Два подхода к тестированию кластеров в технологии периферийного сканирования

Технология периферийного (граничного) сканирования позволяет нам независимо от функций ядра соответствующей микросхемы управлять ее выводами, используя всем известный интерфейс JTAG. Сам интерфейс и архитектура периферийного сканирования (дополнительные тестовые регистры) закреплены в стандарте IEEE 1149.1, и, соответственно, если микросхема поддерживает данный стандарт, то у нас есть отличная возможность «заглянуть» в недоступные области собранного печатного цифрового узла без необходимости применять измерительное оборудование, такое как мультиметр, осциллограф или логический анализатор. Все эти приборы требуют физического доступа к цепям или выводам платы, а периферийное сканирование использует встроенные тестовые ячейки, назначенные для большинства функциональных выводов ИМС. На сегодня десятки тысяч процессоров, ПЛИС, контроллеров и СБИС различного назначения поддерживают стандарт IEEE 1149.1. При этом существует не только возможность тестировать связи между компонентами с поддержкой периферийного сканирования, но и при помощи вышеупомянутых ячеек получить доступ к функциональной логике, окружающей эти компоненты.

Алексей ИВАНОВ
Alexey@jtag.com

Про тест межсоединений и цепей, окружающих компоненты с поддержкой стандарта IEEE 1149.1, и про автоматическую генерацию векторов для этих целей было написано в [1]. Напомним, что математика для генерации тестов использует анализ межсоединений, основываясь на электронной схематике из САПР. Очевидно, что для тестирования связей с периферийными компонентами также существуют средства автоматической генерации тестов. В этой статье речь пойдет о двух принципиально разных подходах к созданию тестовых приложений для функциональной логики, окружающей компоненты, поддерживающие периферийное сканирование. Оба подхода реализованы в системе проектирования тестовых приложений JTAG ProVision компании JTAG Technologies.

Определение «кластера»

С точки зрения технологии периферийного (граничного) сканирования кластер — это любой функциональный компонент платы или группа компонентов, не поддерживающих IEEE 1149.1 (JTAG) (рис. 1). Часто случается так, что выводы кластера, который представляет собой некое функциональное устройство, соединены с JTAG-компонентами. Фактически тест кластера представляет собой тестовые воздействия на него при помощи векторов, загружаемых в регистры периферийного сканирования окружающих ИС с JTAG-логикой. При этом нужно понимать, что во время такого теста сам JTAG-компонент не использует функции своего ядра, оперируя регистром периферийного сканирования и интерфейсом JTAG, вдвигая и выдвигая по нему тестовые векторы. Тестируемый же «кластер», напротив, работает в функциональном режиме, отвечая на воздействия со стороны выводов компонента с поддержкой периферийного сканирования. Типичными «кластерами» являются ОЗУ, ПЗУ, логические микросхемы, интерфейсные и другие устройства, не поддерживающие периферийное сканирование.

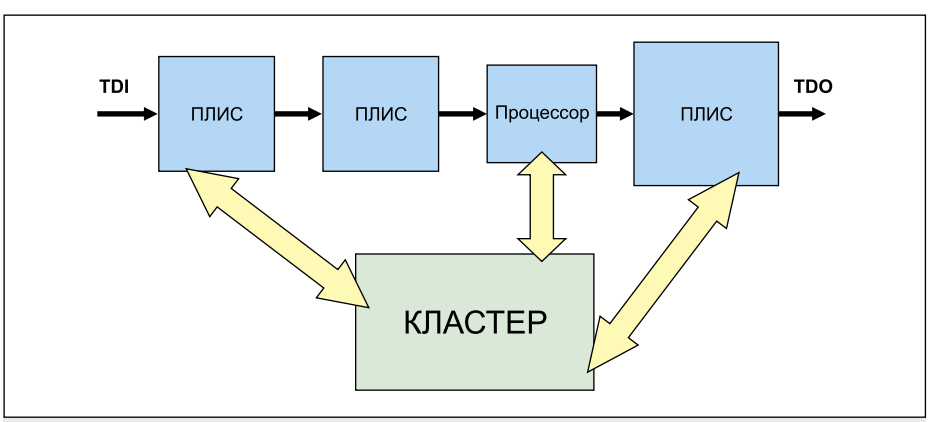

Рис. 1. Кластер с точки зрения технологии периферийного сканирования

Как показывает практика, за счет тестов кластеров получается наибольшая часть тестового покрытия изделия. Даже если на тестируемой плате всего один компонент, поддерживающий периферийное сканирование, он может быть окружен всевозможными

кластерами, за счет теста связей с которыми покрывается добрая половина цепей и выводов.

Классический подход к генерации на основе моделей

С момента своего появления система JTAG ProVision оперировала при генерации кластерных тестов подходом, основанным на функциональных моделях кластеров. На данный момент в комплект программного обеспечения входит обширная библиотека различных моделей компонентов без поддержки периферийного сканирования. До появления JTAG ProVision программное обеспечение также использовало эту технику, однако там отсутствовала библиотека, и модели по шаблону нужно было создавать самому, чаще всего в виде текстовых файлов, но суть метода от этого не меняется. Модель на сегодня является универсальной структурой, и ее можно использовать в других проектах, если там будут содержаться компоненты такого же типа.

Что наиболее характерно для такого подхода — это практически полная неизменность алгоритмов генерации. Есть модель, описывающая функциональность кластера, и набор алгоритмов, создающих на основе модели и схематики тестовые векторы, которые выставляются с компонентов, поддерживающих периферийное сканирование.

Этот процесс представлен на рис. 2.

Несмотря на неизменность алгоритмов генерации, для разных типов кластеров они различаются, что и понятно: кластеры могут быть совершенно разными по функциональности. Поэтому и вид моделей также различается: например для логического элемента и ОЗУ. Если для логического элемента модель может представлять собой таблицу истинности, то для памяти такой подход не представляется удобным или даже возможным. Поэтому для микросхем ОЗУ модель представляет собой описание шин данных, адреса, управляющих сигналов и типа устройства (SRAM, SDRAM, DDR и т. д.). Остальное содержится не в модели, а в алгоритме генерации, который будет отличаться в зависимости от типа памяти. При этом алгоритм генерации построен на алгоритме работы самой памяти.

Пример модели для логического компонента декодера в системе JTAG ProVision показан на рис. 3.

Достоинство этого подхода неоспоримо — легкость и быстрота создания кластерных тестов. В современном программном обеспечении при наличии обширной библиотеки моделей или шаблонов для создания кластерного теста необходимо по возможности определить как можно больше моделей для компонентов из нетлиста, а затем просто выбрать их щелчком мыши, как показано на рис. 4. После выбора компонента из списка в считаные секунды происходит генерация тестовых векторов, и тест готов.

Рис. 2. Классический подход к генерации теста кластера на основе моделей

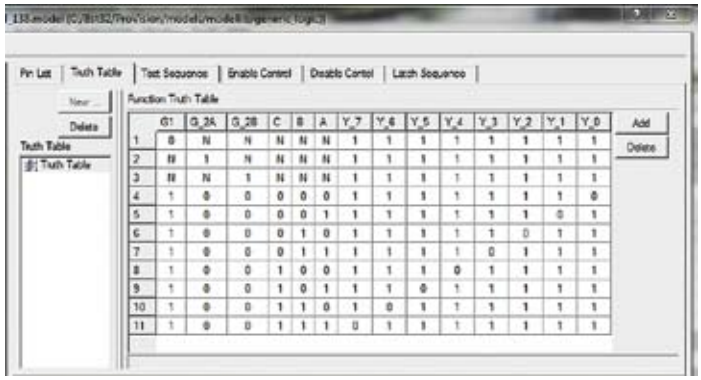

Рис. 3. Пример модели логического элемента в JTAG ProVision, используемой для генерации теста

К недостаткам можно отнести отсутствие гибкости в создании тестов. Алгоритм генерации, как правило, неизменен. Страдают от этого специалисты, которые любят протестировать или верифицировать дополнительные возможности кластера, а не только те, которые требуются для простого электрического контроля собранного печатного узла. Часто инженеры, например, задают вопрос: «А можно ли протестировать адресное пространство ОЗУ целиком при помощи модели?» Теоретически можно, но алгоритмы генерации в любых автоматических системах проектирования тестов выбирают только те сочетания адресных линий, которые в совокупности позволят локализовать во время будущего теста все возможные неисправности на линиях связи JTAG-компонента с ОЗУ. Нетрудно догадаться, что здесь отсутствует избыточность, и все адресное пространство не проверяется. Кроме того, большинству пользователей это не нужно: связи проверяются, память работает, для электрического контроля этого достаточно. Точно

Рис. 4. Автоматическая генерация кластерного теста в JTAG ProVision

так же, как и память, другие типы кластеров могут содержать гораздо бóльшую функциональность, которую инженеры хотели бы проверить, но, конечно же, в модели (а их десятки тысяч) всего не учтешь, да и алгоритмы генерации могут работать только со «знакомой» функциональностью.

Другой пример ограниченности данного подхода — это неоднозначность реакции кластера на входное воздействие, например, выход АЦП, который будет зависеть от поданного напряжения на плату. Неоднозначной может рассматриваться и реакция устройства, работающего «по готовности», когда, например, для выполнения следующей операции необходимо получить сигнал ***Ready***. Модельный подход предполагает определенный отклик на определенное воздействие. Даже в случае с тестированием памяти мы должны считать именно то, что записали по определенному адресу, это значение неизменно на исправном изделии, а если оно отличается, то это сигнал о дефекте.

Все эти и другие ограничения, которые возникают при генерации тестов на основе моделей, побудили JTAG Technologies создать дополнительный метод, который, с одной стороны, даст инженерам абсолютную гибкость и свободу в создании тестов, а с другой — будет использовать унифицированный язык описания, графический интерфейс и связь с информацией из схематики изделия. Так появился дополнительный программный продукт, полностью интегрируемый в JTAG ProVision, под названием JTAG Functional Test (сокращенно JFT).

Функциональный подход

Основное принципиальное отличие JFT от классического подхода — это то, что алгоритм автоматической генерации заменяется в данном случае на самостоятельно написанный программистом скрипт. Фирма JTAG Technologies выбрала за основу высокоуровневый язык программирования Python (Питон). Этот язык полностью интегрируется в среду JTAG ProVision, позволяя использовать для стимуляции воздействий те же BSDL-модели компонентов с поддержкой периферийного сканирования и цепи, которые использовались при автоматической генерации тестов межсоединений и кластеров. Предустановленные библиотеки с функциями позволяют использовать названия или номера выводов JTAG-компонентов и устанавливать или считывать с них значения, записывая их при необходимости в переменные, а также оперировать сразу группами выводов:

```
DeclareGroup ("4switches",("D500.22", "D500.24", "D500.19", "D500.17"))
DeclareGroup ("4upperLeds",("D600.73", "D600.1", "D600.83", "D600.5"))
i =1
while i ==1:
        Result = GetGroup ("4switches")
        DriveGroup ("4upperLeds",Result)
        print ("Value of the 4 switches is now:", 0xF-Result)
        i = GetVar("D500.11")
```

В приведенном выше отрывке программы в переменную ***Result*** считывается группа значений с выводов микросхемы D500. Устройство D500 в данном случае является «локомотивом» теста, то есть поддерживает периферийное сканирование. Цепи, которые заходят на эти выводы, соединены также с кнопками на тестовой плате, в зависимости от нажатия которых будут меняться и значения, которые мы считываем в регистр периферийного сканирования D500 при помощи функции ***GetGroup***. Затем считанные значения будут установлены уже на выводах другой микросхемы с периферийным сканированием D600, при этом к данным выводам подключен светодиодный дисплей. Таким образом, эта программа позволяет отобразить на светодиодах, которые схемотехнически не связаны с клавиатурой, значения состояния кнопок, однако эти манипуляции производятся только за счет регистров периферийного сканирования и JTAG-интерфейса.

Можно заметить, что мы также ввели переменную ***i***, при помощи которой можно осуществлять прекращение цикла, например, связать ее еще с какой-нибудь кнопкой, которая будет выполнять функцию сброса. То есть при нажатии кнопки сброса на тестируемой плате скрипт автоматически перестанет выполняться. Функция ***print***, как нетрудно догадаться, выводит на экран компьютера значение кода, выставленного кнопками. Следует заметить, что функции ***DeclareGroup***, ***GetGroup***, ***GetVar*** и ***DriveGroup*** уже предустановлены вместе с JFT. Таким образом, при выполнении этого скрипта на тестовой плате нажатие каждой из кнопок будет отображаться на соответствующем светодиоде, а в консоли JFT будет примерно такая информация (в зависимости от нажатых кнопок):

```
Value of the 4 switches is now: 15
```

В самой программе, которую пишет программист, нет описания ни схемы, ни BSDL-моделей компонентов на плате, все это берется из готового проекта в JTAG ProVision. Название компонентов (D500 и D600) и номера их выводов берутся из сконвертированного нетлиста платы, а BSDL-файлы уже заранее определены.

Следует отметить, что в среде программирования JFT можно создавать и свои модули, которые можно использовать в последующих программах. Модуль представляет собой описание кластера и его функций. Пользовательские функции затем можно использовать в основной программе. Здесь проявляется еще одно достоинство данного подхода: если в классическом подходе генерации тестов кластеров на основе моделей кластер, как правило, может представлять собой только один компонент, привязанный к одной библиотечной модели конкретной микросхемы, то в JFT кластером может являться некий узел, состоящий из группы устройств и элементов: мы оперируем только выводами окружающих JTAG-компонентов.

На рис. 5 показан фрагмент программы на Python, использующей готовый модуль с прописанными функциями. Кластер, который описывает модуль (сам модуль

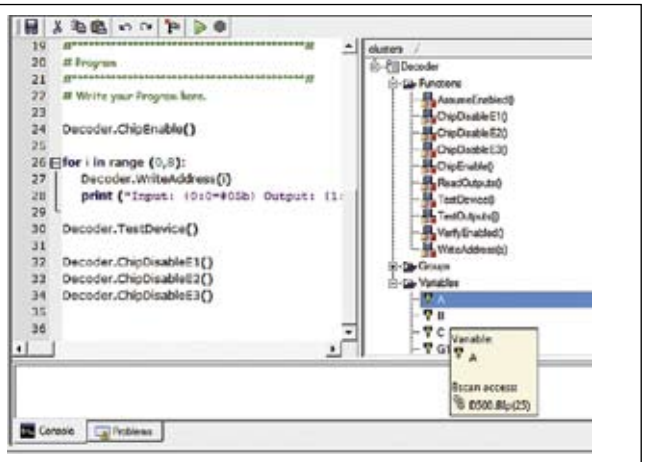

Рис. 5. Пример использования готового модуля-кластера в JFT

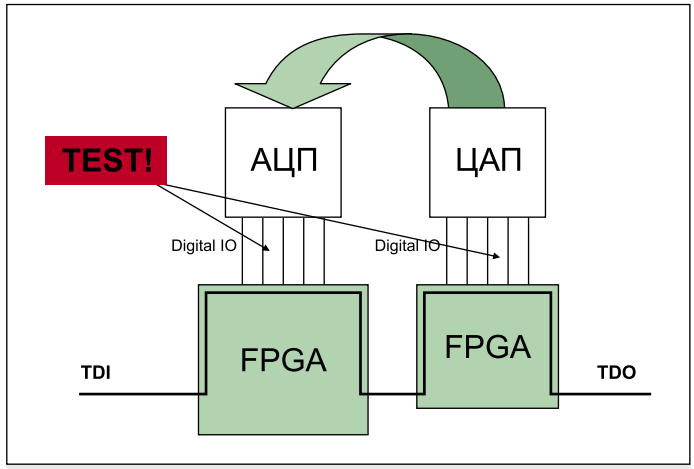

Рис. 6. Задача тестирования цепей АЦП

Таблица. Использование классического подхода при тестировании кластеров и JTAG Functional Test: области применения

	Классический подход, основанный на генерации тестов по моделям	JTAG Functional Test (JFT)
Наиболее подходящие типы кластеров для тестирования	Память. Логика (буферы, MUX, комбинаторная)	Устройства со сложной инициализацией (if... then... else). Аналогово-цифровые участки схемы
Диагностика дефектов	Обычно генерируется вместе с тестовым приложением	Пишется вручную как часть программы
Свобода действия	Минимальна. Если изменить модель ОЗУ, тест сгенерируется некорректно или выдаст ошибку. Алгоритм генерации неизменен	Максимальна. Программист может добавлять или удалять различные дополнительные проверки, измерения и т. д.

здесь не приведен) мы назвали Decoder. На экране виден список его функций, а также выводов (переменных). Эти выводы описаны в тексте самого модуля в виде переменных. Затем, при использовании модуля в JFT-приложении их можно связать с выводами устройства, поддерживающего периферийное сканирование вручную или автоматически, если название модуля совпадает с каким-либо компонентом из нетлиста. В самой программе мы просто используем название модуля-кластера и какую-либо функцию, например ***Decoder.WriteAddress(i)***.

Другим примером может послужить тестирование выхода АЦП. Представим, что перед тестовым инженером стоит задача, показанная на рис. 6. Необходимо протестировать с максимально возможным обнаружением дефектов связи АЦП и FPGA. Здесь можно пойти разными путями: стимулировать аналоговое воздействие на АЦП с ЦАП, который может находиться на той же плате или, например, в плате оснастки. Или можно использовать внешний источник питания. При этом аналоговым воздействием хотелось бы управлять в рамках того же самого тестового приложения, то есть само приложение должно, в идеале, быть способно работать не только с JTAG-контроллером, но и с внешними приборами.

Кроме того, здесь может иметь место неопределенность — ведь аналоговое напряжение может иметь погрешность и, неизбежно, в младших разрядах цифрового выхода будут иметь место вариации. То есть напрашивается вариант считывания (конечно же, при помощи регистра периферийного сканирования FPGA) цифрового кода в переменную и проверки попадания ее в заданный диапазон. Второй момент — это то, что для нормальной локализации возможных дефектов желательно было бы протестировать цифровые цепи АЦП при разном входном напряжении. Третье — возможно, нам вообще захочется просто измерить напряжение на входе АЦП.

Вот пример одного из вариантов кода, который можно написать для этой ситуации:

```
vref = 4.096
# read voltage from ADC
voltage=U28.ReadVoltage()

print ('ADC value is 0x{0:2x}'.format(voltage))
# calculate analog voltage
analog_voltage = voltage/255 * vref

print ('Voltage is', analog_voltage)
# test for good or fail (expected is 2.5V +/- 10%)
if (analog_voltage > (2.5 * 0.9)) and (analog_voltage < (2.5 * 1.1)):
    print ('Measured voltage in range')
else:
    print ('Measured voltage not in range')
```

Здесь в переменную voltage считывается значение на выходе АЦП U28. Следует отметить, что программист в дополнение к основному коду написал также модуль под названием U28, в котором прописаны все функции, которые можно проделать с устройством АЦП, например функция ***ReadVoltage*** (сам модуль мы не приводим). Далее считанный код приводится в актуальное значение аналогового напряжения, которое проверяется на соответствие разрешенному диапазону.

Преимущество использования языка программирования Python заключается и в том, что в приведенном примере с АЦП мы можем управлять и любым внешним источником питания, подключенным к ПК, зная особенности его интерфейса. Таким образом, тестовый инженер сможет создать приложение, которое может одновременно управлять источником питания, устанавливая различные напряжения, и считывать полученные значения на выходе АЦП. Все приложения, использующие язык программирования Python, можно запускать в общем секвенсоре JTAG ProVision наряду с остальными тестами, созданными при помощи автоматической генерации, и приложениями для программирования. Кстати, нетрудно догадаться, что приложения для программирования флэш-ПЗУ тоже можно создавать на Python, однако обширная библиотека JTAG ProVision и техподдержка по созданию новых моделей флэш-памяти позволяют делать это с гораздо меньшими трудозатратами при помощи автоматической генерации.

Еще одна особенность использования JFT — это возможность работы с любыми нестандартными JTAG-регистрами и командами, что позволяет использовать дополнительные возможности JTAG-интерфейса, даже если микросхема не поддерживает в полной мере стандарт IEEE 1149.1.

Общее сравнение двух подходов к тестированию кластеров и области их применения показаны в таблице. Судя по таблице, выбор метода зависит не только от типа кластера, но и от решаемого спектра задач, желаемой свободы творчества в создании тестов и автоматизации диагностики. Автоматическую генерацию тестов на основе моделей удобно использовать тогда, когда требуется быстрый универсальный метод создания приложений, процедура создания тестов занимает считаные минуты с учетом отладки. JTAG Functional Test — программное средство, дающее абсолютную гибкость при создании тестов, однако требует небольших временных затрат на программирование и отладку. ■

Литература

1. Иванов А. Инструменты для периферийного сканирования: тестирование и отладка цифровых узлов // Компоненты и технологии. 2010. № 9.