УДК 519.237.5
DOI: 10.17587/prin.13.415-424

В. А. Галатенко, д-р физ.-мат. наук, зав. сектором автоматизации программирования, galat@niisi.ras.ru, **К. А. Костюхин,** канд. физ.-мат. наук, ст. науч. сотр., kost@niisi.ras.ru, Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук, Москва

Аппаратная отладка: обзор современных подходов*

Повсеместное распространение сложных устройств, построенных на так называемых системах на кристалле (System on Chip, SoC), с несколькими процессорными ядрами ставит новые задачи перед разработчиками встраиваемых систем. Новые средства разработки, специально предназначенные для сложных систем на кристалле, могут помочь их решить. Однако эти средства, как правило, ограничены функциональностью инструментов поддержки отладки. Высококачественная поддержка отладки с расширенными функциями необходима для использования всех преимуществ сложных SoC-устройств при одновременном сокращении времени разработки. В статье рассмотрены разные механизмы и способы реализации поддержки отладки систем на кристалле, предназначенных для сложных систем реального времени, используемых, например, в парадигме интернета вещей. Этот обзор включает оценку доступных решений и их пригодности для использования со следующим поколением сложных систем на кристалле с несколькими процессорными ядрами. Показано, что многие существующие решения не позволяют разработчикам легко воспользоваться преимуществами сложных функций, интегрированных в SoC следующего поколения. Обобщены и обсуждены основные функции поддержки отладки для многоядерных SoC. Даны рекомендации для разработчиков SoC и для будущего направления исследований в этой области в целях обеспечения более подходящей основы для новых инструментальных средств разработки. Такие средства крайне необходимы для всех встраиваемых систем жесткого реального времени и имеют первостепенное значение для минимизации сложности их разработки

Ключевые слова: *системы на кристалле, аппаратная отладка, SoC, цепочки сканирования, JTAG, трассировка*

Введение

От современных систем и устройств, реализующих парадигму интернета вещей (*Internet of Things,* IoT), все чаще требуется достаточно высокая производительность в режиме реального времени при сохранении низкого энергопотребления. Эти требования приводят к созданию многоядерных SoC-решений с поддержкой широкого спектра периферийных устройств и коммуникационных протоколов. Системы дополнительно ограничены требованиями к работе в суровых условиях, например, в моторном отсеке автомобиля или на радиолокационной вышке. Разработка встраиваемых систем жесткого реального времени, в которых неуспевшая завершиться в срок задача может привести к физическому повреждению устройства, является сложным процессом.

Следует отметить, что основными решениями разработки SoC (в том числе и отладки в процессе разработки) являются моделирование и верификация. Для этого используют средства, подобные FireSim [1], способные выполнять детальное моделирование на неограниченных по производительности облачных ресурсах с привлечением для адекватности моделирования аппаратных ресурсов. Тем не менее создать систему, полностью свободную от ошибок, невозможно.

В настоящей работе речь пойдет о средствах и методах обнаружения ошибок, проявляющихся непосредственно в ходе эксплуатации систем. Среди них выделяют традиционные инструментальные средства, такие как интерактивные отладчики и профилировщики. Они крайне необходимы для разработки надежных встраиваемых систем. Современные технологии позволяют ин-

* Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме "Исследование и реализация программной платформы для перспективных многоядерных процессоров" (FNEF-2022-002).

тегрировать всю систему на одном кремниевом чипе, известном как система на кристалле, что приводит к перемещению существующих внешних интерфейсов, широко используемых в целях разработки, на чип. Традиционно связь внутри встраиваемой системы осуществляется с использованием внешней системной шины процессора, которая реализована в виде дорожек на печатной плате. Печатная плата должна поддерживать соответствующие интерфейсы инструментов разработки, требующих физического подключения, например, логических анализаторов и осциллографов. Размещение ранее внешних интерфейсов на кристалле оставляет все меньше вариантов для внешних инструментов анализа и, фактически, делает разработчиков "слепыми" к внутреннему состоянию SoC. Без надежного и последовательного представления состояния встраиваемой системы обнаружение дефектов или ошибок в ней может стать очень трудной или даже не имеющей решения задачей. Основным решением задачи отсутствия интерфейсов является предоставление доступа к внутренним узлам системы извне через существующие интерфейсы. Существуют различные подходы к достижению требуемого при этом уровня наглядности, которые в настоящей работе изложены в общих чертах.

Поддержка процесса отладки является жизненно важной частью этапа разработки встраиваемой системы. Причина в том, что независимо от того, насколько тщательно проводится работа на этапах проектирования или тестирования, возникают ошибки, которые можно обнаружить только когда система работает. Источники таких ошибок включают неправильно определенные (сформулированные) или неправильно понятые спецификации, изменения оборудования на стадии производства и наиболее распространенный источник — человеческий фактор.

По оценкам Национального Института Стандартов и Технологий (NIST), ошибки и сбои обходятся только экономике США примерно в 59,9 млрд долл. США в год [2]. Улучшение качества готовой системы за счет более тщательного тестирования и верификации может снизить эти затраты на одну треть. Отмечается, что 77 % электронных сбоев в автомобилях были связаны с программным обеспечением [3]. Есть все основания предполагать, что число проблемных вопросов отладки встраиваемых систем и систем на кристалле будет только возрастать по мере развития технологий.

Как правило, в основе SoC-устройств лежат системы жесткого реального времени. Такие системы являются одними из самых сложных в реализации в условиях дефицита аппаратных и программных ресурсов. Современные подходы к проектированию систем жесткого реального времени основаны на детерминированном времени выполнения задач. Даже небольшое изменение времени может отрицательно сказаться на механизмах, которыми эти системы управляют. Например, если контроллер двигателя пропустит крайний срок запуска или запустит импульс слишком рано, двигатель может быть выведен из строя. Недетерминированность в системах жесткого реального времени настолько проблематична, что многие системные интеграторы даже не используют кеш-память, повышающую производительность, и нередко отказываются от преимуществ использования операционных систем реального времени, предпочитая реализовывать задачи непосредственно для выполнения на аппаратуре [4].

Существующие стратегии аппаратной отладки

Среди основных требований к эффективной поддержке процесса отладки аппаратуры можно выделить следующие:

- поддержка отладки не должна существенно изменять поведение устройства;
- наличие необходимой инфраструктуры для внешнего наблюдения за внутренним состоянием системы и другими критическими узлами;
- обеспечение внешнего доступа для управления состоянием системы и ресурсами, включая сложные периферийные устройства;
- средства поддержки аппаратной отладки не должны слишком усложнять SoC-устройство с точки зрения дополнительных контактов или площади чипа.

Повторное использование дизайна имеет решающее значение при разработке систем на кристалле для обеспечения своевременного выхода на рынок, поэтому поддержка отладки должна быть легко адаптируемой к различным системным архитектурам.

Существуют также различные методы отладки, которые обычно поддерживаются инфраструктурой отладки. Независимо от выбранной стратегии поддержки отладки базовая архитектура обычно содержит ряд основных частей. В следующих разделах дан обзор этих частей.

Одним из методов является посмертная отладка, которая останавливает SoC в ответ на такое событие, как, например, доступ по определенному адресу. В большинстве систем ошибкам требуется

время, чтобы проявить себя, они могут оставлять "улики", когда система остановлена, что упрощает поиск причины. К сожалению, некоторая часть этих улик исчезает к моменту остановки SoC, что затрудняет поиск ошибок и требует больших затрат на разработку. Посмертная отладка — это традиционный способ отладки встраиваемых систем с одним процессором [5], однако он имеет серьезные ограничения. Альтернативный подход заключается в наблюдении за системой во время ее работы.

Теперь рассмотрим четыре типа инфраструктуры отладки.

Программная поддержка отладки. Ресурсы поддержки отладки могут быть программными или аппаратными. Программная поддержка отладки реализуется с помощью процедур мониторинга [6, 7], в которых программное обеспечение компилируется или собирается с дополнительным программным кодом, осуществляющим отладочные действия при наступлении некоторого события, например, обращения по некоторому адресу или выполнению определенной команды процессора. Программное обеспечение может быть независимым от платформы и часто поддерживается в процессорах с помощью инструкций отладки или специальным режимом отладки [8]. Примером программного инструментария является использование так называемых print-процедур в программном коде для отображения или протоколирования некоторых переменных состояния системы. Помимо средств мониторинга следует упомянуть средства профилирования с поддержкой со стороны аппаратуры, например, интерфейс доступа к специальным регистрам-счетчикам процессора *Performance Application Programming Interface*, PAPI [9].

Поддержка процесса отладки на программном уровне почти всегда сопровождается средствами базовой аппаратной отладки [8], включая как традиционное стендовое оборудование, так и базовую поддержку отладки на кристалле. Стендовое оборудование включает в себя логические анализаторы и осциллографы. Базовая поддержка отладки на кристалле обеспечивает минимальный интерфейс для управления SoC. Поддержка отладочного программного обеспечения может быть активна в штатном режиме, однако при этом она может конфликтовать с системными задачами, существенно влияя на поведение системы. Следует учесть, что активное вмешательство в процесс работы SoC не всегда оправдано, особенно для систем реального времени. Остановка и последующее возобновление работы системы обычно изменяют взаимосвязь системы и окружения (периферийных устройств).

Навязчивый характер программных средств отладки делает их малопригодными для систем жесткого реального времени. Любые отладочные механизмы, используемые во время разработки, как правило, будут удалены или остановлены в конечном "боевом" варианте системы, изменив тем самым общее поведение [10]. Вторым недостатком этой стратегии является то обстоятельство, что, фактически, она поддерживает только "посмертную" отладку, которая предоставляет данные только после того, как ошибки проявляются, а не тогда, когда они действительно происходят. Тем не менее такое программное обеспечение очень полезно в некоторых приложениях и его гораздо безопаснее использовать в системах с мягкими сроками выполнения, например, в системах, не контролирующих критически важные аппаратные узлы, особенно, если система использует не все доступные ресурсы.

Отладка с использованием интерфейсов тестирования устройств. В настоящее время обычной практикой является использование технологии *design-for-test* (DFT) при разработке интегральных схем для поддержки производственного тестирования и отладки [6, 11]. Обычно она строится на механизме цепочек сканирования (*scan chains*) в соответствии со стандартом, обсуждаемым далее. В этот механизм включаются многие критически важные данные и способы управления SoC, что позволяет считывать состояние триггеров системы. В сочетании со средствами программной поддержки процесса отладки такой подход позволяет обеспечить простой инструментарий наблюдения и контроля. Основное преимущество решения на основе цепочек сканирования заключается в том, что оно использует непосредственно инфраструктуру DFT, поэтому не увеличивает производственные затраты.

Обычные цепочки сканирования используют один путь сканирования для уменьшения накладных расходов на маршрутизацию, но это ограничивает производительность. Еще одним недостатком является тот факт, что они обычно доступны только в режиме тестирования, поскольку цепочки сканирования мультиплексируются на контакты устройства, используемые для других целей в функциональном режиме. Использование выделенного тестового интерфейса позволяет преодолеть эту сложность. На практике производственные тесты преодолевают ограничения производительности за счет мультиплексирования нескольких цепочек сканирования на запасные

функциональные контакты [12]. Однако в функциональном режиме запасных контактов нет, поэтому такой подход не годится для поддержки отладки. Одним из решений этой задачи является введение дополнительного режима отладки, в котором цепочки сканирования объединяются вместе [13] или мультиплексируются [14]. Тогда они становятся доступны через выделенный интерфейс с одной линией сканирования.

Работающие цепочки сканирования приводят к перемещению данных, что потенциально изменяет выходные данные. Использование цепочек сканирования в неактивном состоянии позволяет избежать повреждения состояния системы, но делает невозможной ее отладку. В работе [15] приведены некоторые подходы, позволяющие проводить отладку SoC вместе с применением цепочек сканирования.

Устройства встроенной эмуляции. Встроенный эмулятор (*In Circuit Emulator*, ICE) — это специально изготовленный прототип, заменяющий обычную производственную SoC. Такой эмулятор помогает преодолеть ограничения отладки на основе тестовой инфраструктуры или базовой аппаратной и программной инструментальной отладки, предоставляя дополнительные возможности подключения отладочных инструментов. Множество дополнительных отладочных контактов, расположенных на расширенной площади платы, не используются системой в функциональном режиме, вместо этого они подключены к внутренним выводам для обеспечения возможности наблюдения за поведением системы с помощью, например, логического анализатора. Следует отметить, что отладочные данные в этом случае потенциально доступны во время нормальной работы системы, поэтому, в принципе, использование дополнительных контактов не влияет на поведение SoC.

Недостатком такого подхода становится увеличенный размер микросхемы и наличие дополнительных соединений внутри нее, зачастую с использованием длинных соединительных проводов. Это обстоятельство отрицательно сказывается на надежности микросхемы, что может привести к полной ее непригодности для эксплуатации в суровых условиях, таких как отсек двигателя или коробки передач. Кроме того, подключение дополнительных соединительных проводов нагружает внутреннюю коммуникацию, снижая производительность и изменяя поведение прототипа по сравнению с реальным устройством, внося таким образом, дополнительные потенциальные ошибки.

Поддержка отладки на основе эмуляции на кристалле. Более распространенной альтернативой встроенным ICE является эмуляция на кристалле (*on-chip emulation*), расширяющая возможности отладки обычной SoC за счет создания на самом кристалле дополнительных инструментальных средств сбора данных [16]. Такие средства обычно включают в себя логику управления точками останова, позволяющими останавливать систему при достижении определенного адреса в сегменте кода или доступе к определенной ячейке памяти. Аппаратные ресурсы отладки работают параллельно с обычным режимом функционирования SoC, поэтому поведение системы практически не меняется. Согласованное поведение системы делает стратегии отладки на основе самоэмуляции подходящими для большинства приложений, включая разработку сложных систем реального времени.

Технология эмуляции на кристалле позволяет использовать такой инструментарий, как трассировка, которая предоставляет разработчикам достаточную информацию об активности системы во время выполнения для последующего углубленного анализа.

Еще одним преимуществом трассировки является ее способность фиксировать события, приводящие к возникновению проблемы, а не только ее последствия. Поддержание правильного временного порядка событий также имеет важное значение для предоставления разработчику достоверной информации.

Трассировка на кристалле значительно отличается от встроенных ICE, которые выдают только необработанные сигналы. Встроенная трассировка допускает первичную обработку данных, например, их сжатие, необходимое для наблюдения за несколькими высокоскоростными встроенными процессорами и доступом к данным.

По мере развития интегральных схем развивалась и поддержка отладки. Каждая из четырех рассмотренных технологий имеет свои преимущества и недостатки. Современная тенденция отладки заключается в том, чтобы в значительной степени фокусироваться на отслеживании труднообнаруживаемых спорадических ошибок, поскольку их иногда невозможно обнаружить с помощью методов традиционной посмертной отладки. Профилирование и оптимизация систем реального времени также в большой степени зависят от наличия средств аппаратной трассировки.

Управление SoC с несколькими процессорными ядрами

Сложные многоядерные SoC-устройства с несколькими процессорами и периферийными устройствами ставят новые проблемные вопросы

разработчикам SoC. Современная SoC — это не просто интеграция нескольких схем на печатной плате, а новый подход к проектированию, требующий эволюции методов проектирования.

Интерфейсы управления запуском и доступом к памяти. Традиционным интерфейсом разработки SoC является так называемый интерфейс управления запуском (*run control*). Он используется для управления работой микроконтроллера или одноядерного процессора SoC. В частности, он позволяет разработчику запускать и останавливать такие устройства, как процессор.

Некоторые устройства SoC используют свой собственный проприетарный интерфейс [17], однако широко распространенным решением является использование существующего порта тестирования граничного сканирования (JTAG) [18]. Применение в целях отладки стандартизированного интерфейса тестирования не требует дополнительных выводов на плате, что дает этому подходу потенциал для внедрения на разных платформах. Системы с несколькими процессорами на печатной плате ранее включали несколько интерфейсов отладки, часто основанных на стандарте IEEE 1149.1, который первоначально предназначался для структурного тестирования цифровых печатных плат. Ранние SoC-устройства с несколькими процессорными ядрами были сконструированы путем интеграции большинства некогда отдельных компонентов на одной системной печатной плате. Многие из ядер имеют собственные встроенные контроллеры TAP (*Test Access Port*), что делает актуальным вопрос о том, как управлять множеством контроллеров TAP, сохраняя при этом соответствие стандарту IEEE 1149.1. В работе [15] предложены несколько вариантов решения этого вопроса, включая популярный вариант включения всех контроллеров TAP в одну цепочку сканирования.

Однако предложенный в работе [15] так называемый агент-ориентированный подход видится авторам более удачным решением. По сути, этот подход является следующим эволюционным шагом в развитии систем на кристалле, обеспечивая внедрение системно-ориентированной парадигмы, в которой каждая SoC имеет один контроллер TAP для связи с внешними инструментами отладки и тестирования [19]. Следует отметить, что устаревшие процессорные ядра не могут быть изменены, поэтому придется их поддерживать, пока они не будут выведены из эксплуатации. Контроллер TAP на уровне кристалла целиком, который взаимодействует с интерфейсами отладки каждого ядра, обеспечивает более масштабируемую архитектуру.

Существует реализация интерфейса, называемая JTAG+, сочетающая в себе аспекты как центральной, так и распределенной архитектур [16]. В ней используется один центральный контроллер TAP и цепочка сканирования для связи со специальным отладочным процессором (агентом отладки), который, в свою очередь, использует специальную системную шину для доступа к памяти и внутреннему состоянию остальных процессоров SoC. Это существенно отличается от таких решений, как EJTAG [20], где отладочный сопроцессор привязан к каждому процессорному ядру и доступен с помощью цепочек сканирования. Устаревшие ядра поддерживаются путем подключения цепочек сканирования к центральному контроллеру TAP, который обеспечивает выбор между встроенными контроллерами TAP и агентом отладки.

Контроль хода выполнения SoC-устройств с несколькими ядрами и таймерами. В большинстве современных SoC-устройств точки останова поддерживаются аппаратными триггерами. В системах с трассировкой триггеры также могут использоваться для запуска и остановки сбора трассы, а также фильтрации событий для сохранения пропускной способности и объема памяти. Этот процесс фильтрации известен как квалифицирование трассы, поскольку выбираются только наиболее полезные для последующего анализа события.

Большинство сложных SoC-устройств сейчас имеют несколько системных таймеров, что усложняет синхронизацию событий, произошедших на разных ядрах. Это обстоятельство также влияет на отладку, поскольку усложняет остановку работающей системы целиком. При использовании одного системного таймера вся SoC может быть остановлена одновременно, но с разными или несинхронизированными таймерами существует вероятность рассинхронизации данных [21]. Хотя рассинхронизация данных может быть обнаружена, вопрос с перезапуском SoC остается. По-прежнему одним из главных отладочных действий остается полный останов и последующий запуск всей системы на кристалле. Чтобы избежать изменения поведения системы, все ее компоненты должны быть остановлены в один и тот же момент времени. Одним из решений является построение распределенного коммутатора для всех отладочных шин [22]. Преимуществом такого решения является тот факт, что компоненты SoC подключаются к общему блоку, действующему как центральный узел для детерминированной маршрутизации сигналов прерывания с минимальной задержкой. Использование коммутатора позволяет разработчику настроить, какие ядра будут реагировать на сигналы прерывания.

Поддержка трассировки

Рассмотренные до сих пор функции поддержки процесса отладки в основном связаны с контролем и наблюдением за состоянием системы в отдельные моменты времени. Чтобы найти сложные ошибки, разработчикам необходимо просмотреть срезы состояния системы за некоторое время до проявления ошибки. Чтобы предоставить доступ к этой информации, SoC может получать и обрабатывать данные о своем состоянии в режиме реального времени. Затем полученная трасса становится доступной для внешних средств разработки либо через порт отладки/трассировки [7, 20, 22—25], либо путем сохранения во встроенной памяти, откуда ее можно считывать в автономном режиме с помощью интерфейса управления запуском [20, 24]. Трассировка — это не просто полезный инструментарий для отладки одноядерных систем. Он также крайне необходим для отладки взаимодействия между несколькими процессорными ядрами и активными периферийными устройствами.

Стандартные инструментальные средства сбора данных о трассе просто регистрируют поток управления процессорного ядра [25], протоколируя тип выполняемых инструкций в порядке их появления. Существуют также инструментальные механизмы регистрации событий доступа к данным [23], которые помогают найти ошибки, связанные с общими переменными, блокировками и другими взаимодействиями, где важен порядок событий. Отслеживание данных всех активных устройств также очень важно, поскольку не все взаимодействия связаны с процессором [10]. Базовая трасса программы может содержать только информацию, достаточную для восстановления потока управления в автономном режиме. Она может быть дополнена подробными сведениями о, например, запущенной задаче или возникшем прерывании. Трассы критически важных систем, применяемых, например, в космических аппаратах, могут включать гораздо больше деталей [26].

Типичная трасса потока данных содержит такую информацию, как начальный адрес данных, их размер и значения, а также флаг доступа — чтение или модификация. Периферийные устройства могут также предоставлять дополнительную информацию о состоянии, такую как пропускная способность интерфейса или статистика возникновения ошибок соединения.

Информация о состоянии, не требующая анализа в реальном времени, обычно считывается из регистров, отображенных в памяти, с помощью интерфейса управления запуском.

Сжатие трассы. Некоторые инструментальные средства сбора данных о трассе при протоколировании данных от потока управления генерируют простое в реализации сжатое сообщение фиксированной ширины для каждого процессорного такта [8, 25]. Альтернативная стратегия, определенная стандартом NEXUS [23] и используемая некоторыми производителями SoC [7], заключается в том, чтобы формировать сообщения различной длины и отправлять их только при необходимости. Уменьшение размера сообщений в стандарте NEXUS достигается за счет того, что зачастую программы, выполняемые на процессоре, обращаются в память на коротком диапазоне адресов (например, обработка массивов). В этом случае возможно хранить и передавать только часть адреса, а не весь адрес целиком (так называемое разностное сжатие, *differential compression*). Некоторые производители пользуются универсальными алгоритмами сжатия данных, такими как коды Хаффмана [27].

В стандарте NEXUS описывается алгоритм сжатия данных о трассе, в котором сообщения отправляются только для событий, требующих синхронизации, таких как инструкции ветвления или trap-инструкции. Вместо отдельных сообщений для последовательных инструкций подсчитывается число последовательных инструкций, выполненных с момента формирования предыдущего сообщения, и это число включается в следующее сообщение. Фактически речь идет о протоколировании не отдельных инструкций, а целых линейных участков кода. Такой подход позволяет значительно уменьшить размер трассы, но при условии, что у инструментального средства анализа данных о трассе имеется доступ к исходному коду, используя который можно воспроизвести поведение программы.

Наиболее эффективным методом уменьшения объема трассы является возможность управления трассировкой на уровне отдельных аппаратных модулей SoC, а также фильтрация событий по разным условиям. К таким относятся, например, события только чтения или только записи данных, события выполнения только инструкций сопроцессора плавающей арифметики и т. п. [28].

Управление трассировкой. Кроме способов генерации трассы также важны методы управления сбором трассы и ее хранения. Сбором данных о трассе можно управлять вручную через внешний интерфейс. Однако управление с помощью определенных аппаратно-программных триггеров более эффективно, поскольку позволяет разработчикам получить данные о системе в момент, непосредственно предшествующий триггерному событию ("до"-данные), или сразу за ним последу-

ющий ("после"-данные) [28]. Сбор "после"-данных полезен в ситуации, когда разработчик подозревает о наличии некорректного поведения системы, начиная с определенного события, в то время как сбор "до"-данных нужен, когда разработчик не уверен в причине ошибки и хочет увидеть изменение поведения системы вплоть до проявления ошибки.

Самый простой подход к трассировке системы с несколькими ядрами заключается в том, что средство управления трассировкой переключается между каждым ядром — источником трассы [22]. Однако такой подход не ориентирован на систему целиком и имеет ограниченное применение там, где ядра взаимодействуют между собой.

Настоящая системно-ориентированная комбинированная трассировка должна обеспечивать согласованное представление о взаимодействиях внутри системы. Алгоритм объединения сообщений от каждого источника является важным фактором в увеличении эффективности трассировки. При наличии разных системных таймеров необходимо уметь упорядочивать сообщения, используя логическое время. В этом случае разработчик получит достоверную картину того, что происходит внутри SoC в целом. Полученная такая образом трасса позволяет разработчику обнаруживать ошибки, связанные с параллелизмом, включая доступ к общим переменным. Одним из решений, способных объединить трассы из нескольких источников в правильном временном порядке, является сеть на кристалле (*Network-on-Chip*, NoC [29]).

В устройствах SoC с разнородными процессорными ядрами часто нет другого выбора, кроме как пытаться объединить данные от несколько отдельных систем трассировки, поскольку каждое семейство процессоров имеет собственные реализации отладочных средств и интерфейсов. Введение общего стандартизированного интерфейса отладки решило бы задачу системной интеграции.

Рекомендации по поддержке отладки

Для отладки сложных систем на кристалле разработчикам требуются инструментальные средства, учитывающие наличие нескольких процессорных ядер и активных периферийных устройств, которые являются ключевыми компонентами высокопроизводительных встраиваемых систем. Различные архитектуры ядер, наличие нескольких системных таймеров, требования к работе в реальном времени и повышенная сложность других компонентов систем на кристалле показали ограничения традиционных методов посмертной отладки. Применение инфраструктуры отладки и тестирования (JTAG, JTAG+, EJTAG) для отладки всей системы на кристалле не является жизнеспособным методом. В основном это связано с тем обстоятельством, что тестовая инфраструктура сосредоточена почти исключительно на деятельности, связанной с тестированием. Поэтому исследование и разработка новых методов, учитывающих все аспекты отладки, принесло бы пользу SoC. Отметим, что это решение должно заменить устоявшиеся методологии тестирования.

Механизм аппаратных точек останова, по сути, остается незаменимым средством отладки. Дополнение этого механизма функциональными возможностями трассировки (вместо создания отдельных аппаратных средств трассировки) поможет снизить накладные расходы на поддержание средств отладки.

Программные методы и средства отладки хорошо изучены и будут оставаться ценным инструментарием. Однако для повышения эффективности неразумно при проектировании SoC полагаться только на программное обеспечение.

Для систем реального времени трассировка является минимальным требованием и должна быть усилена наличием механизмов фильтрации и аппаратными триггерами. Многоядерным SoC требуется инфраструктура для объединения механизмов сбора трассы каждого компонента, с сохранением при этом истинного временного порядка их сообщений с идентификацией источника. Пока существуют только базовые решения, следовательно, необходимо разрабатывать и исследовать новые методологии для поддержки будущих поколений более сложных систем на кристалле.

Одной из таких методологий, по мнению авторов, может являться концепция контролируемого выполнения [30]. В этой концепции все инструментальные средства отладки тесно интегрированы. Кроме того, сама система на кристалле еще на этапе проектирования получает именно те средства отладки, которые направлены на локализацию и устранение типичных ошибок для той целевой области, под которую эта система разрабатывается. Разработчики аппаратных компонентов и программного обеспечения должны понимать, что им необходимо тесно сотрудничать с точки зрения определения задач и для достижения практических результатов.

При современном подходе к разработке SoC, ориентированном на IP-блоки, важно, чтобы поддержка процессов разработки и отладки стала самостоятельным блоком многократного использования. Такой блок должен быть отделен от платформ и процессоров, он должен обладать спецификой для конкретного поставщика, иметь стандартизированные интерфейсы, подобными тем, которые определены в работе [31].

Заключение

Системы на кристалле эволюционировали до очень сложных устройств, но при этом поддержка средств их разработки и отладки фактически осталась на прежнем уровне. Возрастающая сложность архитектуры с одной стороны и повышенные требования к высокому уровню качества и надежности с другой стороны сделали поддержку разработки решающим фактором для успешного создания новых современных систем на кристалле. С помощью "правильных" инструментальных средств можно решить вопросы соответствия новым повышенным требованиям современных сложных систем на кристалле. Для продвижения интеграции системы важно, чтобы поддержка разработки SoC перестала фокусироваться исключительно на недорогих и менее эффективных архитектурах. Будущее отладки, несомненно, в системно-ориентированным решении. Как показано в настоящей работе, здесь был достигнут определенный прогресс, но многие разработчики SoC не спешат внедрять новые методы и инструментальные средства отладки, в которых остро нуждается конечный пользователь.

В настоящей работе выделены области, в которых есть возможность для дальнейших исследований. Очевидно, что у каждой системы на кристалле есть свое собственное предназначение и следующие из этого требования. Соответственно, по мере развития технологий необходимо и совершенствование средств отладки таких систем.

Список литературы

1. **FireSim.** URL: https://fires.im
2. **US** Department of Commerce. The economic impacts of inadequate infrastructure for software testing, Technical Report, RTI-7007.011US, National Institute of Standards and Technology (US), 2002. 309 p. URL: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf
3. **Mayer A., McDonald-Maier K. D.** Debug support, calibration and emulation for multiple processor and powertrain control SoCs [automotive applications] // Design, Automation and Test in Europe. Vol. 3. Munich (DE). 7—11 March 2005. P. 148—152.
4. **Scottow R. G., McDonald-Maier K. D.** Measuring determinism in real-time embedded systems using cached processors//Proceedings of the 2005 International Conference on Embedded Systems and Applications, ESA'05. Las Vegas, 7—11 March 2005. P. 38—44.
5. **Huang I.-J., Lu T.-A.** ICEBERG: an embedded in-circuit emulator synthesizer for microcontrollers // Proceedings of the Design Automation Conference. 1999. P. 580—585. DOI: 10.1109/DAC.1999.94.
6. **Zorian Y., Jan Marinissen E., Dey S.** Testing embedded-core-based system chips // Computer. 1999. Vol. 32, No. 6. P. 52—60. DOI: 10.1109/2.769444.
7. **Motorola Inc.** MPC565/MPC566 user's manual, 1387 p. URL: https://seniord.ece.iastate.edu/projects/archive/may0715/resources/MPC565_MPC566RM.pdf
8. **Infineon** Technologies AG. Tricore 1 architecture manual, ver. 1.3.8, January 2008, 280 p. URL: https://www.infineon.com/dgdl/tc_v131_corearchitecture_v__138.pdf?fileId=db3a304412b407950112b409c4500359
9. **Performance** Application Programming Interface, PAPI. URL: https://cvw.cac.cornell.edu/Profiling/papi
10. **Mayer A., Siebert H., Kolof A., el Baradie S.** Debug support for complex system-on-chips // Proceedings of the Embedded Systems Conference, April 2003. P. 1—16.
11. **Golshan F.** Test and on-line debug capabilities of IEEE Std 1149.1 in UltraSPARCTM-III microprocessor // Proceedings of the International Test Conference, October 2000. P. 141—150. DOI: 10.1109/TEST.2000.894201.
12. **Vermeulen B., Goel S. K.** Design for debug: catching design errors in digital chips // IEEE Design & Test of Computers. 2002. Vol. 19, No. 3. P. 35—43.
13. **Van Rootselaar G. J., Vermeulen B.** Silicon debug: scan chains alone are not enough // Proceedings of the International Test Conference (IEEE Cat. No. 99CH37034). 1999. P. 892—902. DOI: 10.1109/TEST.1999.805821.
14. **Jung D.-J., Kwak S.-H., Lee M.-K.** Reusable embedded debugger for 32 bit RISC processor using the JTAG boundary scan architecture // Proceedings of the IEEE Asia-Pacific Conference on ASIC, 2002. P. 209—212. DOI: 10.1109/APASIC.2002.1031569.
15. **Hopkins A. B. T., McDonald-Maier K. D.** Debug support for complex systems on-chip: a review // Computers and Digital Techniques. 2006. No. 4. P. 197—207. DOI: 10.1049/ip-cdt:20050194.
16. **Maier K. D.** On-chip debug support for embedded systems-on-chip // Proceedings of the International Symposium on Circuits and Systems, Bangkok, Thailand. 25—28 May 2003. P. 565—568. DOI: 10.1109/ISCAS.2003.1206375.
17. **Melear C.** Using background modes for testing, debugging and emulation of microcontrollers // Proceedings of the WESCON/97 Conference, 1997. P. 90—97. DOI: 10.1109/WESCON.1997.632324.
18. **IEEE** JTAG 1149.1-2013 Std. IEEE standard test access port and boundary-scan architecture, IEEE Computer Society, 2013. 444 p.
19. **Oakland S. F.** Position statement: TAPs all over my chips // Proceedings of the International Test Conference. 7—10 October 2002. P. 1192—1193. DOI: 10.1109/TEST.2002.1041899.
20. **MIPS** Technologies. EJTAG trace control block specification, MD00148 Rev. 1.04, 2002. 66 p. URL: http://www.t-es-t.hu/download/mips/md00148a.pdf
21. **Goel S. K., Vermeulen B.** Hierarchical data invalidation analysis for scan-based debug on multiple-clock system chips // Proceedings of the International Test Conference (ITC02). Baltimore. USA, 7—10 October 2002. P. 1103—1110. DOI: 10.1109/TEST.2002.1041867.
22. **Infineon** Technologies AG. Infineon TC1920 system units user's guide v1.3, October 2003. 78 p. URL: https://www.alldatasheet.com/datasheet-pdf/pdf/80124/INFINEON/TC1920.html
23. **IEEE-ISTO** 5001-2012 Std. Standard for a global embedded processor debug interface, Version 3.0. The Nexus 5001Forum, 2012. 190 p. URL: https://nexus5001.org/wp-content/uploads/2018/05/IEEE-ISTO-5001-2012-v3.0.1-Nexus-Standard.pdf
24. **Zeinolabedin S., Partzsch J., Mayr C.** Real-time Hardware Implementation of ARM CoreSight Trace Decoder // IEEE Design & Test. 2021. Vol. 38, No. 1. P. 69—77. DOI: 10.1109/MDAT.2020.3002145.
25. **Renesas** Technology Corp. Hitachi SuperH RISC engine SH7144 Series hardware manual, Rev. 2.0, 2002. 773 p. URL: https://datasheet.octopart.com/HD64F7145F50V-Renesas-datasheet-154546106.pdf
26. **Gaisler Research.** GRLib, 2008. 77 p. URL: https://opencores.org/websvn/filedetails?repname=mips_enhanced&path=%2Fmips_enhanced%2Ftrunk%2Fgrlib-gpl-1.0.19-b3188%2Fdoc%2Fgrlib.pdf
27. **Huffman D. A.** A method for the construction of minimum redundancy codes // Proceedings of the IRE. 1952. Vol. 40, No. 9. P. 1098—1101. DOI: 10.1109/JRPROC.1952.273898.
28. **ARM.** Embedded trace macrocell architecture specification, 2011. URL: https://developer.arm.com/documentation/ihi0014/q/
29. **Benini L., De Micheli G.** Networks on chips: a new SoC paradigm // Computer. 2002. Vol. 35, No. 1. P. 70—78. DOI: 10.1109/2.976921.
30. **Бетелин В. Б., Галатенко В. А., Костюхин К. А.** Контролируемое выполнение приложений на многопроцессорных платформах // Информационные технологии. 2015. Том 21, № 10. С. 723—728.
31. **Hopkins A. B. T., McDonald-Maier K. D.** Debug support strategy for systems-on-chips with multiple processor cores // IEEE Transactions on Computers. 2006. Vol. 55, No. 2. P. 174—184. DOI: 10.1109/TC.2006.22.

Hardware Debugging: an Overview of Modern Approaches

V. A. Galatenko, galat@niisi.ras.ru, **K. A. Kostyukhin**, kost@niisi.ras.ru,
Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences", Moscow, 117218, Russian Federation

Corresponding author:

Konstantin A. Kostyukhin, Senior Researcher, Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences", Moscow, 117218, Russian Federation
E-maill: kost@niisi.ras.ru

Received on July 12, 2022
Accepted on July 27, 2022

The ubiquity of complex devices built on systems on chip (SoC) with multiple processor cores poses new challenges for developers of embedded systems. New development tools specifically designed for complex systems on chip can help solve them, but these tools are usually limited by the functionality of debugging support tools. High-quality debugging support with advanced features is necessary to take full advantage of complex SoC devices while reducing development time. The article discusses various mechanisms and ways to implement support for debugging systems on chip designed for complex real-time systems used, for example, in the Internet of Things (IoT) paradigm. This review includes an assessment of the available solutions and their suitability for use with the next generation of complex systems on chip with multiple processor cores. It is shown that many existing solutions do not allow developers to easily take advantage of complex functions integrated into the next-generation SoC. The basic debugging support functions for multicore SoCs are summarized and discussed. Recommendations are given for SoC developers and for the future direction of research in this area in order to provide a more suitable basis for new development tools. Such tools are extremely necessary for all embedded hard real-time systems and are of high importance for minimizing the complexity of their development.

Modern systems and devices implementing the Internet of Things paradigm are increasingly required to have sufficiently high real-time performance while maintaining low power consumption. These requirements lead to the creation of multicore SoC solutions with support for a wide range of peripheral devices and communication protocols. The systems are limited by the requirements for working in harsh conditions, such as, for example, in the engine compartment of a car or on a radar tower. The development of embedded hard real-time systems, in which a task that fails to be completed on time can lead to physical damage of the device, is a complex process.

Effective tools, such as interactive debuggers and profilers, are an integral part of solving these problems and are vital for developing reliable embedded systems. Modern technologies now allow the integration of the entire system on a single silicon chip, known as a system on a chip (SoC), which leads to the relocation of existing external interfaces, widely used for development purposes, to the chip. Traditionally, communication within an embedded system is carried out using an external processor system bus, which is implemented in the form of tracks on a printed circuit board. The printed circuit board must support the appropriate interfaces of development tools that require a physical connection, such as, for example, logic analyzers and oscilloscopes. Previously, placing external interfaces on a chip left fewer options for external analysis tools and, in fact, makes developers "blind" to the internal state of the SoC. Without a reliable and consistent representation of the state of the embedded system, the detection of defects or errors in the system may become difficult or even impossible. The main solution to the problem of the lack of external interfaces is to provide access to internal nodes from outside the system through existing interfaces. There are many different approaches to achieving the required visibility, and they are outlined in this review.

Keywords: *system on a chip, hardware debugging, SoC, scan chains, JTAG, tracing*

For citation:

Galatenko V. A., Kostyukhin K. A. Hardware Debugging: an Overview of Modern Approaches, *Programmnaya Ingeneria*, 2022, vol. 13, no. 9, pp. 415—424.

DOI: 10.17587/prin.13.415-424

References

1. **FireSim,** available at: https://fires.im
2. **US** Department of Commerce. The economic impacts of inadequate infrastructure for software testing, Technical Report, RTI-7007.011US, National Institute of Standards and Technology (US), 2002, 309 p, available at: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf
3. **Mayer A., McDonald-Maier K. D.** Debug support, calibration and emulation for multiple processor and powertrain control SoCs [automotive applications], *Design, Automation and Test in Europe*, vol. 3, Munich (DE), 7—11 March 2005, pp. 148—152.
4. **Scottow R. G., McDonald-Maier K. D.** Measuring determinism in real-time embedded systems using cached processors, *Proceedings of the 2005 International Conference on Embedded Systems and Applications, ESA'05*, Las Vegas, 7—11 March 2005, pp. 38—44.
5. **Huang I.-J., Lu T.-A.** ICEBERG: an embedded in-circuit emulator synthesizer for microcontrollers, *Proceedings of the Design Automation Conference*, 1999, pp. 580—585, DOI: 10.1109/DAC.1999.94.
6. **Zorian Y., Jan Marinissen E., Dey S.** Testing embedded-core-based system chips, *Computer*, 1999, vol. 32, no. 6, pp. 52—60. DOI: 10.1109/2.769444.
7. **Motorola Inc.** MPC565/MPC566 user's manual, 1387 p., available at: https://seniord.ece.iastate.edu/projects/archive/may0715/resources/MPC565_MPC566RM.pdf
8. **Infineon** Technologies AG. Tricore 1 architecture manual, ver. 1.3.8, January 2008, 280 p., available at: https://www.infineon.com/dgdl/tc_v131_corearchitecture_v__138.pdf?fileId=db3a304412b407950112b409c4500359
9. **Performance** Application Programming Interface, PAPI, available at: https://cvw.cac.cornell.edu/Profiling/papi
10. **Mayer A., Siebert H., Kolof A., el Baradie S.** Debug support for complex system-on-chips — Proceedings of the Embedded Systems Conference, April 2003, pp. 1—16.
11. **Golshan F.** Test and on-line debug capabilities of IEEE Std 1149.1 in UltraSPARCTM-III microprocessor, *Proceedings of the International Test Conference,* October 2000, pp. 141—150. DOI: 10.1109/TEST.2000.894201.
12. **Vermeulen B., Goel S. K.** Design for debug: catching design errors in digital chips, *IEEE Design & Test of Computers*, 2002, vol. 19, no. 3, pp. 35—43.
13. **Van Rootselaar G. J., Vermeulen B.** Silicon debug: scan chains alone are not enough, *Proceedings of the International Test Conference (IEEE Cat. No.99CH37034),* 1999, pp. 892—902. DOI: 10.1109/TEST.1999.805821.
14. **Jung D.-J., Kwak S.-H., Lee M.-K.** Reusable embedded debugger for 32 bit RISC processor using the JTAG boundary scan architecture, *Proceedings of the IEEE Asia-Pacific Conference on ASIC*, 2002, pp. 209—212. DOI: 10.1109/APASIC.2002.1031569.
15. **Hopkins A. B. T., McDonald-Maier K. D.** Debug support for complex systems on-chip: a review, *Computers and Digital Techniques*, 2006, no. 4, pp. 197—207. DOI: 10.1049/ip-cdt:20050194.
16. **Maier K. D.** On-chip debug support for embedded systems-on-chip, *Proceedings of the International Symposium on Circuits and Systems,* Bangkok, Thailand, 25—28 May 2003, pp. 565—568. DOI: 10.1109/ISCAS.2003.1206375.
17. **Melear C.** Using background modes for testing, debugging and emulation of microcontrollers, *Proceedings of the WESCON/97 Conference*, 1997, pp. 90—97. DOI: 10.1109/WESCON.1997.632324.
18. **IEEE** JTAG 1149.1-2013 Std**.** IEEE standard test access port and boundary-scan architecture, IEEE Computer Society, 2013, 444 p.
19. **Oakland S. F.** Position statement: TAPs all over my chips, *Proceedings of the International Test Conference*. 7—10 October 2002. P. 1192—1193. DOI: 10.1109/TEST.2002.1041899.
20. **MIPS** Technologies. EJTAG trace control block specification, MD00148 Rev. 1.04, 2002, 66 p., available at: http://www.t-es-t.hu/download/mips/md00148a.pdf
21. **Goel S. K., Vermeulen B.** Hierarchical data invalidation analysis for scan-based debug on multiple-clock system chips, *Proceedings of the International Test Conference (ITC02),* Baltimore, USA, 7—10 October 2002, pp. 1103—1110. DOI: 10.1109/TEST.2002.1041867.
22. **Infineon** Technologies AG. Infineon TC1920 system units user's guide v1.3, October 2003, 78 p., available at: https://www.alldatasheet.com/datasheet-pdf/pdf/80124/INFINEON/TC1920.html
23. **IEEE-ISTO** 5001-2012 Std**.** Standard for a global embedded processor debug interface, Version 3.0, The Nexus 5001Forum, 2012, 190 p., available at: https://nexus5001.org/wp-content/uploads/2018/05/IEEE-ISTO-5001-2012-v3.0.1-Nexus-Standard.pdf
24. **Zeinolabedin S., Partzsch J., Mayr C.** Real-time Hardware Implementation of ARM CoreSight Trace Decoder, *IEEE Design & Test,* 2021, vol. 38, no. 1, pp. 69—77. DOI: 10.1109/MDAT.2020.3002145.
25. **Renesas** Technology Corp**.** Hitachi SuperH RISC engine SH7144 Series hardware manual, Rev. 2.0, 2002, 773 p., available at: https://datasheet.octopart.com/HD64F7145F50V-Renesas-datasheet-154546106.pdf
26. **Gaisler** Research. GRLib, 2008, 77 p., available at: https://opencores.org/websvn/filedetails?repname=mips_enhanced&path=%2Fmips_enhanced%2Ftrunk%2Fgrlib-gpl-1.0.19-b3188 %2Fdoc%2Fgrlib.pdf
27. **Huffman D. A.** A method for the construction of minimum redundancy codes, *Proceedings of the IRE*, 1952, vol. 40, no. 9, pp. 1098—1101. DOI: 10.1109/JRPROC.1952.273898.
28. **ARM.** Embedded trace macrocell architecture specification, 2011, available at: https://developer.arm.com/documentation/ihi0014/q/
29. **Benini L., De Micheli G.** Networks on chips: a new SoC paradigm, *Computer*, 2002, vol. 35, no. 1, pp. 70—78. DOI: 10.1109/2.976921.
30. **Betelin V. B., Galatenko V. A., Kostyukhin K. A.** Multiprocessor applications controlled execution), *Informacionnye tehnologii,* 2015, vol. 21, no. 10, pp. 723—728 (in Russian).
31. **Hopkins A. B. T., McDonald-Maier K. D.** Debug support strategy for systems-on-chips with multiple processor cores, *IEEE Transactions on Computers*, 2006, vol. 55, no. 2, pp. 174—184. DOI: 10.1109/TC.2006.22.