УДК 004.3.06

В. Б. Бетелин, акад. РАН, директор, e-mail: betelin@niisi.msk.ru,
В. А. Галатенко, д-р физ.-мат. наук, ст. науч. сотр., зав. сектором, e-mail: galat@niisi.msk.ru,
К. А. Костюхин, канд. физ.-мат. наук, ст. науч. сотр., e-mail: kost@niisi.msk.ru
НИИ системных исследований РАН, Москва

Контролируемое выполнение приложений на многопроцессорных платформах

Разработка качественных сложных аппаратно-программных комплексов — длительный, трудоемкий процесс. Считается, что около половины времени уходит на отладку. Переход на многоядерные процессорные архитектуры сделал использование сильносвязанных систем нормой. Это делает отладку таких систем еще более важной и, одновременно, более сложной. Рассматривается предложенная авторами концепция контролируемого выполнения и ее применение к отладке сильносвязанных многопроцессорных комплексов.

Ключевые слова: *контролируемое выполнение, отладка, мониторинг, трассировка, сильносвязанные системы*

Введение

Специализированные сильносвязанные аппаратно-программные комплексы, предназначенные для решения ответственных задач, таких как обработка потока данных в реальном времени, в общем случае строятся из множества компонентов [1].

Основная особенность сильносвязанных систем — использование сложных, зачастую асинхронных, взаимодействий между компонентами системы. Эта особенность влияет на подходы к отладке систем, определяет выбор инструментов и методов отладки. Сложность отладки также обусловлена числом компонентов сильносвязанных систем, часть из которых могут быть аппаратными. Традиционный набор инструментов отладки в широком смысле (интерактивный отладчик, трассировщик, библиотеки самоконтроля, воспроизведения и отладки производительности) не теряют свою актуальность, но варианты их применения изменяются.

Определение концепции контролируемого выполнения

Аппаратно-программные комплексы создаются для решения определенных задач, достижения заданных целей. Для краткости будем говорить, что у каждого комплекса есть *миссия*.

Контролируемое выполнение — это процесс функционирования аппаратно-программного комплекса, при котором комплекс имеет возможность выполнить свою миссию, несмотря на возможное проявление ошибок, атаки и отказы.

Традиционно отладку и мониторинг относят к этапам разработки, тестирования и отладки систем. Широкое распространение программного обеспечения с открытыми исходными текстами, по крайней мере в принципиальном плане, дает возможность распространить отладку и на этап эксплуатации. Разумеется, чтобы подобное решение стало разумным, необходимо изменение взгляда на эксплуатационную документацию к программным системам, а также на информацию, доступную во время выполнения. Их следует дополнить, предоставив пользователям и отладчику достаточно сведений для анализа нештатных ситуаций и их разрешения.

Для сложных ответственных сильносвязанных комплексов целесообразно постоянно контролировать характер их функционирования, по возможности выявляя и ликвидируя проблемы на ранних стадиях возникновения. Поведение таких комплексов определяется не только программными компонентами, но и наличием доступных ресурсов, сбоеустойчивостью аппаратных компонентов и т. п. Если не располагать регистрационной информацией, то на практике не удастся воспроизвести ситуацию, приведшую к ошибке и, соответственно, не удастся организовать отладку.

Применение средств самоконтроля программ можно рассматривать как распространение объектно-ориентированного подхода на контролируемое выполнение комплексов. Программы в таком

случае трактуются не как пассивные объекты, к которым внешним образом применяются отладочные и управляющие действия с внешней же интерпретацией семантики выполнения, а как активные сущности, самостоятельно контролирующие корректность своего функционирования.

Средства аппаратной отладки

EJTAG. Традиционным способом аппаратной отладки сильносвязанных комплексов является отладка через специальный инженерный порт. В тех комплексах, с которыми имеют дело авторы статьи, для этих целей используется интерфейс EJTAG [2].

EJTAG (Enhanced JTAG) — это спецификация аппаратно-программной подсистемы для семейства процессоров с ядрами MIPS. EJTAG использует инфраструктуру IEEE 1149.1 JTAG [3] в качестве интерфейса доступа к данным и командам и расширяет семейство инструкций ядра MIPS, а также набор компонент ядра. Таким образом создается унифицированная архитектура для отладки встраиваемых систем. Спецификации EJTAG постоянно обновляются.

Первоначально в основу отладочной архитектуры MIPS легла RISC-философия, ориентирующаяся на минимизацию аппаратной составляющей. Первые ядра MIPS определяли небольшой набор примитивов отладки — инструкцию прерывания, порождающую исключение, инструкции проверки значений регистров, пару необязательных регистров слежения за памятью, а также специальный режим устройства управления памятью, позволяющий реализовать защиту от изменения. Все эти механизмы требовали поддержки программного обеспечения, выполняющегося на процессоре, — операционной системы или некоторого монитора, позволяющего вставлять точки прерывания и передавать данные пользователю. Кроме того, внешние анализаторы логики и эмуляторы контуров могли управлять выполнением приложений. Эта модель была достаточно эффективной, но при появлении сложных многоядерных конфигураций с несколькими ядрами на одном чипе, использование внешних интерфейсов процессора стало значительно более сложным. Одним из препятствий стала отладка приложений, выполняющихся из неизменяемой памяти. Другим препятствием стала необходимость наличия специального отладочного порта, что значительно затрудняет разработку встраиваемых систем, увеличивая их стоимость. Важно обеспечить отладку приложения в полевых условиях. Отладка с использованием интерфейса EJTAG позволила решить эти проблемы.

Процессор с поддержкой EJTAG может быть встроен в цепь устройств и отлажен с использованием внешнего оборудования. EJTAG использует стандартный интерфейс для доступа к TAP. Для управления процессором используются регистры

Регистры EJTAG

IMPCODE	Регистр, содержащий информацию о процессоре и реализованных в нем функциях EJTAG
ADDRESS	Регистр для доступа к шине адреса
DATA	Регистр для доступа к шине данных
CONTROL	Регистр управления процессором
ALL	Регистр для одновременного доступа к шинам адреса, данных и управления
EJTAGBOOT	Регистр для установки режима отладки после перезагрузки
NORMALBOOT	Регистр для обычной перезагрузки
FASTDATA	Регистр для ускоренного доступа к данным
IMPCODE	Регистр, содержащий информацию о процессоре и реализованных в нем функциях EJTAG
TCBCONTROL[A-E]	Регистры для доступа к блоку трассировки
TCBDATA	Регистр для доступа к данным трассировки
PCSAMPLE	Регистр для профилирования
FDC	Регистр для доступа к каналу быстрой отладки

контроллера TAP, дополняющие набор стандартных регистров IDCODE, BYPASS и др. Эти регистры приведены в таблице.

Спецификация EJTAG описывает следующие возможности.

Отладочное исключение и отладочный режим. Для того чтобы предоставить доступ к состоянию процессора в любой момент выполнения приложения, в спецификации EJTAG описывается отладочное исключение, после которого процессор переходит в специальный отладочный режим. В этом режиме возможен свободный доступ к сопроцессору, памяти и другим устройствам. Исключения по доступу к адресу и прерывания маскируются. Обработчик отладочного исключения предоставляется отладочной системой или приложением, если это предусмотрено разработчиком. Если это необходимо, разработчик может при наступлении отладочного исключения послать сигнал о нем внешним аппаратным модулям, заставляя их приостановить работу на время отладки.

Внешняя память EJTAG. EJTAG позволяет процессору MIPS в отладочном режиме выполнять инструкции, поступающие из TAP интерфейса. Для этого память EJTAG в специально выделенном сегменте преобразуется в операции над интерфейсом TAP. И данные, и инструкции становятся доступными на инструментальной платформе, что позволяет отлаживать программы в постоянной памяти. Коммуникация с агентом отладки осуществляется посредством регистра CONTROL, бит которого говорит об ожидании процессора. После этого адрес транзакции выставляется в регистре ADDRESS, а данные — в регистре DATA. Бит в ре-

гистре управления обновляется, чтобы сигнализировать, что транзакция совершена.

С частотой 40 МГц это занимает порядка 5 мкс, а скорость передачи данных составляет 800 кбайт/с. Оптимизация передачи может быть осуществлена посредством использования программных методов, таких как предсказание следующего адреса, или с помощью регистра FASTDATA.

Инструкции останова. Спецификации EJTAG определяют новую инструкцию SDBBP, отличающуюся от инструкции BREAK ядер MIPS32 и MIPS64 тем, что ее исключение переводит процессор в отладочный режим и позволяет выполнить обработчик из внешней памяти EJTAG.

EJTAG определяет различные типы аппаратных точек останова. Исключение порождается до того, как действие приводит к изменению состояния процессора или памяти, например, прерывание по выполнению по адресу происходит после того, как инструкция будет извлечена, но до того, как она будет выполнена. Прерывание по записи в память выполняется до того, как запись будет проведена. Аппаратные точки останова срабатывают без изменения памяти и поэтому обладают преимуществом перед программными.

EJTAG определяет два типа простых точек останова:

- останов по выполнению инструкции по заданному виртуальному адресу;
- останов по доступу к данным, причем срабатывающий при некотором значении записываемых/считываемых данных.

Возможна реализация до 15 точек останова различных типов.

Начиная с версии 4.00 спецификации EJTAG определяют сложные точки останова различных типов, например:

- инструкции со счетчиками срабатывания;
- инструкции с условием останова по адресу выполнения и данным одновременно;
- точки останова для заданных процессов;
- точки останова, зависящие от других точек останова;
- точки останова по таймеру.

Точки останова могут сигнализировать о начале трассировки, работа которой определяется спецификацией MIPS PDTrace [4].

Режимы трассировки и профилирования

Спецификации EJTAG также содержат разделы, посвященные аппаратной трассировке PDTrace и аппаратному профилированию. Аппаратная трассировка позволяет передавать данные о выполнении приложения по специальному внешнему каналу на инструментальную машину. Аппаратная трассировка настраивается с помощью регистров TAP.

Поддержка аппаратного профилирования заключается в периодическом сохранении в память инструментальной машины текущей информации о выполнении приложения, такой как адреса инструкций, промахи в кэш-память или записываемые данные в регистр TAP PCSAMPLE, посредством которого она становится доступна на инструментальной платформе.

Аппаратные компоненты

В общем случае аппаратная инфраструктура EJTAG обладает следующими свойствами:

- наличие режима пошагового выполнения;
- наличие порта TAP;
- наличие точек останова, в том числе точек слежения;
- наличие управляющих регистров;
- наличие регистров профилирования;
- наличие канала быстрой передачи данных.

Расширения для многоядерной отладки. Спецификации MIPS MT ASE определяют процессор, состоящий из нескольких виртуальных модулей VPE (виртуальный процессорный элемент). С точки зрения аппаратуры и приложений такой процессор не отличается от набора нескольких процессоров. EJTAG может поддерживать все VPE по отдельности или модуль целиком. В первом случае каждое ядро должно иметь свой контроллер TAP и все они должны быть связаны в единую цепь, как определяет спецификация JTAG 1147.1. При этом значительная часть аппаратной составляющей не разделяется между модулями, каждый имеет свой отладочный регистр, регистр точек останова и т. д. Во втором случае ядра могут иметь один регистр отладки, поддерживать одновременное пошаговое выполнение и другие совместные отладочные действия. Версии спецификаций EJTAG определяют необходимые в этом случае изменения в значения регистров.

Использование средств аппаратной отладки позволяет эффективно отлаживать сильносвязанные комплексы, поскольку имеется возможность практически одновременной остановки всех компонентов комплекса. В этом случае уменьшается риск нарушения их синхронизации.

Отладка аппаратных конфигураций. Современная постановка задачи отладки включает в себя такие аспекты, как отладка специфических аппаратных конфигураций, включающих в себя процессоры цифровой обработки сигналов или графические процессоры (см. Total Recall [5], NUDA [6]). Для отладки сложных сетей на кристалле предназначены инструменты, описанные в работе [7].

Средства программной отладки и мониторинга

Интерактивная отладка. Традиционно под отладкой понимается в первую очередь интерактивная отладка. В то же время чем сложнее отлаживаемая система, тем менее эффективной оказывается по-

добная отладка. Слишком много факторов приходится принимать во внимание, слишком много данных необходимо проанализировать, чтобы ответить, например, на простейшие, но и важнейшие вопросы: случились ли уже в системе ошибки, и каковы их первопричины.

Вероятно, наиболее популярным инструментом интерактивной отладки является GDB [8]. На момент написания данной статьи он оставался однопроцессным; многопроцессные возможности были на начальной стадии реализации и, как показал наш практический опыт, их нельзя было считать работоспособными. В то же время развитие GDB в интересующем нас направлении активно обсуждается в литературе.

В [9] предложена реализация так называемой безостановочной многопоточной отладки. Безостановочность заключается в возможности отладки отдельного потока в процессе в операционной среде Linux, без остановки процесса целиком.

Такая функциональность достигается за счет расширения возможностей агента отладки, а также существенной переработки внутреннего устройства отладчика, создания механизма асинхронного обслуживания событий и контроля применимости отладочных действий. Это шаг очень важен и для развития многопроцессной отладки.

Интерактивной отладке присуща также такая проблема, как внесение недопустимо серьезных возмущений в процесс функционирования отлаживаемых систем. Этот процесс и без отладчика, вообще говоря, является недетерминированным, поэтому заранее нельзя сказать, проявятся ли ошибки в данном конкретном запуске; отладочные действия усугубляют недетерминизм, что способно привести к исчезновению предмета отладки.

Проблеме минимизации возмущений, вносимых отладочными инструментами категории GDB, посвящена статья [10]. В ней описываются различные механизмы, необходимые для многопроцессной отладки. Это централизованный ввод-вывод отладочных сообщений, параллельный сбор данных во время выполнения, использование встроенных в приложение отладочных средств и перехват передачи сообщений между процессами.

Командный язык отладки. При отладке многопроцессных систем возрастает роль командного языка отладчика. Условные точки прерывания и реакция на их достижение могут и должны программироваться заранее. Весьма вероятно, что программы отладки должны затрагивать не только непосредственно отлаживаемый процесс, но и другие процессы, функционирующие, быть может, на других процессорах. Не обязательно видеть состояние сразу всех процессов (для больших систем это не имеет смысла), но необходимо иметь возможность воздействовать на все процессы — по одиночке или группами. Подобное групповое воздействие — обязательная черта многопроцессного отладчика.

В настоящее время в качестве командного языка отладчика применяется интерпретируемый полноценный язык программирования, такой как Python. Поддержка Python появилась в GDB версии 7.0. Наличие полноценного языка с большим числом библиотек позволяет расширить возможности отображения данных, в том числе в графическом формате, организовать взаимодействие с базами данных и реализовать сложные механизмы управления приложением.

Трассировка. Вероятно, основным инструментом отладки многопроцессных систем можно назвать трассировку. Она может иметь различную степень детализации — от машинной инструкции до программных транзакций. При аппаратной поддержке возмущения, вызываемые трассировкой, могут быть приемлемыми, а центр тяжести переносится на последующий анализ собранной информации.

При трассировке с высокой степенью детализации объем собранной информации может исчисляться гигабайтами. Для его уменьшения существуют средства сжатия трасс и, что более существенно, — условные точки трассировки, благодаря которым можно накапливать только информацию, вероятно, имеющую отношение к ошибкам и их проявлениям. В таких точках может также активизироваться интерактивный отладчик, который в данном случае остается средством "ближнего боя".

Помещать события в трассу можно не только внешними средствами, но и из программы, используя механизмы самоконтроля и генерации событий. Подобное оснащение программ представляется наиболее целесообразным, так как оно учитывает семантику программ и предполагаемую первопричину разыскиваемых ошибок, минимизирует возмущения, вносимые в процесс функционирования систем, и объем собираемой информации. И здесь возможна активизация интерактивного отладчика.

Практичный инструментарий трассировки — популярный предмет публикаций и реализаций (см., например, MkTrace [11, 12]).

Самоотладка. Для самоотладки можно предложить следующий подход. Окна интерактивной отладки ассоциируются с некими "представительными" элементами аппаратно-программной конфигурации, но за счет использования групповых операций (глобальных команд gdb) gstop, gcontinue, gdetach, greset, gbreak отладка сильносвязанных систем оказывается реальной.

Многопроцессорные конфигурации, хотя они и создают сложности при отладке, могут в то же время использоваться и как инструмент отладки. При этом на части процессоров функционирует отлаживаемая система, а на других — средства отладки. Поскольку память в сильносвязанных системах явля-

ется, как правило, разделяемой, то возмущения, вносимые отладочными действиями, могут быть минимальными.

Детерминированное воспроизведение выполнения. Запуски, в которых проявляются ошибки, могут не только детерминировано воспроизводиться, но и синтезироваться как результат анализа программ. Вообще, предварительный и "посмертный" анализ программ и данных, собранных в процессе функционирования, позволяет минимизировать вносимые отладкой возмущения.

Вариантом детерминированного воспроизведения может считаться механизм создания контрольных точек останова. Такие контрольные точки могут использоваться для более быстрого воспроизведения ошибки при ее обнаружении. При этом выполнение может быть неполным, но способствовать более быстрому воспроизведению аварийной ситуации.

Интеграция средств аппаратной и программной отладки в единый комплекс контролируемого выполнения

Авторами статьи были разработаны средства аппаратной и программной отладки сильносвязанных систем и их интеграция в единый комплекс контролируемого выполнения.

В состав комплекса вошли компонент аппаратной отладки через инженерный порт, компонент интерактивной отладки, компонент сбора и анализа трассы, компонент самоконтроля и компонент воспроизведения выполнения. Все эти компоненты имеют возможность взаимодействия друг с другом при возникновении определенного события в целевой системе.

Например, компонент самоконтроля в процессе функционирования целевой системы может проверять заданные разработчиком условия на корректность, в частности, контролировать последовательность доступа к общей памяти разных процессоров. Если условия были нарушены, то компонент самоконтроля порождает определенное событие, которое перехватывается и обрабатывается компонентом сбора и анализа трассы. В свою очередь, компонент сбора и анализа трассы может инициировать сеанс интерактивной отладки или аппаратной отладки через инженерный порт. При необходимости, выполнение системы, вызвавшее сбой, может быть воспроизведено с помощью компонента воспроизведения выполнения по событиям, собранным компонентом сбора и анализа трассы.

Заключение

Набор отладочных инструментов, применяемых в отладке сильносвязанных систем, значительно шире набора инструментов традиционных последовательных отладчиков. Это усложнение — необходимость, определяемая сложностью отлаживаемой системы. Без средств визуализации, трассировки, воспроизведения, самоконтроля найти ошибку зачастую невозможно. Эффективная отладка требует применения всего набора инструментов, причем каждый из них должен быть разработан с учетом сложности системы, с использованием вышеперечисленных подходов.

Аппаратная отладка открывает перед разработчиками новые возможности, является необходимым средством разработки приложений для ответственных сильносвязанных комплексов. Стандартизация отладки увеличивает эффективность отладочных средств, помогает их распространению. Таким образом, представляется целесообразным развивать и применять архитектуры аппаратной отладки, основанные на общедоступных спецификациях.

Важно, что подход к многопроцессной отладке требует не только применения различных инструментов, но и их интеграции в отлаживаемую систему, взаимодействия средств отладки с вычислительной средой. Эту новую по сравнению с традиционной отладкой особенность необходимо принимать во внимание не только при разработке отладчика, но и при проектировании приложений.

Список литературы

1. **Timmerman M., Gielen F., Lambrix P.** High level tools for the debugging of real-time multiprocessor systems // ACM SIGPLAN Notices, 1993. Vol. 28, Is. 12. P. 151—157.

2. **MIPS** EJTAG 5.00. URL: http://www.t-es-t.hu/download/mips/md00047f.pdf

3. **IEEE** STD. 1149.1—2001 (IEEE Standard Test Access Port and Boundary Scan Architecture). URL: http://ieeexplore.ieee.org/iel5/481/20326/00938734.pdf

4. **MIPS** PDtrace specification. URL: http://www.t-es-t.hu/download/mips/md00439g.pdf

5. **Ahmad S., Hsien-Hsin L. S.** Total recall: a debugging framework for GPUs // GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware. 2008. P. 13—20.

6. **Chi-Neng W., Shu-Hsuan C., Tien-Fu C., Peisheng S. A.** NUDA: a non-uniform debugging architecture and non-intrusive race detection for many-core // DAC '09: Proceedings of the 46th Annual Design Automation Conference. 2009. P. 148—153.

7. **Mayer A., Siehert H., McDonald-Maier K. D.** Boosting Debugging Support for Complex Systems on Chip // Computer. 2007. Vol. 40, N. 4. P. 76—81.

8. **GNU** Debugger. URL: http://www.gnu.org/software/gdb/

9. **Non-stop** Multi-Threaded Debugging in GDB. URL: http://www.codesourcery.com/publications/non stop multi threaded_ debugging in gdb.pdf

10. **Beynon M. D., Andrade H., Saltz J.** Low-Cost Non-Intrusive Debugging Strategies for Distributed Parallel Programs // IEEE International Conference on Cluster Computing. 2002. P. 439.

11. **Li Y., Wang F., Wang G., Liu X., Liu J.** MKtrace: An Innovative Debugging Tool for Multi-Threaded Programs on Multiprocessor Systems // Software Engineering Conference. APSEC 2007. 2007. P. 510—517.

12. **Moore L. J., Moya A. R.** Non-intrusive debug technique for embedded programming // Software Reliability Engineering. ISSRE 2003. 2003. P. 375—380.

V. B. Betelin, Director, e-mail: betelin@niisi.msk.ru,
V. A. Galatenko, Head of Sector, e-mail: galat@niisi.msk.ru,
K. A. Kostyukhin, Senior Researcher, e-mail: kost@niisi.msk.ru
Research Institute of System Analysis of RAS, Moscow

Controlled Execution of Tightly-Coupled Multiprocessor Systems

Development of complex software and hardware systems is a long laborious process. It is believed that about a half of the development time is spent on debugging. It makes a debugging of tightly-coupled multiprocessor systems more important and more complex.

This article discusses the controlled execution concept proposed by the authors and its application in debugging of tightly-coupled multiprocessor systems.

The main problem in debugging of tightly-coupled systems is in usage of complex asynchronous interactions between its components. This feature offers various approaches to tightly-coupled systems debugging process and determines the choice of tools and methods for debugging. The complexity of debugging is also increases due to a big number of tightly-coupled system components, software and hardware. The traditional debugging tools such as an interactive debugger, a tracer, self-control and playback libraries, and performance profiling does not lose its relevance, but its application is changed.

Keywords: *tightly-coupled multiprocessor systems, controlled execution, debugging, tracing, monitoring, EJTAG*

References

1. **Timmerman M., Gielen F., Lambrix P.** High level tools for the debugging of real-time multiprocessor systems. *ACM SIGPLAN Notices*, 1993, vol. 28, is. 12, pp. 151—157.

2. **MIPS** *EJTAG* 5.00. URL: http://www.t-es-t.hu/download/mips/md00047f.pdf (date: 08/04/2015).

3. **IEEE** *STD.* 1149.1—2001 (IEEE Standard Test Access Port and Boundary Scan Architecture). URL: http://ieeexplore.ieee.org/iel5/7481/20326/00938734.pdf (date: 08/04/2015).

4. **MIPS** *PDtrace specification*. URL: http://www.t-es-t.hu/download/mips/md00439g.pdf (date: 08/04/2015).

5. **Ahmad S., Hsien-Hsin L. S.** Total recall: a debugging framework for GPUs. *GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, 2008, pp. 13—20.

6. **Chi-Neng W., Shu-Hsuan C., Tien-Fu C., Peisheng S. A.** NUDA: a non-uniform debugging architecture and non-intrusive race detection for many-core. *DAC'09: Proceedings of the 46th Annual Design Automation Conference*, 2009, pp. 148—153.

7. **Mayer A., Siebert H., McDonald-Maier K. D.** Boosting Debugging Support for Complex Systems on Chip. *Computer*, 2007, vol. 40, no. 4, pp. 76—81.

8. **GNU** Debugger. URL: http://www.gnu.org/software/gdb/ (date: 08/04/2015).

9. **Non-stop** Multi-Threaded Debugging in GDB. URL: http://www.codesourcery.com/publications/non_stop_multi_threaded_debugging_in_gdb.pdf (date: 08/04/2015).

10. **Beynon M. D., Andrade H., Saltz J.** Low-Cost Non-Intrusive Debugging Strategies for Distributed Parallel Programs. *IEEE International Conference on Cluster Computing*, 2002, P. 439.

11. **Li Y., Wang F., Wang G., Liu X., Liu J.** MKtrace: An Innovative Debugging Tool for Multi-Threaded Programs on Multiprocessor Systems. *Software Engineering Conference*. APSEC 2007, 2007, pp. 510—517.

12. **Moore L. J., Moya A. R.** Non-intrusive debug technique for embedded programming. *Software Reliability Engineering*. ISSRE 2003, 2003, pp. 375—380.

УДК 004.384, 004.31

С. Г. Бобков[1, 2], д-р техн. наук, зам директора, e-mail: bobkov@cs.niisi.ras.ru,
О. В. Сердин, канд. техн. наук, зам. зав. отделения, e-mail: serdin@cs.niisi.ras.ru
[1] НИИ системных исследований РАН,
[2] НИЯУ МИФИ

Проблемы унификации микросхем космического применения на примере международного проекта космического телескопа "ГАММА-400"

Рассмотрены аспекты унификации электронных компонентов различных уровней: микросхем, интерфейсов, модулей, программного обеспечения, применяемых в космическом приборостроении. Представлена система сбора научной информации, разрабатываемая НИИСИ РАН для проекта космического телескопа "ГАММА-400". Даны основные характеристики комплекта высоконадежных и сбоеустойчивых микросхем разработки НИИСИ РАН, которые позволяют проектировать различные электронные системы космической аппаратуры.

Ключевые слова: *"ГАММА-400", система сбора научной информации, КНИ-технология, система на кристалле, 8-портовый коммутатор SpaceWire, 6-портовый коммутатор Serial RapidIO, интерфейс SpaceWire, интерфейс Serial RapidIO.*