Микропроцессор своими руками-4. Как отладить встроенный в FPGA микроконтроллер?

Иосиф КАРШЕНБОЙМ
losifk@narod.ru

Введение

Один из моих читателей, Д. Шехалев (известный так же как des00) высказывал в письме следующий тезис: «Вот разработал я плату с DSP процессором и FPGA. Так вот, очень обидно порой становится, когда программист подключает к DSP на JTAG-порт свой отладочный инструмент и ему сразу становятся доступны все внутренние ресурсы процессора. У меня в FPGA проект ничуть не проще, а для отладки проекта такого удобного инструмента нет. И чтобы сделать проект отлаживаемым через JTAG, сил и, главное, времени не хватает». В этой статье будет показано, как сделать проект отлаживаемым через JTAG. Таким образом, она продолжит цикл статей, озаглавленный «Микропроцессор своими руками».

Автор благодарит за предоставленные материалы и оборудование фирму Inline, Российское представительство Agilent Technologies, фирму ЭФО, а также всех тех, кто помог ему в его нелегком деле.

Иллюстрация из книги «Сочинения Жуковского», издание 1905 года

Что будет предложено вниманию читателей в этой статье

На маленькой цветной заставке образно показано состояние разработчика на стадии отладки проекта. Особенно когда сроки «горят», а ошибку не удается найти. То есть проект не работает так, как задумано, а для поиска и локализации ошибки, как обычно, не хватает удобного отладочного инструмента. Такая проблема встает каждый раз, как только проект приходит к стадии отладки «железа». Причем именно эта стадия разработки проекта является наиболее трудоемкой и требует наивысшей квалификации разработчика. Какие же есть методики, позволяющие ускорить этот процесс?

Поскольку отладка для разных типов проектов имеет различную специфику, то целью данной статьи не является описание «универсального» способа отладочного инструмента. В рамках этой статьи автор поставил себе цель: показать, как производить отладку на примере софт-процессоров. И как для этой цели использовать порт JTAG в качестве порта общего назначения для приема и передачи данных в/из проекта пользователя. Таким образом, данная статья продолжает цикл статей, озаглавленный «Микропроцессор своими руками», см. [1–4] и статью о JTAG тестировании [5]. Специально для этого материала будут разработаны следующие проекты:

1. Проект встроенного отладочного блока. Язык описания — Verilog. Среда ISE — 8.1. Проект будет отлажен на плате Xilinx Spartan-3E Sample Pack. Микросхема — XC3S100E-TQ144.
2. Проект встроенного отладочного блока. Язык описания — Verilog. Среда Quartus. Проект будет отлажен на плате MAX2 — DEVKIT-1270. Микросхема — EPM 1270 F256C5.
3. Проект управляющей программы для работы с JTAG-портом. Среда — Borland C++ Builder 6.

Все эти проекты будут разработаны только как дополнения к данной статье. При их разработке не ставилась цель получить законченный коммерческий продукт. Поэтому некоторые части программ написаны наиболее простым способом, но как представляется автору, это будет более понятно начинающим. А для опытных разработчиков не составит особого труда модифицировать файлы этих проектов для своих потребностей. Главное, что при их помощи читатели смогут сами проверить работу JTAG-порта в режиме отладки их проекта. Проекты можно взять на сайте автора, и они будут выкладываться по мере готовности. Но, поскольку выполнить все проекты к моменту выхода статьи достаточно тяжело, то автор приносит свои извинения в случае их задержки. И еще одна просьба. У автора сейчас нет возможности набрать статистику по испытаниям данных проектов на разных платах и разных микросхемах. Поэтому если вы, уважаемый читатель, найдете что-либо интересное в данном вопросе, то не сочтите за труд написать автору письмо с вашими наблюдениями, дополнениями и рекомендациями.

Что значит «отладить»?

Для начала давайте уточним терминологию. Что значит «отладить» и как это происходит?

При этом необходимо еще одно уточнение. Отладку программного обеспечения (ПО) для микроконтроллера можно разделить на две части:

1. Отладка программного обеспечения на кросс-средствах.
2. Отладка на реально работающем изделии.

Отладка ПО на кросс-средствах известна и описана многократно. Поэтому в данной статье будем считать, что ПО для встроенного процессора уже отлажено на софт-симуляторах данного микропроцессора. Таким образом, речь не идет о полной отладке ПО. Речь в данном случае идет только о второй части, а именно — только о привязке ПО к реальному устройству, об исследовании характеристик «внешнего мира» данного микроконтроллера.

Разделим разные способы отладки по признаку работы в реальном времени:

1. Отладочные системы, позволяющие отлаживаемому устройству продолжительно работать в режиме реального времени.

2. Отладочные системы, позволяющие отлаживаемому устройству работать в режиме реального времени только кратковременно.
3. Отладочные системы, не позволяющие отлаживаемому устройству работать в режиме реального времени.

Как же отлаживают встроенный в FPGA микропроцессор? Практически так же, как и обычный однокристальный процессор.

Перед тем как начать рассматривать структуры отладочных средств, необходимо обсудить еще один вопрос. Тактовые частоты встроенных в FPGA микроконтроллеров уже перевалили за диапазоны в десятки мегагерц и теперь тактовые частоты в сотни мегагерц являются обычными рабочими частотами для многих кристаллов. А каналы вывода информации из FPGA в ряде случаев оказываются достаточно медленными. Поэтому для вывода информации из кристалла необходимо довольно значительное время. И чем больше информации мы хотим вывести, тем сложнее это будет сделать, что в свою очередь скажется на всем процессе отладки.

Вот почему настал тот момент, когда отладочное устройство должно переместиться из «внешнего мира» внутрь кристалла.

И еще один аспект разработки отладочных устройств необходимо рассмотреть сейчас. Инженерный труд — это всегда решение, которое необходимо принять при противоречивых условиях. Как не бывает абсолютных истин, так не бывает и абсолютных технических решений. Одно решение будет хорошо для одних случаев, но плохо для других. Поиск компромисса труден. Но именно он приводит к тому, что ваше изделие будет превосходить изделия конкурентов. Тот блок, который мы будем рассматривать в этой статье, выполняет вспомогательную роль при отладке. Он не влияет на потребительские качества основного изделия. Он нужен только на этапе настройки, отладки, опытной эксплуатации. Поэтому для таких вспомогательных блоков целевую функцию можно сформулировать так: «Нам нужен блок, который бы выполнял функцию отладки, но при этом, чтобы самого его не было…». Вот вам и первое противоречие. Исходя из того, что все реализации имеют свои достоинства и недостатки, мы постараемся рассмотреть каждую из предлагаемых реализаций с двух точек зрения. Что дает нам данная реализация и что она позволяет сделать с отлаживаемым изделием? Какими ресурсами за это надо заплатить и какие ограничения на работу изделия и на режим отладки необходимо наложить? И уже исходя из достаточно полного набора решений, читатель сам сможет выбрать то решение, которое он сочтет нужным применить в своем проекте.

О фразе: «У меня все работает»

Именно такую фразу можно услышать чаще всего в ответ на вопрос начальника о том, как обстоят дела у его подчиненного. В таком случае автор этой статьи всегда вспоминает другую фразу, которую когда-то говорил его начальник: «Все так, но… только трошечки не так». Если представить процесс разработки, как процесс заряда конденсатора, который протекает по экспоненте, то за одно «Тау» выполняется 30% задания, за 3 «Тау» мы подойдем к уровню 95% или, можно сказать, достигнем уровня ошибок в 5%. А вот до уровня ошибок в доли процента будем добираться за многие десятки этих самых «Тау».

Зачем приведены эти рассуждения? Процесс отладки считается завершенным, и устройство считается отлаженным только до уровня, определенного отладочным средством и методикой отладки. Можно отладить изделие в шаговом режиме, а можно и в режиме реального времени. И в том, и в другом случае изделие будет числиться в категории «У меня все работает». Просто трудоемкость отладки и требуемые для этих вариантов отладки ресурсы потребуются совершенно разные. И если в одном случае отладка в шаговом режиме будет вполне достаточна, то в другом случае такой вариант отладки будет совсем неприемлем.

А теперь любимое рассуждение автора о монете. Вполне естественно, когда вы подкидываете монету, она падает и встает точно на ребро. Ну, правда, если эта монета не «зависнет» в воздухе… Вы улыбаетесь, уважаемый читатель? А что же происходит при отладке? Все ли условия для поиска ошибок вы предусмотрели? Все ли эти условия охватывает отладочное средство? Выводит ли оно сообщения об этих ошибках? Какими ошибками можно пренебречь на данном этапе отладки? И что значит «все работает»? Давайте отвлечемся от микроконтроллеров и FPGA и перейдем от абсолютных истин к относительным. (Автор заранее просит прощения, если вы, уважаемый читатель, сочтете последующие высказывания и термины, отличающимися от того, что вы прочли в учебнике по философии.)

Так что же такое тестирование и когда можно считать, что мы получаем достоверный результат? Скорее всего, ответ на этот вопрос будет таким: «Мы устойчиво получаем результат, который вызывается определенным внешним воздействием, при том условии, что все другие внешние воздействия не изменяются или их изменение не оказывает влияния на получаемый результат». То есть речь идет о результате, который получается только относительно каких-то условий. Примером для данного рассуждения может быть серия из 100 попыток включить сгоревшую электролампочку. Можно смело утверждать, что во всех 100% тестов мы получим именно тот результат, который и ожидали, вне зависимости от окружающей температуры, влажности и напряжения электросети. Но еще раз необходимо повторить — этот результат будет достоверен только для вполне определенных условий. И нельзя абсолютно уверенно говорить о том, что не включенная в электросеть лампочка ВСЕГДА НЕ будет гореть. И если вы видели, как светится неоновая лампочка, которой медсестра проверяет прибор под названием «УВЧ», который находится в кабинете физиотерапии, то я думаю, что вы меня поймете. Неоновая лампочка светится, хотя провода к ней не подключены. Просто в этом случае действуют совершенно другие условия. Но вернемся к обычной электролампочке и к обычным условиям.

Гораздо сложнее получить достоверный результат испытаний для исправной электролампочки, так как в этом случае необходимо предусмотреть, чтобы довольно много параметров соответствовало норме. То же самое и с микроконтроллерами. Без сомнения, микроконтроллер гораздо сложнее электролампочки, поэтому у него гораздо больше возможностей НЕ работать! И, соответственно, гораздо сложнее определить те условия, при которых он будет работать, причем устойчиво. Для того чтобы упростить процедуру, принято тестирование всего изделия начинать с простейших тестов, и затем принимать решение о том, работает изделие по данному тесту или нет. Далее тестирование следует проводить поэтапно. На следующем этапе тестирования проверенная часть изделия считается исправной, и с ее помощью производится проверка других частей изделия. Тестирование постепенно охватывает все большие и большие части изделия, и так до тех пор, пока изделие не будет считаться полностью протестированным. Только после этого принимается решение о полной или частичной исправности проверяемого изделия. И только с учетом конкретных условий. Например, выглядящий вполне работоспособным измерительный тракт в приборе, который испытывается на лабораторном столе, может давать погрешность измерений гораздо меньше той, что предусмотрена его классом точности. Но в термокамере такой прибор может «удрейфовать» и выйти за пределы допустимой погрешности. Так работает этот прибор или нет? Все зависит от того, для каких условий работы он предназначен. Если для лабораторных, то прибор признается работоспособным. Если для индустриальных — не работоспособным.

Как же производится тестирование на практике? Вспомните, как происходит процесс включения вашего компьютера и запуск операционной системы. Компьютер тестируется постепенно, начиная от центра (от процессора) и постепенно охватывая периферию. Сначала процессор, далее подключается видеосистема в минимальном режиме, потом в графическом режиме и т. д. Точно так же необходимо тестировать и встроенные изделия. Первый тест — проверка на припайку по технологии граничного сканирования (Boundary Scan), далее минимальное отладочное средство для проверки линии связи, затем более полное и т. д. И чем больше условий мы

хотим протестировать, тем сложнее должно быть само отладочное средство. А если отладочное средство представляет собой сложную структуру, то и оно должно самотестироваться при включении и сообщать об ошибках в работе, возникающих при работе самого отладочного средства. Причем всегда необходимо принимать меры, направленные на то, чтобы само отладочное средство работало более надежно, чем проверяемое им изделие.

Можно сделать следующий вывод: для тестирования необходимо иметь возможность выбрать нужный аппаратно-программный инструмент, для того чтобы с наименьшими затратами выполнить необходимое для данного случая тестирование.

Вот теперь давайте рассмотрим, как делается отладка проектов в FPGA на аппаратно-программном уровне.

Логический анализатор

Кратко можно напомнить историю. К первым однокристальным микроконтроллерам делались специальные кристаллы для отладочных устройств. Эти кристаллы содержали такие же микроконтроллеры, как и те, которые надо отлаживать, только все их регистры были на выводах микросхемы и эти выводы были подключены к схеме отладочного устройства. В то время как процессор выполняет команды и пересылает данные, внешнее по отношению к процессору устройство производит запись кодов адреса и данных, а также состояние служебных регистров и флагов микропроцессора в свой вспомогательный блок памяти. Далее эти данные пересылаются пользователю. Такое решение позволяло получать все состояния шин и регистров микропроцессора за определенный интервал времени.

Такие кристаллы устанавливались в специализированные устройства — внутрисхемные эмуляторы. Эти устройства позволяли получить полную трассировку отлаживаемой программы и значительно облегчали работу. На рис. 1 показана структура внутрисхемного эмулятора IECube. А на рис. 2 показано сопряжение эмулятора с отлаживаемой платой при помощи устройства сопряжения и соответствующего кабеля. Данный эмулятор фирма NEC поставляет для отладки своих микроконтроллеров. Гибкий кабель и набор переходных колодок позволяет надежно подключить устройство сопряжения с отладочным устройством к тестируемой плате. При этом дополнительно поставляемые переходники и кабели сопряжения позволяют подключать эмулятор к различного типа контактным площадкам микроконтроллеров на отлаживаемой плате. И если для корпусов типа DIP внутрисхемные эмуляторы привычно вставляются в колодки, то к контактным площадкам под BGA корпуса припаивается специальная переходная плата, а уже с нее сигналы подаются на эмулятор.

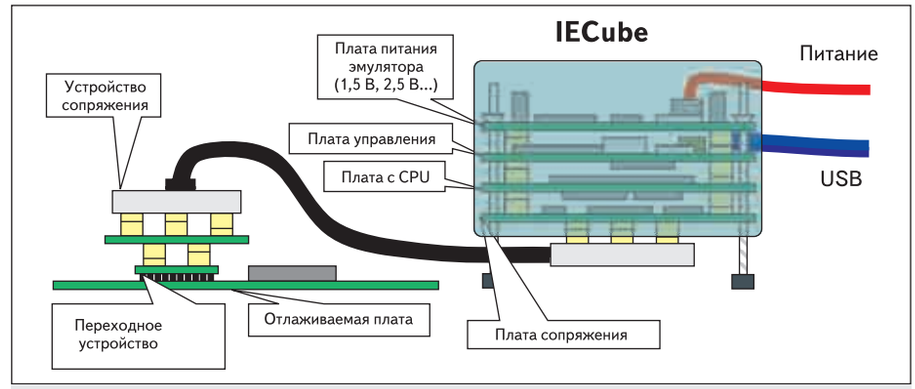

Рис. 1. Структура внутрисхемного эмулятора IECube

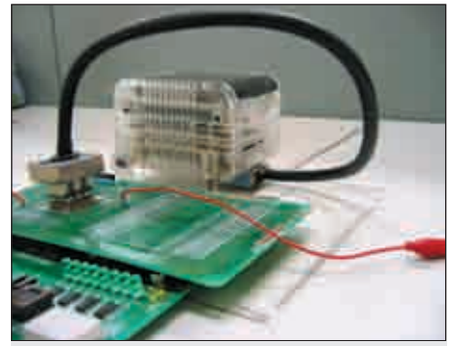

Рис. 2. Внешний вид внутрисхемного эмулятора IECube

Обычно наиболее распространенные типы внутрисхемных эмуляторов полностью поддерживались стандартными программными инструментами и не требовали изучения какого-либо дополнительного программного инструментария. Для разработчика оставалось только указать тип применяемого внутрисхемного эмулятора, и далее программа предоставляла разработчику полный сервис при отладке и трассировке. Но, к сожалению, ничто не вечно… Прорыв технологии изготовления полупроводников привел к появлению микропроцессоров, работающих на тактовых частотах свыше 30 МГц. Сегодня обычными тактовыми частотами становятся частоты в сотни мегагерц. А при таких частотах применение внешнего устройства становится невозможным.

Но вернемся к микроконтроллерам в FPGA. Точно такое же решение, а именно внутрисхемный эмулятор, можно применить и для отладки встроенного в FPGA микроконтроллера. При применении этого способа отладки у нас не будет таких ограничений по тактовой частоте, как в случае применения внешнего внутрисхемного эмулятора. Но, тем не менее, проект должен выполняться так, чтобы не возникало нарушений при обработке сигналов микроконтроллера.

Проект выполняется таким образом, что все регистры и память микроконтроллера делаются доступными по чтению и записи от host-машины. Примером данного подхода является подключение к проекту многоканального логического анализатора. Конечно, логический анализатор не может менять содержимое регистров, но он очень просто выполняется и может занимать достаточно мало ресурсов (объем занимаемых ресурсов определяется потребностями пользователя), поэтому он очень полезен при отладке в режиме реального времени. Итак, давайте рассмотрим пример выполнения логического анализатора.

Встроенный логический анализатор (ЛА)

Преимуществом данного способа отладки является то, что пользователь получает полную картину событий, причем запись ведется в реальном времени и не требует изменения в ПО пользователя и/или аппаратных ресурсов самого микроконтроллера. При таком способе работы получается запись максимально возможного числа параметров в реальном времени.

Но у данного способа есть и свои недостатки. К ним можно отнести несколько повышенный, по сравнению с первоначальным проектом, «расход ресурсов». Для записи информации необходимо иметь достаточный объем памяти. Для того чтобы информация о состоянии регистров и флагов попала в память, необходимо задействовать дополнительные ресурсы для прохождения (роутинга) сигналов внутри микросхемы. Чтобы преодолеть это противоречие, можно для отладки проекта использовать специальные платы, на которых устанавливают FPGA большего объема. Часто на такие платы устанавливают и дополнительные микросхемы памяти. При использовании внешней памяти объем данных, используемых при отладке, значительно увеличивается. Но при этом необходимо использовать корпуса микросхем FPGA с большим числом выводов. При использовании внешней памяти необходимо учитывать и ограничения по быстродействию.

Блок ЛА должен запоминать входные данные в буферную память, и затем эти данные могут быть считаны host-машиной. Для этого в FPGA имеются встроенные блоки памяти,

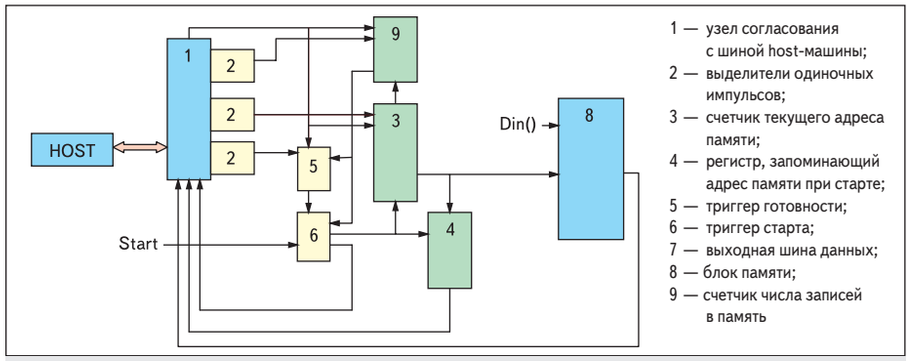

Рис. 3. Блок-схема простейшего логического стабилизатора

которые позволяют создавать массивы памяти требуемой разрядности. Далее системы большой сложности обычно имеют какую-либо шину, связывающую FPGA с внешним миром. Тип интерфейса и способ передачи информации существенной роли не играют.

Наиболее важным здесь является то, что есть возможность выполнить проект ЛА параметризируемым и, установив параметр USED = NO, не занимать ресурс в том блоке (файле), куда помещен ЛА. Встроим ЛА в каждый отлаживаемый нижний блок. Остается только снабдить каждый блок входами и выходами для ЛА и подвести все сигналы через все блоки к host-машине. Теперь, установив в нужном блоке ЛА параметр USED = YES, мы получим средство для контроля сигналов в любом нижнем блоке, и для этого не нужно исправлять весь проект. Мало того, таких ЛА можно задействовать в проекте несколько и считывать с них данные поочередно. Такое решение позволит отладить каждый блок в проекте на реальном «железе», отладить часть проекта или весь проект. У микросхем FPGA, большей частью, имеется возможность устанавливать на ту же самую посадочную площадку микросхему «большей емкости», поэтому на этапе отладки есть возможность встроить ЛА, а после отладки в серийных изделиях использовать микросхемы «меньшей емкости». Кроме того, устройства с микросхемами FPGA сами по себе могут служить и внешними ЛА.

Рассмотрим блок-схему простейшего логического стабилизатора (рис. 3).

ЛА работает следующим образом. До начала работы ЛА производится установка счетчика 3, то есть необходимо указать, сколько раз будет производиться запись в память после сигнала старта.

При обращении от host-машины сбрасывается триггер готовности, и ЛА переходит в состояние ожидания сигнала старта. При этом входная информация начинает записываться в память ЛА, что позволяет увидеть состояние сигналов на входах ЛА до момента старта. При приходе сигнала старта в регистр 4 записывается текущее состояние счетчика адреса памяти 3 и взводится триггер старта 6. Триггер старта 6 разрешает работу счетчика числа записей 9. По выполнении заданного числа записей триггер старта 6 сбрасывается, триггер готовности 5 взводится, и ЛА переходит в исходное состояние.

Далее host-машина читает счетчик адреса при старте 4, добавляет или вычитает смещение, задаваемое пользователем, и записывает полученное значение в счетчик текущего адреса памяти 3. При чтении данных host-машиной производится автоинкремент счетчика 3. Таким образом, производится чтение только части блока памяти или всего блока памяти.

Такое построение ЛА позволяет строить диаграммы с требуемым смещением относительно сигнала старта, то есть сигнал старта может быть в начале диаграммы, как у аналоговых осциллографов, в середине или в конце диаграммы — как у цифровых ЛА.

Поскольку данный ЛА находится «целиком в руках пользователя», то это позволяет варьировать условия запуска ЛА по его желанию от любого сигнала в блоке пользователя.

Тактовая частота записи данных также может быть выбрана пользователем. Это может быть как тактовая частота, используемая в конкретном блоке пользователя, так и внешняя частота. Мало того, тактовая частота может переключаться в зависимости от состояний сигналов в блоке пользователя. Можно выбрать другой режим работы управления синхрочастотой — при помощи сигнала разрешения. Управляя этим сигналом, можно блокировать запись данных, что позволяет существенно сократить требуемый для анализатора объем памяти.

ЛА, построенные таким образом, дают возможность строить диаграммы сигналов в любом требуемом пользователем формате, при любой требуемой развертке и любом, задаваемом параметрически, числе отсчетов данных. То есть один и тот же блок внутренней памяти можно сначала использовать как ЛА, имеющий 16 лучей, при 128 отсчетах и получить «панораму» событий, далее переключиться на 8 лучей при 265 отсчетах и так далее — до 1 луча при 2048 отсчетах.

Приведем фрагмент файла-рапорта, полученного при компиляции ЛА, имеющего 8 лучей при 256 отсчетах (рис. 4).

```
** DEVICE SUMMARY **

Chip/ Input  Output  Bidir  Memory  Memory   LCs
POF  Device  Pins  Pins  Pins  Bits %  Utilized  LCs %
Utilized

oscill EPF10K30ETC144-1  25  21  0  2048  8 %  105  6 %

User Pins:               25  21  0
```

Рис. 4. Часть рапорта, полученного при компиляции ЛА, имеющего 8 лучей при 256 отсчетах

Всего 105 ячеек, большая часть из которых занята привязкой асинхронного интерфейса host-машины к внутреннему синхронному проекту на системной тактовой частоте 33 МГц.

Таким образом, при очень скромных затраченных ресурсах можно получить сервис настоящего цифрового осциллографа.

Если пользователь имеет возможность затратить несколько больше ресурсов, то появляется возможность реализовать уже не простейший ЛА, а более функциональный, имеющий гораздо больше возможностей как по быстродействию, так и по возможности отображения записанной информации.

При необходимости проверить большее число сигналов на вход ЛА может быть подключен коммутатор сигналов, так можно расширить возможности работы ЛА внутри отлаживаемого блока.

Программа управления логическим анализатором представляет собой удобный интерфейс взаимодействия с ЛА и расширяет его возможности и область применения. Основное ее назначение — в отображении и сохранении данных, полученных из ЛА. Программа позволяет просматривать значение сигнала в любой момент времени, масштабировать графики, а также вводить на графики дополнительную информацию, такую как маркеры и подписи.

Кроме того, существует возможность объединения групп каналов в шину. Возможно различное трактование отображения сигналов шин. Так, например, для шины I²C можно отображать адресную часть кадра или данные кадра в виде цифровых значений, при этом сигналы могут быть представлены в прямом или инверсном виде.

Программная оболочка может представлять собой и простейший визуальный компонент из имеющихся библиотек, либо быть специализированной программой. Поскольку данный ЛА с точки зрения интерфейса также находится «целиком в руках пользователя», то и логика работы с шиной host-машины задается пользователем. Именно это и определяет легкость встраивания программной поддержки ЛА в тестовые, а при необходимости и в рабочие программы пользователя. Это позволяет контролировать работу ЛА непосредственно из программ пользователя, что значительно облегчает работу и требует меньших затрат ресурсов host-машины.

На рис. 5 показано основное окно программной оболочки для встроенного ЛА,

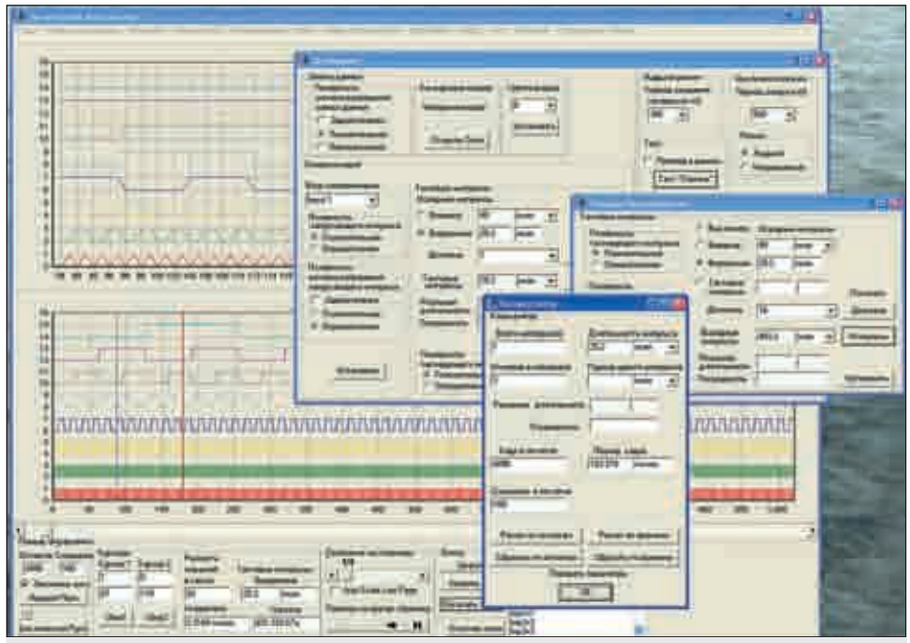

Рис. 5. Окно программной оболочки для встроенного ЛА (поверх него показано несколько окон для выбора настроек)

и поверх него показано несколько окон для выбора настроек.

Программа имеет несколько вспомогательных окон, необходимых для настройки интерфейса, режимов снятия данных и режимов отображения данных. Основное окно программы разделено на три части. В нижней части окна находятся элементы управления режимом работы. В средней части расположены основные диаграммы сигналов. Они могут быть разбиты на несколько страниц, и пользователь получает возможность просматривать диаграммы сигналов постранично. Также есть возможность установить и перемещать по диаграмме два курсора. В верхней части окна расположена «линза». Здесь изображены состояния сигналов, находящихся между курсорами на основной диаграмме.

Пример разработки аналогичного анализатора приведен на сайте фирмы Digilent Inc. Там же можно получить открытый проект, выполненный студентами университета Technical University of Cluj-Napoca [7].

Оценим «за» и «против» в этом случае. Сравним фирменные логические анализаторы и «самодельные». Сначала рассмотрим фирменные ЛА. Плюсы: поставляемые вместе с софтом фирм-производителей микросхем такие ЛА всегда и гарантированно работают правильно, по крайней мере, именно это обещают фирмы-производители. А минусы — это то, что работают они только через свои «родные» аппаратные адаптеры и никак иначе. Теперь рассмотрим вторую группу ЛА. Самодельные, и особенно взятые из открытых проектов, — они тоже работают. Но — без гарантии того, что «написано, как сказано» (фраза из «Оптимистической трагедии» Вс. Вишневского), поэтому прежде чем использовать такой логический анализатор, необходимо потратить время на его проверку. Но, с другой стороны, «самодельный» ЛА работает с тем аппаратным адаптером, который есть в распоряжении пользователя. Его можно запускать из любого программного обеспечения и это — его несомненное достоинство.

Внешний логический анализатор

Интересное совместное решение данного вопроса предложено компаниями Xilinx и Agilent. Это динамический пробник N5397A, N5406A для осциллографов смешанных сигналов серии Infiniium 8000 и 6000.

Ядро ATC2 (Agilent Trace Core — ядро трассировки Agilent) добавляется к проекту пользователя на этапе выполнения проекта. С помощью программы Core Inserter компании Xilinx можно задать нужные параметры ATC2 и создать необходимое для отладки ядро, которое лучше всего будет соответствовать потребностям разработки пользователя. Параметры включают число внешних выводов, число банков сигналов и вид измерения (анализ логических состояний или анализ временных диаграмм). Логические каналы осциллографа способны выполнять сбор данных только в режиме временных диаграмм. Анализ логических состояний производится осциллографом Infiniium 8000 в режиме постобработки данных, при этом возможно произвести объединение сигналов в шины.

Рис. 6. Осциллограф Infiniium 8000 с отлаживаемой платой

- На каждом внешнем выводе ПЛИС, выделенном для отладки, можно измерить до 64 внутренних сигналов. У осциллографов серии 8000 или 6000 имеется возможность с помощью их 16 логических каналов, подключенных к выводам FPGA, выделенных для отладки, получить доступ к 1024 внутренним сигналам FPGA.
- Перемещение контрольных точек внутри ПЛИС требует затрат времени. Теперь менее чем за секунду можно выполнить измерения на различных наборах внутренних сигналов без изменения схемы. При этом временные соотношения внутри ПЛИС остаются неизменными даже при выборе для исследования нового набора внутренних сигналов.
- Автоматическое включение цифровых каналов и шин в соответствии с выбранным банком сигналов. Названия сигналов, используемые в системе проектирования ПЛИС, автоматически отображаются в виде меток логических каналов в осциллографе смешанных сигналов.
- Поддерживаемые семейства ПЛИС компании Xilinx: Virtex-4, Virtex-II Pro, Virtex-II и Spartan-3.

На рис. 6 показано подключение отлаживаемой платы к осциллографу.

Какие же «за» и «против» в этом случае? По поводу «за» все и так понятно, ибо хороший инструмент — половина успеха. Что касается «против» — жаль, что не каждый разработчик может позволить себе иметь такой инструмент.

Трассирование микроконтроллера

Другой путь, требующий заметно меньше ресурсов для отладки, — это трассирование. Но он требует дополнительных ресурсов для ядра микроконтроллера. И для того чтобы выполнять само трассирование, необходимо к ядру процессора добавить блок, который будет выполнять функции трассирования.

Как же устроено такое отладочное устройство? Оно может выполнять несколько функций. Простейшей будет функция временного останова отлаживаемого микропроцессора, так как это делалось раньше в микропроцессорах по сигналу «Готовность». Функция останова обычно запускается либо при выполнении какой-либо процедуры процессора, либо при совпадении каких-либо условий, либо на каждой инструкции, выполняемой процессором. Например, при отладке есть необходимость проверить сигналы, выдаваемые в порт при каком-то условии.

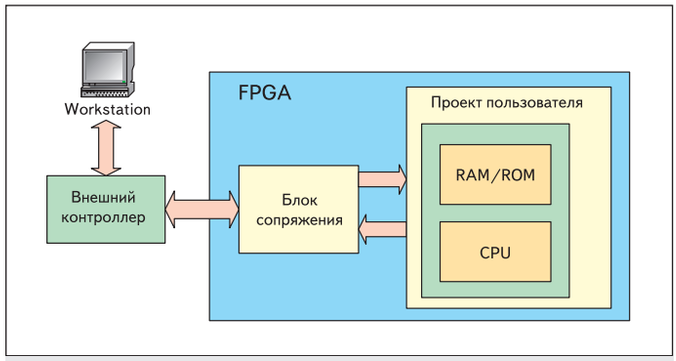

Рис. 7. Отладка проекта пользователя при помощи дополнительного внешнего микроконтроллера и блока сопряжения

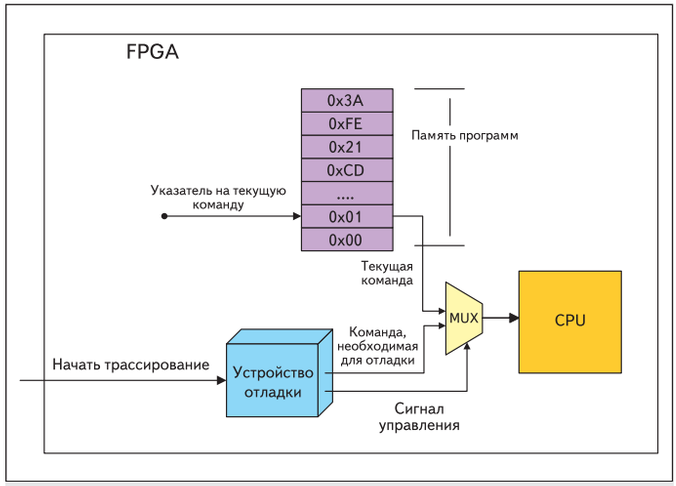

Рис. 8. Отладочное устройство, работающее в режиме трассирования

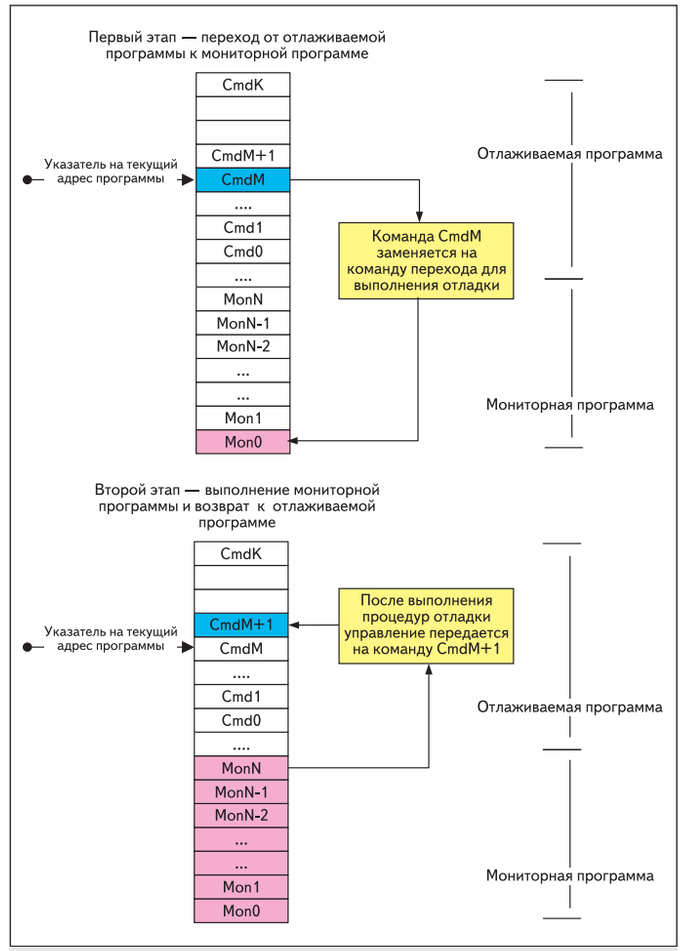

Рис. 9. Порядок выполнения команд при трассировании

Однако здесь необходимо учесть то, что при изменении режима работы процессора во времени необходимо изменить и режимы работы других узлов схемы. Представим, что в проекте имеется системный таймер. Этот таймер отсчитывает метки времени и, при достижении заданного значения, выдает свой запрос на блок прерывания процессора. Если мы не изменим режим работы такого таймера, то при выполнении процедур отладки таймер будет находиться в режиме счета и не даст процессору выполнить запланированное число команд между своими прерываниями. В данном случае есть два пути: первый из них — запретить прерывания этого таймера. Но при этом у нас может нарушиться режим работы программных ветвей, привязанных к прерываниям таймера. Второй путь — это синхронизировать работу такого таймера с выполнением команд основного поля памяти процессора. То есть отладочное устройство должно блокировать узлы, привязанные к реальному времени, на тот промежуток времени, пока процессор находится в ожидании или выполняет подпрограмму отладки. Такой способ работы хорошо подходит для тех узлов, которые привязаны к системной синхрочастоте. Однако для узлов, работающих с периферией, такой способ работы может и не подойти. Если отлаживаемый процессор должен очень быстро реагировать на поступающее к нему входное воздействие, то в режиме отладки такое воздействие может быть пропущено. То же самое может произойти, если входные данные должны приходить не кадрами, а потоком. Например, поток E1 или звуковые, либо видеоданные. При останове процессора для отладки часть входных данных может быть потеряна. Поэтому достаточно трудно выбрать способ отладки, чтобы он одновременно удовлетворял всем, подчас противоречивым потребностям.

Итак, возвращаемся к тому, как «заставить» процессор выполнять трассирование. Здесь можно поступить двумя способами.

Первый способ — привлечь для отладки нечто внешнее. Например, внутри кристалла можно организовать блок сопряжения, а снаружи кристалла задействовать дешевый микроконтроллер (рис. 7).

При срабатывании триггера в отладочном устройстве данные из FPGA передаются во внешний микроконтроллер, например по последовательной шине, связывающей отлаживаемый кристалл с микроконтроллером. Достоинством такого способа реализации будет то, что протокол передачи данных от внешнего микроконтроллера до хоста достаточно просто может быть реализован на стандартных аппаратных и программных ресурсах микроконтроллера. Причем такой узел может быть «проектонезависимым» и легко перестраиваться от проекта к проекту. После окончания процедуры передачи данных в хост отладочное устройство должно получить сигнал об окончании цикла останова.

Дополнительным преимуществом такой структуры является то, что внешний по отношению к кристаллу микроконтроллер может кроме отладки выполнять и многие другие функции. Например, функции загрузки и сохранения различных кодов ключей, паролей и т. д. Аналоговая часть микроконтроллера способна проверять аналоговые параметры FPGA — напряжение питания, температуру и другие.

Второй способ — использовать для отладки внутренние ресурсы. Чем занят отлаживаемый микроконтроллер в предыдущем примере? Да он просто бездельничает! Итак, заставим его работать!

Для этого отладочное устройство должно поменять порядок выполнения инструкций микропроцессора. При такой организации отладочное устройство выдает свою инструк-

цию вместо той, что должна была прийти в потоке команд, читаемых из памяти команд. При этом счетчик команд блокируется. Примером такой трассировки может служить известная команда TRAP. Команда, выдаваемая отладочным устройством, может вызвать какую-либо подпрограмму, находящуюся в основной памяти команд микропроцессора (рис. 8).

На рис. 9 показано, что замена кода команды из отлаживаемой программы на код команды перехода на мониторную программу выполняется аппаратно. Но ведь эту же процедуру можно делать и программно. Сначала суть дела. Необходимо в памяти программ микропроцессора заменить код команды на код перехода, выполнить процедуры трассирования, вернуться из процедур трассирования и вернуть в памяти команд код той команды, которая была заменена на команду перехода. Для такого способа работы необходимо иметь возможность где-то временно хранить код той самой команды, которую потом нужно «вернуть» назад. Необходимо анализировать коды команд, так как после выполнения операций по отладке программа должна будет «знать» то место, куда она должна вернуться. В том случае, если трассирование выполнялось на той команде, которая не может изменить порядок выполнения команд, то при возврате отлаживаемая программа вернется на следующую команду. Здесь все просто и понятно. А вот в случае команд ветвления, особенно команд с условным ветвлением, необходимо анализировать признаки ветвления. Например, команда перехода по биту четности «заставит» нас проверить значение бита четности на момент исполнения программы. Но и здесь дело довольно предсказуемое. А вот переход по значению аппаратного флага может оказаться роковым, если программа начнет анализировать текущее значение флага, а в это время само значение флага изменится. Конечно, здесь подразумевается, что флаг привязан к синхрочастоте процессора и все его изменения происходят вполне корректно под эту синхрочастоту. Речь в данном случае идет о том, что команда анализируется в одно время, а текущее значение флага выводится на консоль в другое время. Этот промежуток времени в несколько команд процессора может оказаться вполне достаточным для того, чтобы значение флага изменилось. В таком случае пользователь на консоли увидит, что процессор выполнил команду, например ветвление по флагу, равному 1, а само значение флага будет показано равным нулю. Или, например, команда возврата по флагу будет обработана при одном значении флага, а сам переход будет выполнен по другому значению. Чтобы у нас не происходило таких недоразумений, необходимо значения переменных считывать только один раз и хранить эти значения в дополнительных регистрах или в ячейках памяти.

И последнее замечание при режиме программной замены кодов команд. Чтобы заменить код команды, ее необходимо записать в поле памяти команд. А как мы знаем, не все процессоры это умеют делать. Конкретная реализация здесь может быть самая разная. Например, можно сделать порт косвенного адреса и косвенных данных. В регистр или в порт ввода/вывода записываются параметры, а при обращении к регистру данных по окончании этого обращения необходимо сформировать аппаратный цикл записи в память программ.

Несомненным достоинством второго способа трассирования является то, что для отладки не нужно иметь ничего внешнего. «Пусть процессор отлаживает себя сам!» — вот девиз этого способа. А недостатки? Недостатки очевидны. Для того чтобы этот способ заработал, необходимо в память программ, которую мы собираемся отлаживать, добавить кусок кода, в котором были бы описаны все процедуры, необходимые для того, чтобы получить информацию, обработать ее и затем передать по каналу связи. После чего отлаживаемый процессор должен произвести процедуру восстановления контекста в регистрах, памяти и т. д., то есть во всех тех узлах, которые использовались для отладки.

Итак, процессор должен устойчиво работать и достоверно передавать информацию, нужна дополнительная память и дополнительный код. С памятью проще всего — для отладки можно взять FPGA с большими ресурсами. Код можно написать и отладить в софт-симуляторе. Но, кроме этого, возможно, есть и еще одна трудность. Когда в этой статье был введен термин «отлаживаемый микроконтроллер», то структура самого микроконтроллера не была определена. А если обратиться к предыдущим статьям этого цикла, то читатель может заметить, что автор неоднократно повторял тезис о том, что не бывает «просто микропроцессора», так же как и не бывает «просто автомобиля». Для каждой конкретной задачи будет оптимален только свой набор команд и только своя архитектура. Вывод: возможен случай, когда микроконтроллер будет оптимален для целевой задачи, но он не сможет делать обработку и пересылку данных. Например, просто потому, что у него не будет соответствующих команд, не будет UART'ов и таймеров для генерации тактовой частоты для UART'а.

Подведем итог этого раздела. В данном случае режим реального времени выполняется только до тех пор, пока не начинаются процедуры трассировки.

И еще один вариант второго способа

То, что было описано выше, достаточно хорошо известно. Программы под названием «Монитор» начали применяться достаточно давно, и они вполне успешно применяются и сегодня. Но ведь мы ведем речь о микроконтроллерах в FPGA, а это значит, что у нас для отладки гораздо больше возможностей, чем у пользователей обычных «однокристалок». Для того чтобы пользоваться мониторными программами, пользователь однокристального микроконтроллера вынужден размещать мониторную программу в основном поле памяти программ. А это значит, что разрядности всех счетчиков адресов и указателей на память должно хватать для работы со всем полем памяти. Значит, команды прямой загрузки указателей также должны иметь поля соответствующей разрядности. И менять их тоже не хочется. Так как же делать отладку, чтобы монитор не был в основном поле памяти программ? В случае проектирования в FPGA — все в наших руках. Делаем второе поле памяти и помещаем мониторную программу туда. Теперь команда трассировки должна не только делать переход по адресу нужной директивы монитора, но и автоматически менять поля памяти. Соответственно, и команда возврата из прерывания, например, должна переключать поля памяти на основное поле памяти. При работе пользовательской программы в основном поле памяти такая команда будет только подтверждать, что выбрано основное поле памяти, а при отладке позволит выполнять переключение полей памяти автоматически.

В чем преимущество данного метода? В теневой памяти можно расположить мониторную программу, отладить ее один раз и далее, при отладке пользовательской программы, не менять содержимое данного поля памяти. Недостатки? Необходим мультиплексор данных, который бы мог коммутировать коды команд, получаемые от основного и теневого полей памяти (рис. 10). Возможно, что для работы с теневым полем памяти может понадобиться не только теневое ПЗУ, но и теневое ОЗУ. Их объем и расположение в поле адресов будет определяться структурой микроконтроллера и его набором команд.

Еще один вариант, представляющий собой гибрид двух предыдущих

Представим себе случай отладки устройства, не требующего режима реального времени. Например, микроконтроллер производит обслуживание АЦП. Связь с АЦП осуществляется последовательным кодом, и у нас нет жестких требований по времени выполнения программы. Основным условием в таком случае будет такое: программа в отлаживаемом микроконтроллере должна выполняться быстрее, чем работает АЦП. И если мы будем выполнять команды в десятки раз медленнее основного рабочего режима, но при этом удовлетворим условие по обслуживанию АЦП, то сможем значительно упростить отладочное устройство.

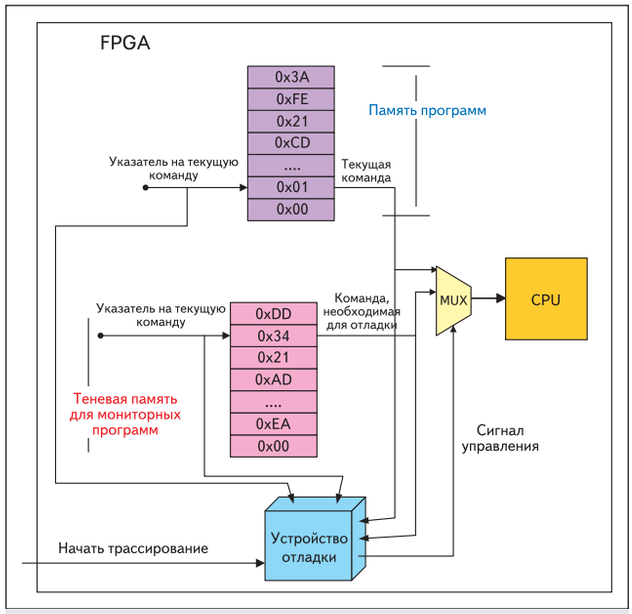

Рис. 10. Отладочное устройство, в котором имеется мультиплексор данных

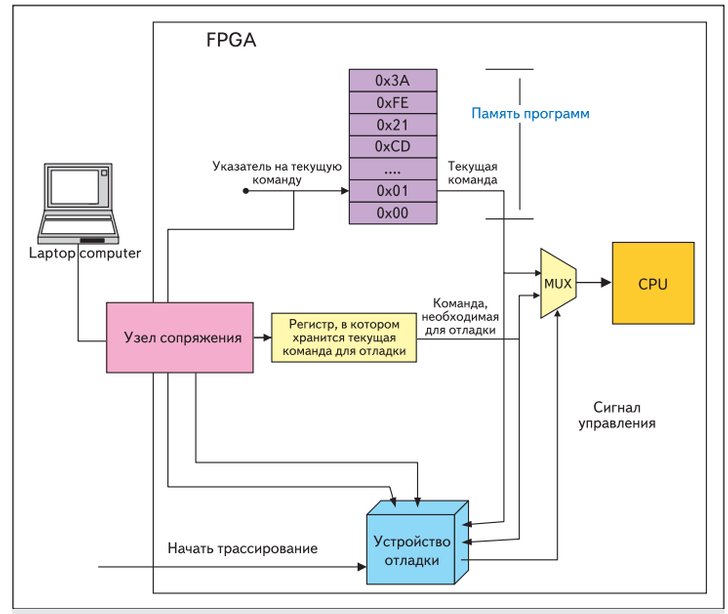

Рис. 11. Режим отладки

Для такого режима можно выполнять команды, выдаваемые отладочным устройством, одну за другой (рис. 11). Представим структуру, состоящую из отлаживаемого микропроцессора, отладочного устройства и внешнего контроллера или host-машины. Отлаживаемый микроконтроллер готов выполнить очередной шаг команды. (Возможны случаи, когда команда выполняется за несколько машинных циклов, например многобайтные команды.) Но отладочное устройство выдает сигнал останова и блокирует выполнение данного цикла. Процессор «замирает» в состоянии ожидания. Далее отладочное устройство производит переключение основного поля памяти на теневое поле памяти. В качестве такового в данном варианте нам будет достаточно только одного регистра для памяти команд и, возможно, еще одного для памяти данных и стека. Далее отладочное устройство заносит в регистр код команды. Если данная команда должна считывать данные из оперативной памяти или из стека, то в дополнительные регистры заносятся соответствующие данные. После завершения записи информации в регистры отладочное устройство «отпускает» отлаживаемый микроконтроллер на один такт. Отлаживаемый микроконтроллер выполняет только одну эту команду и снова останавливается. Устройство отладки вновь получает доступ к регистрам отлаживаемого микроконтроллера и анализирует его состояние.

Рассмотрим в качестве примера такой случай — микропроцессор остановился на команде записи в память данных. Отладочное устройство считывает код команды и анализирует его. По результатам анализа отладочное устройство считывает код адреса, по которому должны записываться данные. Этот код адреса может содержаться либо в самом коде команды при непосредственной записи литерала, либо в регистре указателя, при косвенной адресации. В более сложных процессорах код адреса может формироваться и более сложным образом. Например, при косвенной адресации со сдвигом относительно одного из регистров. При этом отладочное устройство может формировать дополнительные служебные сигналы так, чтобы отдельные узлы формирования адреса завершили бы свою работу по вычислению адреса, и на одной из шин микропроцессора появились бы требуемые данные. Далее отладочное устройство на основании полученных данных и данных о состоянии микропроцессора в host-машине или во внешнем микроконтроллере формирует картину состояния отлаживаемого микроконтроллера.

Структура состоит из отлаживаемого микропроцессора, отладочного устройства и внешнего контроллера или host-машины.

По результатам анализа состояния микропроцессора отладочное устройство принимает решение о том, какие данные и куда надо заносить, продолжать ли режим трассировки или произвести переключение на основное поле памяти. Основное поле памяти отлаживаемого микропроцессора может быть восстановлено в соответствии с той картиной памяти, которая находится в host-машине.

Интерфейс для связи отладочного устройства и внешнего процессора может быть любым. Наиболее предпочтительным в данном случае являются те интерфейсы, которые уже имеются в отлаживаемом изделии. Например, PCI или JTAG.

Рассмотрим преимущества. Такой вариант исполнения отладочного устройства является самым простым и требует минимума ресурсов. Недостатки — невозможен режим реального времени и требуется большая программная поддержка.

Первые итоги

Рассмотрены способы организации отладки микроконтроллеров. Приведены блок-схемы отладочных устройств. Описаны достоинства и недостатки каждой из рассмотренных схем. Конечно, приведены только самые распространенные способы выполнения. И на этом рассмотрение вариантов реализации отладочных устройств мы закончим.

В следующей части статьи перейдем к рассмотрению работы с портом JTAG.

Литература

1. Семенов Н., Каршенбойм И. Микропрограммные автоматы на базе специализированных ИС // Chip News. 2000. № 7.
2. Каршенбойм И. Микропроцессор своими руками // Компоненты и технологии. 2002. № 6, 7.
3. Каршенбойм И. Микропроцессор своими руками-2. Битовый процессор // Компоненты и технологии. 2003. № 7, 8.
4. Каршенбойм И. Микропроцессор своими руками-3. Ассемблер и софт-симулятор // Компоненты и технологии. 2006. № 3, 5.
5. Каршенбойм И.Виртуальные кнопки и светодиоды, или Неизвестное обо всем известном JTAG-сканировании // Компоненты и технологии. 2005. № 6.
6. Каршенбойм И., Паленов К. «Встроенный» логический анализатор — инструмент разработчика «встроенных» систем // Схемотехника. 2001. № 12.
7. **w ww .digilentinc.c om/Resources/DesignContest.cfm?Nav1=Design&Nav2=DesignContest**
8. **ww w .amontec.c om/** JTAG Interface: Common Pinouts amt_ann003 (v1.1) Application Note.
9. **w ww .heanet.dl.sourceforge.n et/sourceforge/jblaster/jblaster-1.1L.tar.gz**