чтобы использовать оттуда функции в основном скрипте. Поскольку в проекте JTAG Live Studio отсутствует информация о связях (нет импорта нетлиста), то программе необходимо знать, к каким выводам JTAG-компонентов подключены выводы ОЗУ. При импорте модуля Python программа покажет все порты ОЗУ, которые с использованием схемы платы JT2156 необходимо связать при помощи контекстного меню с выводами ПЛИС (информацию о выводах ПЛИС программа извлекает из BSDL-файла). После этой операции можно приступать к написанию основной программы, в которой, собственно, будет содержаться лишь одна строчка U19.TestMemory (), вызывающая процедуру тестирования памяти, которая ранее была написана программистом. Результат тестирования будет выводиться в консоль.

Тестовый модуль, созданный для ИС MT47H64M8, может использоваться в дальнейшем и в других проектах, содержащих такой же тип оперативной памяти.

Данный пример демонстрирует, что, несмотря на отсутствие моделей и автоматической генерации, JTAG Live Studio все же позволяет создать тесты даже для памяти DDR2. При этом, несмотря на трудоемкость, у такого подхода имеется некоторая прелесть: программист волен сам задавать любые проверки в любом объеме. Следовательно, язык программирования Python имеется и в JTAG ProVision под названием JFT. Кстати говоря, все приложения, написанные на Python в пакете Studio могут впоследствии выполняться и в JTAG ProVision, и это еще одно преимущество.

СОЗДАНИЕ ТЕСТА КВАРЦЕВОГО ГЕНЕРАТОРА X1

Создание приложения для тестирования генератора также осуществляется с помощью программного модуля Script в JTAG Live Studio. Перед созданием скрипта следует изучить принципиальную схему платы JT2156, чтобы понять, с какими выводами компонентов с поддержкой периферийного сканирования соединен вход генератора, имеются ли у него активные сигналы включения. Всю эту информацию необходимо учесть, т.к. в коде программы будут использоваться именно выводы JTAG-компонентов, а не осциллятора.

После подготовительного этапа можно приступать к написанию программы тестирования. Перечислим основные этапы:

DeclareVar(«CLKOUT»,»U12.G2») — привязка выхода генератора к выводу G2 микросхемы U12 (поддерживающей периферийное сканирование).

Дальнейший код зависит от фантазии программиста, но в целом должен содержать два цикла для проверки изменения логического значения на входе G2 микросхемы U12. Рекомендуемое количество итераций: около 1000. Для детектирования логического уровня на входе G12 можно использовать функцию GetVar(«CLKOUT»). Это предустановленная функция, которая возвращает значение на выводе, указанном в качестве аргумента. Данная библиотека устанавливается вместе с пакетом Studio. Затем проверяется изменяемость результата, полученного функцией GetVar, и, если данные изменяются, пишется скрипт, который сообщает о работе генератора.

СОЗДАНИЕ ТЕСТА ETHERNET ПРИЕМО-ПЕРЕДАТЧИКА KSZ8041

Такой сложный кластер как приемо-передатчик Ethernet в JTAG Live Studio также тестируется с помощью программы, написанной на языке Python.

Перед созданием скрипта необходимо изучить документацию на микросхему приемо-передатчика KSZ8041. Для создания скрипта требуется знать принцип работы, входные и выходные шины, идентификационные коды. По мере изучения документации пользователь самостоятельно определяет, какие проверки можно осуществить.

Также на подготовительном этапе требуется изучить принципиальную схему платы JT2156, чтобы понять, с какими выводами компонентов с поддержкой периферийного сканирования соединены порты приемо-передатчика.

Предположим, что пользователь решил считать идентификатор PHY. Эту информацию можно найти в документации на микросхему. Кроме данной информации необходимо изучить алгоритмы доступа к данным регистрам; данных о том, где содержатся идентификаторы, недостаточно.

На основе полученных из документации данных можно приступать к написанию кода программы. Первым делом, необходимо описать порты микросхемы KSZ8041 и связать их с выводами компонента, поддерживающего периферийное сканирование. Информацию о связях можно почерпнуть из схемы платы.

В дальнейшем на основе определенных ранее переменных можно создавать функции уже с использованием названий выводов приемо-передатчика, а не компонента с поддержкой периферийного сканирования, с ним связанного. Для корректного определения этих функций необходимо тщательно изучить временную диаграмму записывающих и считывающих стробов в документации на микросхему KSZ8041.

Следующий этап — построение функций записи и чтения на основе определенных ранее стробов. Требуется детально изучить механизм записи и чтения в PHY. Как минимум, следует описать, например, процедуры формирования фрейма записи и фрейма чтения.

Сам скрипт тестирования должен начинаться с процедуры сброса для

установки адреса и затем самой установкой. Используя определенные выше функции, можно считать идентификаторы с помощью алгоритма, подобного приведенному ниже:

```
PHY_Identifier_Reg1=ReadFrame(My_PHY_Address,'00010')
PHY_Identifier_Reg2=ReadFrame(My_PHY_Address,'00011')
print(«\nPHY ID Code 1 is:»,hex(PHY_Identifier_Reg1))
print(«\nPHY ID Code 2 is:»,hex(PHY_Identifier_Reg2))
```

Следует заметить, что считанные идентификаторы отобразятся на экране ПК, однако для присваивания тесту атрибута «Прошел/Не прошел» необходимо доработать алгоритм, вставив процедуру сравнения и требуемые системные флаги.

ТЕСТИРОВАНИЕ СВЕТОДИОДОВ

Следует упомянуть еще один вид теста, процесс создания которого, в отличие от описанных выше, одинаков и в JTAG ProVision, и в JTAG Live Studio. Это — создание небольших пользовательских цифровых векторов. Именно поэтому мы оставили данное описание на вторую часть статьи.

Для примера возьмем тест светодиодов, хотя подобным образом могут тестироваться и многие другие узлы. На плате JT2156 имеются восемь трехцветных светодиодов. При тестировании платы для этих компонентов задача предельно проста: необходимо зажечь их и визуально проконтролировать. Поскольку за каждый цвет отвечает отдельная цепь, следует на каждой из трех цепей по очереди подать единицу, и так же — для каждого светодиода.

Эта задача решается в JTAG ProVision программой ActiveTest, а в Studio — ее аналогом, программой Clip. Различие состоит только в том, что в ProVision могут помимо выводов JTAG-компонентов фигурировать названия цепей: пользователь может выбрать любую цепь (для светодиода платы JT2156 — например, LED1_R) и поставить на ней 0 или 1. ActiveTest и Clip позволяют не только устанавливать данные на цепях с помощью компонентов с поддержкой JTAG, как в примере со светодиодами, но и считывать их. Так можно организовать тест небольшой логической микросхемы, входы и выходы которой соединены с компонентами, поддерживающими периферийное сканирование.

На рисунке 5 показан пример тестовых векторов для светодиодов D1 и D2 в программе Clip; в ActiveTest тест выглядеть схожим образом.

СОЗДАНИЕ ТЕСТОВОЙ ПОСЛЕДОВАТЕЛЬНОСТИ

Если у компании, использующей JTAG Live Studio, появилось желание использовать эту систему для производственных целей, то неизбежно и желание организовать общую тестовую последовательность, запускаемую как единую операцию. Как мы помним из первой части статьи, в JTAG ProVision такие последовательности создаются с помощью специального секвенсора с графическим интерфейсом. Приложения для тестирования и программирования перетаскиваются указателем мыши в окно секвенсора в необходимом порядке.

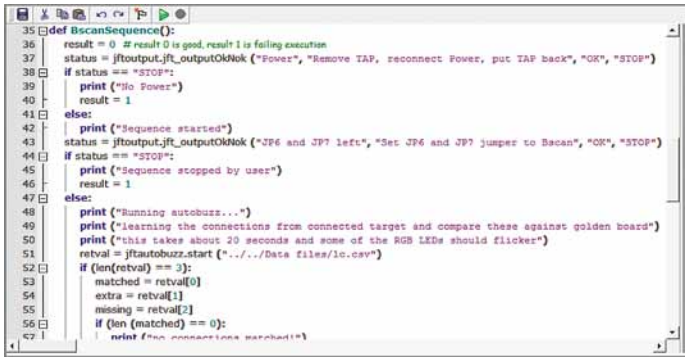

Рис. 6. **Создание тестовой последовательности с помощью скрипта на языке Python**

Следует заметить, что в JTAG Live Studio нет специализированной оболочки для составления последовательностей. Однако наличие встроенного языка программирования Python делает возможным любую задачу, в т.ч. данную.

При установке JTAG Live Studio на ПК вместе с языком Python устанавливаются различные библиотеки, в число которых входят и библиотеки для запуска приложений AutoBuzz (режим сравнения), и для работы с проигрывателями SVF и JAM. Используя созданные ранее модули-подпрограммы для тестирования кластеров и вызов AutoBuzz в одном скрипте, можно создать хорошее подобие тестовой последовательности. Более того, в данном алгоритме можно, например, запрограммировать ПЛИС с помощью библиотеки вызова плееров SVF или JAM.

Следует заметить, что Python — это открытый язык программирования, для которого существует много свободно распространяемых библиотек. Среди них — Tkinter, готовая библиотека для создания графического интерфейса пользователя. Таким образом, пользователи JTAG ProVision и JTAG Live Studio благодаря Python создают, по сути, любой интерфейс для запуска тестов при наличии навыков программирования. Это очень важное преимущество, которого не было бы, если бы использовался какой-нибудь узкоспециализированный, разработанный только для ProVision или Studio, язык программирования. На рисунке 6 показан фрагмент скрипта, описывающего процедуру тестовой последовательности.

В качестве резюме приведем таблицу 1 со сравнением функций JTAG ProVision и JTAG Live Studio. Как видно из таблицы, практически все функции пакета Studio присутствуют и в ProVision. Исключение составляет программа AutoBuzz, которая работает только в среде JTAG Live. Хочется назвать Studio урезанной версией

Таблица 1. **Основные функции систем JTAG Live Studio и JTAG ProVision**

	JTAG Live Studio	JTAG ProVision
Автоматическая генерация теста JTAG-цепочки	+	+
Автоматическая диагностика дефектов теста JTAG-цепочки		+
Анализ схематики, избежание конфликтов, предупреждения		+
Автоматическая генерация теста межсоединений по CAD-данным		+
Автоматическая диагностика дефектов теста межсоединений		+
Тест сравнения связей платы с заранее исправной платой (AutoBuzz)	+	
Автоматическая генерация тестов ОЗУ по библиотечным моделям		+
Автоматическая диагностика дефектов теста ОЗУ		+
Автоматическая генерация тестов логики по библиотечным моделям		+
Создание тестов кластеров на языке Python	+	+
Автоматическая генерация приложений для программирования ПЗУ		+
Автоматический расчет тестового покрытия по CAD-данным		+
Определение тестового покрытия созданных тестов		+
Работа с микросхемами, поддерживающими стандарт IEEE 1149.6		+
Ручное создание тестовых векторов в графическом редакторе	+	+
Визуализация тестового покрытия на топологии и схематике		опция
Работа с ядрами микроконтроллеров	опция	опция

ProVision, но это было бы не совсем корректно. Дело в том, что в ProVision присутствует *генерация* приложений, а в Studio — нет. При этом имеется необходимая среда для быстрого создания простых тестов: например, чтобы выставить на выводах JTAG-компонента любую последовательность нулей и единиц, нет необходимости писать коды и изучать JTAG-регистры установленных на плату микросхем. Импортированные в Studio BSDL-файлы уже поставляют всю необходимую информацию системе, а пользователю требуется лишь перетащить выводы в созданный тестовый вектор.

Изучив невероятно простой язык Python, можно создавать более сложные тесты, к примеру, тесты кластеров. Вместе с Python устанавливаются и библиотеки, позволяющие создавать шины из выводов JTAG-компонентов, устанавливать и считывать с них данные и т.д. Имея такой набор функций, можно написать алгоритм тестирования почти любого цифрового устройства, подключенного к JTAG-компоненту. Наличие библиотек работы с SVF и JAM позволит при необходимости прошить конфигурацию в ПЛИС. А программный модуль AutoBuzz позволит протестировать связи между компонентами с поддержкой периферийного сканирования.

ЛИТЕРАТУРА

1. www.jtaglive.ru.