УДК 004.052.42 DOI: 10.22184/NanoRus.2019.12.89.65.69

# СРЕДСТВО УДАЛЕННОЙ ОТЛАДКИ ПРОЦЕССОРОВ «ЭЛЬБРУС»

# REMOTE DEBUGGING TOOL OF ELBRUS PROCESSORS

**БИЛЯЛЕТДИНОВ ИЛЬЯ ЕВГЕНЬЕВИЧ**[1]
***Инженер-программист***
*Ilia.E.Bilyaletdinov@mcst.ru*

**ОМЕТОВ АЛЕКСАНДР ЕВГЕНЬЕВИЧ**[2]
***Инженер***
*Alexander.E.Ometov@mcst.ru*

**ТИМИН ЛЕОНИД СЕРГЕЕВИЧ**[1,2]
***Начальник сектора***
*le0@mcst.ru*

**ВИНОГРАДОВ АРТЕМ АНДРЕЕВИЧ**[2]
***Инженер***
*Artem.A.Vinogradov@mcst.ru*

[1] *ПАО «Институт электронных управляющих машин им. И. С. Брука»*
*119334, Москва, ул. Вавилова, 24*
*Тел.: +7 (495) 363-95-03*
[2] *АО «МЦСТ»*
*117105, Москва, ул. Нагатинская, 1, стр. 23*
*Тел.: +7 (495) 363-95-03*

**BILYALETDINOV ILIA E.**[1]
***Engineer-programmer***
*Ilia.E.Bilyaletdinov@mcst.ru*

**OMETOV ALEXANDER E.**[2]
***Engineer***
*Alexander.E.Ometov@mcst.ru*

**TIMIN LEONID S.**[1,2]
***Head of sector***
*le0@mcst.ru*

**VINOGRADOV ARTEM A.**[2]
***Engineer***
*Artem.A.Vinogradov@mcst.ru*

[1] *Brook INEUM PJSC*
*24 Vavilov St., Moscow, 119334, Russia*
*Tel.: +7 (495) 363-95-03*
[2] *MCST JSC*
*bld. 23, 1 Nagatinskaya St., Moscow, 117105, Russia*
*Tel.: +7 (495) 363-95-03*

Разработана удаленно управляемая System Verilog-модель JTAG-контроллера, а также программа доступа к отладочным средствам процессора через JTAG с возможностью работы по модели клиент-сервер. Разработаны библиотеки для доступа к серверу на языках программирования TCL, Python, C, C++.
*Ключевые слова: YAML; ZMQ; JTAG; клиент-сервер.*
A remotely controlled System Verilog model of JTAG-controller has been developed, as well as a program to access the debugging tools of processor through JTAG with the ability to work on the client-server model. Libraries for access to the server in programming languages TCL, Python, C, and C++ have been developed.
*Keywords: YAML; ZMQ; JTAG; client-server.*

## ВВЕДЕНИЕ

Во время отладки процессоров требуется доступ к их диагностическим средствам, который можно осуществить через отладочный интерфейс. В процессорах «Эльбрус» в качестве отладочного интерфейса используется JTAG. Он позволяет получить доступ к системе управления частотой процессора, температурным датчикам, физическим уровням памяти и межпроцессорных связей, отладочным механизмам ядер и других частей процессора. В момент прихода новых процессоров нет гарантии, что все работает так, как задумывалось. Из-за этого можно потерять прямой программный доступ к каким-нибудь частям процессора. В таких случаях может помочь использование заранее заложенных в процессор отладочных средств, например, можно менять начальные настройки процессора в обход штатных средств с помощью JTAG. Даже если все работает, отладочный доступ позволяет проводить исследование границ работоспособности процессора, получать информацию и тестировать процессор с минимальным количеством оборудования (часто необходимо лишь подать питание) в экстремальных условиях. Однако работоспособность диагностических средств и программного обеспечения для управления ими необходимо проверять на этапе разработки процессора.

Диагностическое программное обеспечение, работающее через JTAG-интерфейс, взаимодействует с процессором путем подачи тестовых воздействий в виде битовых векторов и наблюдения отклика устройства [1]. Однако отлаживать данный процесс на уже готовом изделии проблематично, т. к. возможно наблюдать только конечный результат воздействий, что не всегда дает точную информацию об ошибке в векторе. В рамках данной работы для осуществления верификации диагностических средств процессора и тестов была разработана модель JTAG-контроллера, позволяющая взаимодействовать с Verilog-моделью целевого устройства через JTAG-интерфейс.

## СТАНДАРТ JTAG

JTAG-интерфейс предназначен для подключения сложных цифровых микросхем или устройств к стандартной аппаратуре тестирования и отладки [3]. Идея тестирования состоит в следующем: в микросхеме выделяются функциональные блоки, входы которых можно отсоединить от остальной схемы, подать заданные комбинации сигналов и оценить состояние выходов каждого блока. Весь процесс производится специальными

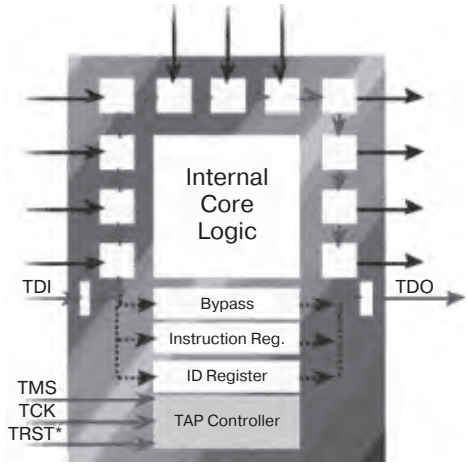


*Рис. 1. Схема работы JTAG*

командами по интерфейсу JTAG (рис. 1), при этом никакого физического вмешательства не требуется.

Сам порт тестирования TAP (Test Access Port) представляет собой четыре или пять выделенных выводов микросхемы:

1) TCK — тактирует работу встроенного автомата управления интерфейсом;
2) TMS — обеспечивает переход схемы в/из режима тестирования и переключение между разными режимами тестирования;
3) TDI — вход последовательных данных;
4) TDO — выход последовательных данных;
5) TRST — опциональный сигнала сброса/инициализации порта тестирования.

Интерфейс JTAG управляется TAP-контроллером, который является конечным автоматом с 16 состояниями. Переходы между ними осуществляются сигналами TMS и TCK.

В дополнение к TAP-контроллеру интерфейс внутри микросхемы состоит из нескольких сдвиговых регистров различной длины, которые называются Scan-цепочками (или DR — Data Registers). Они могут иметь различное назначение: от простого снятия значений с выходных буферов микросхемы до обеспечения доступа к внутреннему функционалу схемы, например к регистрам физического уровня памяти.

Один TAP-контроллер может управлять сразу несколькими такими цепочками. Это осуществляется с помощью инструкций, которые записываются в регистр инструкций (IR — Instruction Register) внутри самого TAP-контроллера. Каждой инструкции соответствует одна Scan-цепочка.

Чтобы выбрать необходимую инструкцию, необходимо с помощью сигналов TCK и TMS последовательно перевести TAP-контроллер через состояния SelectIR, CaptureIR и ShiftIR. После этого, когда мы перешли в последнее состояние, можно начать последовательно вдвигать код инструкции. Последним этапом будет перевод контроллера в одно из состояний UpdateIR.

Как только необходимая инструкция загружена, можно начинать работу с выбранной Scan-цепью. Последовательность действий схожа с вышеописанной: переводим контроллер через состояния SelectDR, CaptureDR и ShiftDR, совершаем необходимые манипуляции, проходим через одно из состояний UpdateDR.

## МОДЕЛЬ JTAG-КОНТРОЛЛЕРА

Для осуществления диагностики микропроцессора необходимо подключить к инструментальной машине JTAG-контроллер, соединенный с JTAG-портом платы с тестируемым процессором. Далее запустить тест и дождаться вывода диагностической информации.

Тесты представляют собой наборы битовых векторов, которые будут вдвинуты в Scan-цепочку. Составлением этих последовательностей занимается специальный САПР или сам программист, основываясь на специализированной документации. На этом этапе необходимо удостовериться, что битовые вектора составлены правильно.

Отладка на уже готовом изделии проблематична, т. к. возможно наблюдать только конечный результат воздействий, что не всегда дает точную информацию об ошибке в векторе. Наиболее простым и очевидным выходом из данной ситуации является использование Verilog-модели устройства, на которой можно получить детальную информацию о векторе и его воздействиях на систему. Однако для этого необходимо иметь модель и самого JTAG-контроллера, который будет преобразовывать битовые вектора в правильную последовательность сигналов JTAG-интерфейса. Поэтому в рамках данной работы для осуществления верификации диагностических средств процессора и тестов была разработана модель JTAG-контроллера, позволяющая взаимодействовать с Verilog-моделью целевого устройства через JTAG-интерфейс.

К любой модели JTAG-контроллера предъявляются следующие требования:

1. Слежение за состоянием TAP-контроллера. Модель должна самостоятельно находить правильные пути перехода по таблице состояний для выполнения поставленной перед ней задачей.
2. Обмен данными по интерфейсу JTAG. Модель должна быть ответственна за преобразование битовых векторов в сигналы и формирование отклика.

Язык Verilog — язык описания аппаратуры. Он специфичен и не подходит для прямого взаимодействия с высокоуровневыми языками программирования, на которых написано отладочное ПО. Поэтому при написания аппаратной части был использован System Verilog. Это комбинация HDL языка Verilog и языков аппаратной верификации. Его отличительной чертой является поддержка интерфейса DPI (Direct Programming Interface). Он связывает System Verilog и другой язык программирования, например C++, позволяя им вызывать функции друг из друга.

Таким образом, для получения максимальной выгоды с функциональной точки зрения контроллер разделен на две части: одна на языке C, программно-интерфейсная, а другая — на System Verilog, аппаратная. Используя такой подход, можно пользоваться обширным количеством различных библиотек языка C и C++, при этом не теряя возможностей языка Verilog.

Часть на языке System Verilog представляет из себя несинтезируемый модуль, который подключается к соответствующему интерфейсу модели процессора внутри тестового окружения. Она ответственна за формирование синхросигнала TCK, за подачу значений сигналов ко входам и за снятие значений с выходов JTAG-интерфейса.

Диагностическое ПО, использующее JTAG-интерфейс, может использовать различную частоту TCK — от 1 до 40 МГц. Поэтому необходимо было реализовать возможность изменения частоты «на лету» без перезапуска моделирования. Исходя из этого генерируемый синхросигнал TCK имеет два параметра C и D, которые доступны для изменения из интерфейсно-программной части и полностью описывают форму и частоту сигнала (рис. 2).

За подачу и снятие значений с JTAG-интерфейса процессора отвечают task'и (аналоги функций из высокоуровневых языков

программирования). Они осуществляют определенные переходы по состояниям TAP-контроллера, преобразуют значения управляющих сигналов, пришедших из части на языке C, в соответствующие сигналы и снимают значения с выходов.

В отличие от реального устройства, модель имеет такие состояния сигналов, как X и Z. В System Verilog этим состояниям присвоены значения 2 и 3, в дополнение к 1 и 0 для высокого и низкого уровней. Это вносит некоторую неопределенность при формировании отклика. Поэтому для максимального приближения к реальной работе устройства при детектировании состояния X или Z модель контроллера возвращает 0.

За формирование значений управляющих сигналов для JTAG-интерфейса ответственна часть на языке C. Эти значения она получает из битовых векторов от диагностического ПО. Вектор преобразуется в последовательность сигналов TDI и TMS, которые затем передаются в аппаратную часть и применяются ко входам JTAG-интерфейса процессора. Одновременно с этим аппаратная часть возвращает значения с выхода TDO, что позволяет интерфейсно-программной части формировать отклик.

Еще одной задачей, которую выполняет эта часть, является слежение за состояниями TAP-контроллера. Модель следит за текущим состоянием и при необходимости вызывает необходимые task'и из аппаратной части, чтобы перейти в необходимое для выполнения операции состояние.

Главная задача части на языке C — обеспечение программного интерфейса модели JTAG-контроллера. После анализа программного интерфейса реальных контроллеров фирм Corelis и JTAG Technologies были выбраны основные функции, используемые в отладочном ПО, и разработан универсальный программный интерфейс модели. Среди этих функций scan_dr (вдвигает вектор в цепочку DR), scan_ir (вдвигает вектор в цепочку IR), tms_reset (сброс состояния TAP-контроллера с помощью подачи высокого уровня сигнала TMS в течении пяти тактов TCK), и другие. Такой подход позволил без больших затрат перестроить ПО под режим отладки тестовых воздействий.

Моделирование работы процессора или даже его небольшой части является сложной и объемной вычислительной задачей, поэтому в большинстве случаев оно происходит на мощных серверных машинах, доступ к которым имеется по сети. Вследствие этого было решено реализовать возможность удаленного тестирования с использованием метода удаленного вызова процедур (RPC).

RPC — класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве (как правило, на удаленных компьютерах). Обычно реализация RPC-технологии включает в себя два компонента: сетевой протокол для обмена в режиме клиент-сервер и язык сериализации объектов. RPC очень удобен в случае реализации многих вычислительных программ, т. к. позволяет минимизировать нагрузку и требования к клиенту, оставляя основные вычисления серверной части.

В данной работе в качестве сетевого протокола была выбрана высокопроизводительная библиотека сообщений ZMQ (ZeroMQ). Она работает на различных архитектурах от ARM до Itanium и поддерживается более чем в 20 языках программирования. Главным преимуществом данной библиотеки является то, что она не требует выделенного сервера, что значительно упрощает работу с программой.

Основой ZMQ являются ZMQ-сокеты. Они очень похожи на TCP-сокеты, но главное отличие в том, что каждый сокет может поддерживать связь с множеством участников сети.

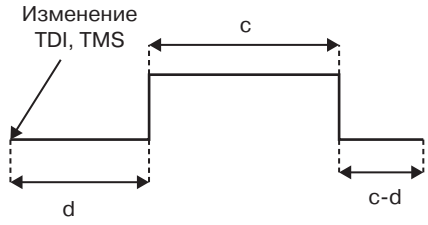


*Рис. 2. Сигнал TCK*

В ZMQ предусмотрены следующие типы обмена:

1. REQUEST/REPLY — двусторонняя связь между программами-абонентами распределенной MQ-системы: одна программа-клиент может взаимодействовать с одной или несколькими программами-серверами. Каждое отправленное сообщение предусматривает уведомление о доставке.
2. PUBLISH/SUBSCRIBE — опубликовать сообщение для множества подписчиков. От предыдущего метода отличается тем, что программа-отправитель не получает уведомлений о получении сообщений программами-подписчиками. В данном методе, однако, тоже имеется механизм регулирования: определяется некоторое пороговое количество сообщений, которое может оставаться в очереди неполученным подписчиком, и при попытке опубликовать очередное сообщение система сбрасывает попытку с соответствующим уведомлением отправителя.
3. UPSTREAM/DOWNSTREAM или Pipeline — этот метод используется для иерархической рассылки сообщений. DOWNSTREAM используется для рассылки вниз по иерархии, а UPSTREAM — наоборот. Программа-отправитель не получает уведомлений о доставке и также, как и в случае с предыдущим методом, имеется ограничение на количество неполученных сообщений.
4. PAIR — взаимодействие только между клиентом и сервером. Данный тип взаимодействия не предполагает маршрутизации сообщений и не содержит уведомлений о доставке.

Исходя из специфики задачи был выбран режим взаимодействия REQUEST/REPLY. Т. е. на стороне клиента создается клиент с типом сокета REQUEST, а на серверной части — с типом REPLY.

Среди преимуществ данной библиотеки еще стоит выделить простое API, наличие большинства базовых протоколов передачи сообщений и наличие активной поддержки со стороны разработчиков.

Языком сериализации для реализации RPC был выбран YAML. Это «дружественный» формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования. YAML поддерживает списки и ассоциативные массивы, а также любые иерархические комбинации этих двух элементов, что активно используется в данной работе.

Этот язык сериализации примечателен тем, что является OpenSourc-проектом, имеет простую и понятную документацию, поддерживается во многих языках программирования и обладает удобным способом сериализации и десериализации данных, а также полностью совместим с JSON.

В итоге после реализации RPC модель контроллера разделилась на серверную и клиентскую части со взаимодействием типа REQUEST/REPLY (рис. 3).

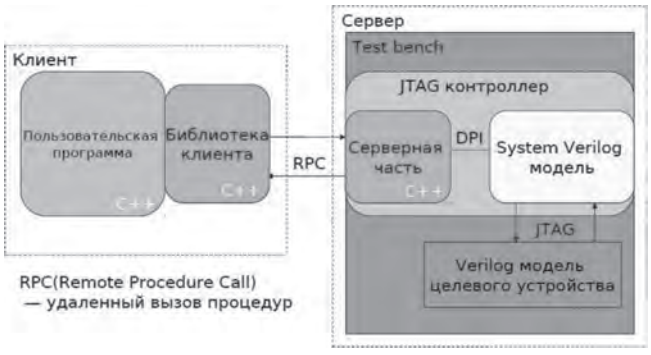


*Рис. 3. Схема работы модели JTAG-контроллера*

Клиентская часть представляет собой библиотеку функций, которые позволяют производить манипуляции с JTAG-цепочками внутри процессора с помощью битовых векторов, абстрагируясь от Verilog-моделирования. Библиотека ответственна за формирование и посылку запроса серверу, а также за получение ответа.

В серверную часть входит модуль на языке System Verilog, часть на C и новая часть на C++, ответственная за получение и обработку запросов клиента и формирование ответов.

В результате реализованная архитектура позволяет:

1) отладить тестовые вектора и диагностическое оборудование на Verilog-модели целевого устройства, абстрагируясь от самого процесса моделирования;
2) детально проследить за воздействием, которое оказывает тестовый вектор, используя возможности САПР, в котором происходит моделирование;
3) производить удаленную отладку на сервере по сети;
4) не прерывать процесс моделирования для изменения тестовых воздействий, т. к. это можно сделать «на лету» в отладочной программе и с помощью библиотеки снова послать на исполнение.

## СРЕДСТВО ОТЛАДКИ

После отладки на модели тесты можно запускать на самом процессоре. Однако простое тестирование через стандартные механизмы JTAG не является полным. Ввиду этого внутрь процессора разработчиками добавляются дополнительные отладочные средства, доступные по JTAG-цепочке с использованием специальных инструкций. Для обеспечения тестопригодности процессора желательно иметь возможность прямого доступа ко всем внутренним элементам через JTAG, однако это невозможно ввиду огромного размера такой инфраструктуры. Поэтому во многих современных процессорах, включая «Эльбрус», реализована возможность подавать инструкции в ядро процессора чрез отладочный порт.

В «Эльбрусе» для подачи инструкций в ядро существует специальный JTAG-регистр DCUR. Он записывает полученную инструкцию в регистр команд процессора, а также управляет отладочным остановом ядра. Для исполнения инструкции сначала необходимо остановить ядро, а затем записать инструкцию и отдать ее на исполнение специальным флагом. Данный механизм позволяет полностью управлять процессором, однако не убирает необходимость иметь прямой отладочный доступ к другим частям процессора. Ядру для функционирования требуется опорная частота, которой может не быть, например, при различных испытаниях.

Для управления различными частями отладочных средств ранее использовались различные специальные программы, написанные под каждый процессор отдельно и с разной функциональностью. Эти программы запускались только на Windows-машинах, которые находились рядом с тестируемым процессором «Эльбрус». Данный подход крайне неудобен сразу по нескольким причинам:

- отсутствие стандартизации,
- отсутствие кросс-платформенности,
- необходимость использования различных программ для разных процессоров,
- неудобное расположение машин далеко от рабочего места разработчика.

Для решения этих проблем авторами было решено использовать тот же подход, что и при создании модели JTAG-контроллера. Программа для отладки была разделена на две части: серверную и клиентскую, общающиеся между собой с помощью библиотек YAML и ZMQ.

## СРЕДСТВО ОТЛАДКИ. СЕРВЕРНАЯ ЧАСТЬ

Серверная часть средства отладки процессоров «Эльбрус» управляет JTAG-контроллером и реализует некоторый набор часто используемых функций. Для быстрой работы она была написана на языке C++. Непосредственное управление драйвером JTAG-контроллера было вынесено в отдельный виртуальный класс для возможности использования различных контроллеров различных фирм без изменения кода программы.

Программа-сервер для упрощения работы исследователя сама определяет тип процессора, к которому она подключена. Информация о структуре JTAG-цепочки процессоров заносится в программу заранее, на этапе компиляции. Для определения типа процессора по команде инициализации используется следующий алгоритм.

1. Передача вектора, состоящего из единиц, в регистр инструкций JTAG IR. Длина вектора равна максимальной длине IR всех процессоров, умноженной на максимальное число устройств в цепочке (задается отдельно).
2. Во всех процессорах «Эльбрус» при таком воздействии на выходе получится вектор, соответствующий IR IDCODE, имеющий код 0b00...010...01...01. Так как передавались одни единицы, можно определить реальную длину цепочки IR и вектор для выставления инструкции IDCODE.
3. Для всех подключенных процессоров выставляется IR IDCODE.
4. Выдвигается регистр DR IDCODE.
5. Полученная пара IR и DR IDCODE последовательными кусками сравнивается с известными регистрами процессоров, совпадение означает нахождение данного процессора в цепочке.

После определения структуры цепочки процессоров сервер запоминает ее и отправляет клиенту. При всех обращениях к серверу пользователю не надо задумываться о формировании правильных векторов для существующей цепочки, достаточно передать номер процессора, к которому идет текущее обращение. Сервер при обработке запроса дополнит вектора инструкцией BYPASS для незадействованных процессоров. Также возможно обращение одновременно ко всем процессорам: воздействие будет размножено для всех.

Управление прямой передачей инструкций в ядро процессора для ускорения и облегчения работы программиста было также реализовано в серверной части. Для прописывания значений регистров процессора используется команда STORE. Однако ввиду ограничения на размер команды, возникающего из-за

ограниченной длины регистра DCUR, напрямую записать значение в регистр уровня невозможно. Поэтому необходимо использовать регистры общего назначения ядра «Эльбрус»: сначала необходимое значение прописывается командой ADD в регистр общего назначения ядра (формат команды ADD позволяет передавать данные напрямую через DCUR), а затем это данные из регистра общего назначения передаются в регистр, который нужно прописать. Адрес прописываемого регистра можно передавать как вместе с командой STORE, так и через регистр общего назначения.

Передача значений из регистров процессора в программу с помощью подачи команд невозможна, однако в «Эльбрусе» предусмотрен специальный отладочный регистр DOR. Главное его свойство — он доступен для чтения и записи не только для ядра микропроцессора, но и через отладочный порт JTAG. Чтобы прочитать значение какого-либо регистра системы, следует сначала перенести его на регистр общего назначения ядра командой LOAD, а затем записать содержимое регистра общего состояния в регистр DOR командой STORE. После этого можно получить значение читаемого регистра с помощью JTAG.

Запись и чтение регистров процессора при использовании средства отладки имеют побочный эффект — изменяются значения части регистров общего назначения. Для борьбы с данным эффектом изначальные значения регистров, используемых программой, считываются и запоминаются без участия программиста — сразу после этапа определения типа процессора. Восстановить значения этих регистров в случае необходимости можно специальной командой сервера.

## КЛИЕНТСКАЯ ЧАСТЬ

Клиентская часть представляет собой библиотеку функций для обращения к процессору, которые позволяют абстрагироваться от битовых векторов и JTAG. После вызова функции происходит составление сообщения формата YAML или JSON (любое сообщение JSON является сообщением YAML, но не наоборот), затем данное сообщение передается с помощью библиотеки ZMQ серверу, ответ которого обрабатывается и возвращается в программу.

Клиентская часть была реализована на нескольких языках программирования, включая скриптовые TCL и Python. В языках программирования, поддерживающих ООП, библиотека представляет собой отдельный класс. Использование скриптовых языков позволяет изменять тестовое воздействие без перекомпиляции программы.

## ИСПОЛНЕНИЕ СИСТЕМНЫХ ТЕСТОВ

При проведении испытаний опытных образцов микропроцессора на стойкость к специальным воздействиям не всегда рационально использовать весь вычислительный комплекс, так как, помимо выхода из строя микропроцессора, может отказать один из узлов вычислительного комплекса. Поэтому при испытаниях используются специальные отладочные платы, содержащие только сам микропроцессор, источники питания и JTAG-порт.

Методика некоторых испытаний требует помещения тестируемого микропроцессора в специальную камеру, где происходит воздействие. На микропроцессоре запускается системный тест, по результатам которого в совокупности со сторонними средствами наблюдения можно судить об исправности микропроцессора.

При такой методике испытаний возникает проблема: как без использования полноценного вычислительного комплекса хранить и запускать на микропроцессоре системный тест? Решением является использование кеш-памяти 3-го уровня (L3). Она предназначена для хранения команд и данных достаточно большого объема и доступна для всех ядер процессора. L3 кеш-память уменьшает поток обращений в память и задержку выполнения чтения, отвечает за поддержку когерентности данных [4]. При правильной загрузке в L3 тест можно запустить напрямую из кеша. Тест должен выполнять ряд требований:

1. Обработка всех запросов в память должна завершаться в кеш-памяти 3-го уровня.
2. Не должно быть промахов по кеш-памяти.
3. Программа не должна вытеснять строки из кеш-памяти.
4. Не должно быть обращений к данным другого микропроцессора.
5. Результаты теста должны быть доступны для чтения извне.
6. Программа должна полностью умещаться в кеш-памяти.

Занесение программы в кеш происходит с помощью обращений к ядру микропроцессора. Однако перед использованием кеша его необходимо проверить на наличие неработоспособных строк с помощью встроенного механизма самотестирования BIST.

В качестве исполняемого системного теста может быть использована любая небольшая вычислительная задача с известным результатом, например вычисление числа пи с помощью ряда Лейбница [5]:

$$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots=\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}=\pi/4\,.$$

## НОВИЗНА РЕЗУЛЬТАТОВ

Авторы считают, что в данной работе новым является объединение всех отладочных инструментов в одной программе-сервере с возможностью абстрагироваться от того, как на самом деле происходит доступ к инфраструктуре процессоров. Были созданы удобные в использовании библиотеки для разных языков программирования. Вместе с моделью JTAG-контроллера появилась возможность запускать одну и ту же программу тестирования для реальных процессоров и их моделей без модифицирования кода. Дополнительным преимуществом является кроссплатформенность всех программ и библиотек, что позволяет запускать их на различных машинах. Данный подход продемонстрировал свою состоятельность при отладке памяти и межпроцессорных связей для различных процессоров «Эльбрус» [6].

## ЛИТЕРАТУРА


1. Laung-Tern Wang, Charles E. Stroud, Nur A. Touba. System-on-Chip Test Architectures: nanometer design for testability. — Burlington: Morgan Kaufmann Publishers, 2008. — 856 p.
2. Laung-Terng Wang, Cheng-Wen Wu, Xiaoqing Wen. VLSI Test Principles and Architectures: Design for Testability. — San Francisco: Morgan Kaufmann Publishers, 2006. — 777 p.
3. IEEE Std 1149.1-2001: IEEE Standard Test Access Port and Boundary-Scan Architecture. — New York: Institute of Electrical and Electronics Engineers, 2001. — 208 p.
4. Ким А. К., Перекатов В. И., Ермаков С. Г. Микропроцессоры и вычислительные комплексы семейства «Эльбрус». — Спб.: Питер, 2013. — 272 с.: ил. ISBN 978-5-459-01697-0.
5. Ranjan Roy (1990). The Discovery of the Series Formula for π by Leibniz, Gregory and Nilakantha. Mathematics Magazine 63, 1990. — P. 291–306.
6. Bilyaletdinov I. E., Ometov A. E., Timin L. S. Optimizing Parameters of High-speed Channels of the Processor in order to Increase the Fault Tolerance of the Computer Complex. Nanoindustry 9 (2018). P. 75–78 (In Russian).