НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
март–апрель 2017 Том 17 № 2 ISSN 2226-1494 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
March–April 2017 Vol. 17 No 2 ISSN 2226-1494 http://ntv.ifmo.ru/en

УДК 004.052

СТОРОЖЕВЫЕ МЕХАНИЗМЫ ВО ВСТРАИВАЕМЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

А.Е. Платунов[a], А.С. Стерхов[a]

[a] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
Адрес для переписки: aeplatunov@gmail.com

Информация о статье
Поступила в редакцию 13.02.17, принята к печати 01.03.17
doi: 10.17586/2226-1494-2017-17-2-301-311
Язык статьи – русский

Ссылка для цитирования: Платунов А.Е., Стерхов А.С. Сторожевые механизмы во встраиваемых вычислительных системах // Научно-технический вестник информационных технологий, механики и оптики. 2017. Т. 17. № 2. С. 301–311. doi: 10.17586/2226-1494-2017-17-2-301-311

Аннотация

В статье приведен обзор сторожевых механизмов, которые представляют собой категорию технических решений, направленных на повышение надежности работы встраиваемых вычислительных систем. Зарубежные публикации в основном посвящены частным проблемам построения и использования сторожевых механизмов. В русскоязычных источниках информация, систематизирующая представление о данных механизмах, отсутствует. Недостаточная осведомленность инженеров в вопросах, касающихся этих важных и сложно интегрируемых в систему элементов, приводит к их неэффективному, а зачастую даже неграмотному использованию. Это, в свою очередь, способно привести к снижению надежности. В статье представлена базовая модель, отражающая принцип работы сторожевого механизма. Проведен анализ представленных в литературе технических решений, используемых для детектирования ошибок. Предложена их классификация. В компактной табличной форме представлены основные свойства данных решений. Рассмотрены различные варианты действий при обнаружении ошибок. Приведен обзор технических решений, направленных на повышение эффективности сторожевых механизмов. Важнейшей проблемой на сегодня представляется отсутствие средств автоматизации процесса, которые по заданным критериям (время обнаружения ошибки и время реакции на нее) позволяли бы эффективно и грамотно интегрировать сторожевой механизм в проектируемую аппаратно-программную систему. Эти вопросы в форме постановки задач для будущих исследований обсуждаются в заключительных разделах работы. Статья адресована исследователям и разработчикам, занимающимся вопросами проектирования встраиваемых систем и повышения надежности их функционирования.

Ключевые слова

сторожевые механизмы, надежность встраиваемых систем, сторожевые таймеры, сторожевые процессоры, встраиваемые системы

WHATCHDOG MECHANISMS IN EMBEDDED SYSTEMS

A.E. Platunov[a], A.S. Sterkhov[a]

[a] ITMO University, Saint Petersburg, 197101, Russian Federation
Corresponding author: aeplatunov@gmail.com

Article info
Received 13.02.17, accepted 01.03.17
doi: 10.17586/2226-1494-2017-17-2-301-311
Article in Russian

For citation: Platunov A.E., Sterkhov A.S. Whatchdog mechanisms in embedded systems. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2017, vol. 17, no. 2, pp. 301–311 (in Russian). doi: 10.17586/2226-1494-2017-17-2-301-311

Abstract

The paper provides an overview of the watchdog mechanisms which are a category of technical solutions aimed at reliability improvement of embedded computing systems. The works presented in the foreign literature are mainly devoted to partial problems of watchdog mechanisms creation and application. The Russian-language sources are free of information, systematizing understanding of these mechanisms. Engineers' lack of awareness in matters relating to these elements, important and difficult for integration into system, leads to their inefficient and even unskilled usage. This, in turn, can reduce reliability. The paper deals with a basic model that reflects operation principle of the watchdog mechanism. The analysis of technical solutions presented in literature and used for error detection was carried out. Their classification was proposed. The compact table form shows the main properties of these solutions. Review of solutions improving the efficiency of watchdog mechanisms was made. Today the most important problem is the lack of process automation tools which would give the

possibility to integrate the watchdog mechanism effectively and correctly in projected hardware-software system in view of specified criteria (error detection time and response time). These questions are discussed in the final sections of this paper in form of setting targets for future research. The paper is addressed to researchers and developers concerned with embedded systems design and improvement of their operational reliability.

Keywords
watchdog mechanisms, embedded system reliability, watchdog timers, watchdog processors, embedded systems

Введение

Встраиваемые вычислительные системы широко используются в областях, где необходимо управление ответственными процессами, сложными объектами и системами. При создании встраиваемых систем необходимо учитывать факторы, которые влияют на качество их функционирования и присутствуют не только на этапах производства, эксплуатации, но и на этапе проектирования, так как сложность аппаратно-программных систем в совокупности со сложностью инструментария не позволяют на практике исключить все ошибки и некорректности [1]. Одним из наиболее эффективных путей, позволяющих парировать в режиме функционирования (run-time) перечисленные проблемы, является включение в состав встраиваемой системы контуров повышения показателей надежности (отказоустойчивости, функциональной безопасности и других) на основе сторожевых механизмов. Такие средства, наиболее известными из которых можно считать сторожевые таймеры (watchdog timer), могут использоваться на всех уровнях организации встраиваемой системы, реализуемых аппаратным, программным или комбинированным способом.

Практика показала, что даже при должном внимании к качеству продукции удается выявить в среднем 95% ошибок программного обеспечения при постоянно растущем объеме программного кода [2]. С другой стороны, даже отлично спроектированное оборудование, на котором выполняется идеально отлаженное программное обеспечение, все равно подвержено отказам. Причиной этому может стать, например, радиация. Взаимодействуя с транзисторами микросхем, заряженные частицы вызывают их переключение и изменяют состояние ячеек памяти [3–5]. К отказам во встраиваемых системах могут приводить и другие факторы: повышенная или пониженная температура, электромагнитные излучения, некачественное электропитание и так далее.

Все вышесказанное обусловливает необходимость осуществления непрерывного контроля функционирования критически важных с точки зрения безопасности систем (safety-critical systems) [6], а также всех других категорий встраиваемых систем, доступ к которым затруднен или невозможен. Следует заметить, что использование сторожевых механизмов на практике полезно для всех категорий встраиваемых систем.

К сожалению, вопросы проектирования этих важных и сложно интегрируемых в систему элементов недостаточно освещены в литературе и практически не поддержаны методологически и инструментально.

Обзор источников, рассматривающих различные вопросы построения, применения и анализа эффективности сторожевых механизмов, позволил выделить следующие аспекты. В [7–25] приводится информация, касающаяся методов детектирования ошибок, используемых в сторожевых механизмах. В источниках [7, С. 144–169; 26] можно найти сведения, касающиеся методов восстановления работы системы. Примеры реализаций сторожевых механизмов продемонстрированы в [8, 10–15, 21, 24, 27–30]. Вопросы методологии их проектирования и применения более подробно рассмотрены в [17–20, 22, 23, 26]. В ряде публикаций рассматриваются методы анализа работы вычислительной системы со сторожевыми механизмами на базе аналитического и имитационного подходов [10, 12–16, 27, 29,31].

По данному направлению не наблюдается обилия литературы, особенно отражающей системные, методические и инструментальные вопросы инженерного применения сторожевых механизмов во встраиваемых системах. Частный взгляд на сторожевые механизмы, представленный в каждом из упомянутых источников, затрудняет построение обобщенной картины. Дополнительную сложность для отечественных специалистов создает тот факт, что подавляющая часть материалов, посвященных сторожевым механизмам, опубликована в зарубежных работах. В русскоязычной литературе имеется дефицит исследований и работ, посвященных данной теме.

Целью настоящей работы является обзор и анализ технических решений в вопросах проектирования сторожевых механизмов встраиваемых систем, функционирующих в реальном времени. На основе данного анализа, практического опыта авторов в создании критически важных вычислительных систем и анализа тенденций в развитии встраиваемых систем и систем на кристалле сделана попытка сформулировать важные и перспективные научно-исследовательские задачи в рассматриваемой области.

В работе не затрагиваются вопросы аналитического и имитационного моделирования работы сторожевого механизма, так как это выходит далеко за рамки одной журнальной статьи. При этом необходимо отметить, что корректность принимаемой математической модели сторожевого механизма в совокупности с реализуемым вычислительным процессом, определяющая результаты анализа, вытекает из глубины понимания исследователем именно технических аспектов реализации и работы вычислительной системы.

Роль сторожевого механизма в работе встраиваемой системы

На рис. 1 приведена модель, полученная на основе анализа ГОСТ 27.002-89[1], [7, С. 52–55; 8, С. 4–6; 32, 33]. Она позволяет продемонстрировать роль сторожевого механизма в задаче обеспечения надежного функционирования встраиваемой системы. Модель отражает взаимосвязь таких ключевых терминов, как неисправность (fault), ошибка (error) и отказ (failure). Необходимо отметить, что термин «ошибка» отсутствует в ГОСТ 27.002-89, но его введение виделось полезным для описания задачи обеспечения надежного функционирования встраиваемой системы.

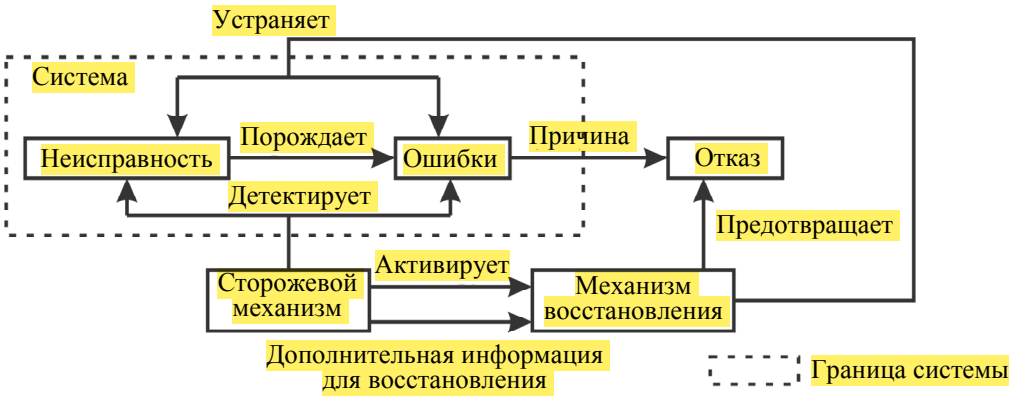

Рис. 1. Функции сторожевого механизма в задаче обеспечения надежного функционирования встраиваемой системы

Неисправности могут присутствовать в системе в пассивном состоянии, не влияя на ее функционирование, но при определенных условиях происходит их активация, приводящая к генерированию ошибок в вычислительном процессе. Ошибки, распространяясь по системе, могут трансформироваться и приводить к другим ошибкам. До определенного момента они влияют только на внутреннее состояние системы, но, достигнув ее границ, становятся причиной нарушений взаимодействия системы с ее окружением. В последнем случае имеет место отказ системы.

В [7, С. 52–55; 33] выделяются основные шаги, направленные на борьбу с отказами и повышение надежности встраиваемых систем.

Первый шаг – это обнаружение ошибок и неисправностей, т.е. установление факта их присутствия в системе. При этом детектирование неисправностей, находящихся в пассивном состоянии, одна из наиболее сложных задач, требующая применения специальных методов. Второй шаг заключается в локализации и изоляции ошибок, установлении тех мест в системе, где они произошли. Третьим шагом является идентификация ошибок, определение степени их серьезности, а также оценка ущерба, который нанесен системе. Возможно, на этом шаге будут выявлены и другие ошибки. Четвертый шаг включает в себя устранение ошибок и их последствий для продолжения штатной работы системы. И, наконец, пятый шаг состоит в идентификации и устранении неисправности, приведшей к ошибкам.

Необходимо сразу же оговориться, что такое деление является условным. Например, некоторые шаги могут объединяться между собой, а пятый шаг, являясь наиболее сложным, во многих случаях может быть реализован лишь частично. Представленное деление демонстрирует высокую комплексность задачи обеспечения надежного функционирования встраиваемой системы.

Вся эта совокупность мер направлена на предотвращение отказов системы и начинается с обнаружения ошибок и неисправностей – задачи, решаемой с помощью сторожевых механизмов. Они же активируют процедуры восстановления нормального функционирования системы. Эти базовые, но не единственные функции сторожевых механизмов отражены на рис. 1. Сторожевые механизмы могут быть задействованы для решения задач любого из обозначенных ранее шагов – например, для сбора информации, требующейся для восстановления системы.

Методы обнаружения ошибок и действия при их обнаружении

В основе работы сторожевых механизмов лежат методы обнаружения ошибок. Существует большое разнообразие таких методов, но все они объединяются общей идеей. Суть ее состоит в сопоставлении текущих показателей системы с теми их значениями, которые предсказывает некоторая модель этой системы. При расхождении значений делается вывод о присутствии в системе ошибки. Данная концепция отражена на рис. 2.

[1] ГОСТ 27.002-89. Надежность в технике. Основные понятия. Термины и определения. Введ. 01.07.90. М.: Изд-во стандартов, 1990. 32 с.

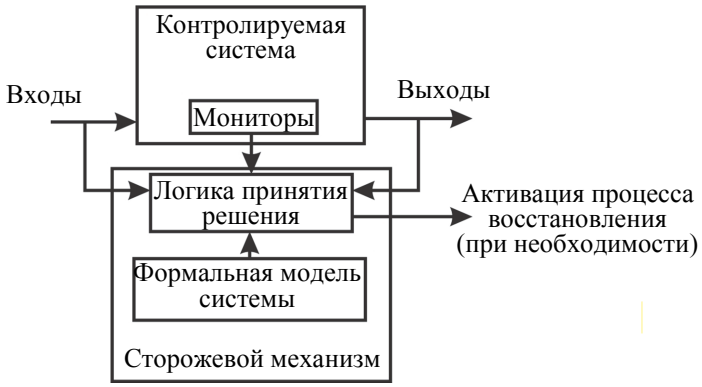

Рис. 2. Принцип работы сторожевого механизма

Под мониторами понимаются аппаратные и (или) программные компоненты, позволяющие получать информацию о внутреннем состоянии контролируемой системы. На основании этой информации, предсказаний формализованной модели системы, а также данных с входов и выходов системы сторожевой механизм делает вывод о наличии или отсутствии в ней ошибок и неисправностей.

В работе монитора важнейшее значение имеет время накопления информации для принятия решения о состоянии вычислительного процесса (интервал или период времени, необходимый для обнаружения и (или) идентификации ошибки). Следует отметить, что именно задача выбора параметров работы монитора и метода его реализации наименее формализована, не обеспечена инженерными методиками и инструментами. Ниже в табл. 1 и 2 представлены методы обнаружения ошибок, используемые в сторожевых механизмах.

Метод (краткое описание)	**Достоинства**	**Недостатки**	**Особенности**
Тестирование / диагностика компонентов системы (diagnostic check) [7, С. 100–114] (Проведение специализированных диагностических тестов компонентов системы)	Высокая точность при локализации и идентификации постоянных ошибок и неисправностей. Позволяет произвести оценку ущерба, нанесенного системе.	Требует значительного времени. Нормальная работа системы при проведении проверки возможна только при наличии резервирующих/избыточных ресурсов. Не подходит для оперативного выявления ошибок и неисправностей. Не подходит для выявления кратковременных ошибок и неисправностей.	Применяется, главным образом, для поиска неисправностей в аппаратных компонентах. Дает хорошие результаты в сочетании с другими, более оперативными методами выявления ошибок и неисправностей для их более точной локализации и идентификации. Требуется максимальная схожесть тестов с реальными режимами работы компонентов. Возможно проведение стрессовых тестов для выявления скрытых неисправностей.
Использование резервирования (replication check) [7, С. 100–114; 8, С. 30–35] (Использование аппаратной (структурной), программной, информационной, временной избыточности системы для сравнения данных с их копиями или результатов нескольких вычислений с целью выявления в них расхождений)	Простота реализации. Возможность выявления ошибок этапа функционирования практически всех типов. Оперативность выявления ошибок и неисправностей. Позволяет не только выявлять ошибки и неисправности, но и оперативно восстанавливать работу системы.	Самые высокие затраты, связанные с необходимостью иметь резервирующие ресурсы. Плохо выявляет ошибки этапа разработки.	В сочетании с тестированием компонентов системы позволяет с высокой точностью диагностировать и идентифицировать неисправности. Используется в высоконадежных и критически важных системах.

Таблица 1. Методы обнаружения ошибок на основе тестирования и резервирования ресурсов встраиваемой системы

Методы первой группы направлены на контроль внутренних компонентов системы без учета задачи, которую решает система. Методы второй группы основаны на проверке корректности выполнения целевого алгоритма (фрагментарно или в целом). Стоит отметить, что некоторые из методов позволяют не только обнаруживать ошибки, но и диагностировать их причины, а иногда и идентифицировать источники неисправности.

В табл. 2, в свою очередь, представлены две большие группы сторожевых механизмов – сторожевые таймеры (watchdog timer / WDT) и сторожевые процессоры (watchdog processor / WDP).

Сторожевой таймер представляет собой счетчик, который ведет обратный отсчет (декрементный

счетчик) от какого-либо начального значения до нуля (возможна также реализация в виде растущего инкрементного счетчика, в этом случае счетчик растет до момента своего переполнения). Признаком ошибки для сторожевого таймера является его обнуление/переполнение. Контролируемая система должна периодически обновлять значение счетчика, не позволяя ему обнулиться/переполниться. Сторожевой процессор является развитием идеи сторожевого таймера и используется для комплексной реализации различных методов поиска ошибок.

При решении задач детектирования ошибок важно учитывать время их существования, так как это напрямую сказывается на эффективности методов их обнаружения. В соответствии с этим свойством можно выделить кратковременные (transient) ошибки, если время их существования весьма незначительно по отношению ко времени функционирования системы. Если же время их существования сопоставимо со временем функционирования системы, то говорят о постоянных (permanent) ошибках.

Метод (краткое описание)	**Достоинства**	**Недостатки**	**Особенности**
1. Реверсивная проверка (reversal check) [7, С. 100–114] (На основе значения, полученного на выходе, делается предположение о значении на входе, после чего эти значения сравниваются. Выявляется несоответствие рассчитанного значения входа и его текущего значения)	Простота реализации и минимальность затрат.	Подходит только для систем, в которых имеется взаимно однозначное соответствие между входами и выходами (отношение один к одному).	Ограниченная применимость метода.
2. Кодирование (coding check) [7, С. 100–114] (Проверяется соответствие текущих значений кодов для данных и команд с их заранее вычисленными эталонными значениями. Выявляется их несоответствие)	Позволяет контролировать большие объемы данных. Оперативное выявление и локализация ошибок и предотвращение их распространения по системе. В некоторых случаях позволяет восстановить данные.	Требует вычислительных ресурсов и дополнительной памяти.	Необходимо учитывать характер ошибок в системе, чтобы выбрать наиболее подходящий алгоритм кодирования.
3. Проверка функциональной корректности (reasonableness check) [7, С. 100–114] (Использование знаний о системе с целью проверки корректности значений некоторых величин. Выявляется несоответствие ожидаемым значениям или невыполнение логических выражений – ассертов (assertions))	Простота реализации. Оперативность выявления ошибок и возможность их локализации.	Требуются дополнительные вычислительные ресурсы и память, тем больше, чем больше величин требуется проверять.	Необходим тщательный выбор контролируемых величин для поддержания баланса затрат и надежности метода. На начальных этапах жизненного цикла системы может контролироваться большее число величин.
4. Обработка исключений (exception handling) [9, С. 173–177] (Выявление особых (например, деление на 0) ситуаций, свидетельствующих о наличии ошибок в системе или тех, что потенциально могут стать их причиной)	Простота реализации. Оперативность выявления и устранения ошибок, а также возможность предотвращать их появление. Позволяет диагностировать ошибки.	Требуются дополнительные вычислительные ресурсы и память.	Позволяет не только диагностировать, но и оперативно устранить ошибки или даже предотвратить их появление. Как правило, применяется в комбинации с другими методами.
5. Проверка доступа к памяти (memory access checking) [8, С. 169–171] (Проверяется корректность доступа к памяти со стороны различных системных объектов)	Оперативность выявления ошибок. Не оказывает влияния на производительность системы, так как реализована на базе сторожевого процессора. Относительная простота реализации, а значит, и невысокая стоимость решения.	Невозможно выявить некоторые типы ошибок, например, разрешенный, но не соответствующий текущим условиям доступ к определенному участку памяти.	Применение может быть ограничено особенностями аппаратной составляющей системы.
6. Проверка целостности структуры (structural check) [7, С. 100–114] (Проверяется целостность структур данных, таких как списки, деревья, очереди и т.д.)	Простота реализации. Возможность локализации ошибок и оперативность в их выявлении.	Требуются дополнительные вычислительные ресурсы и память.	Чем сложнее структура и процесс ее модификации, тем она уязвимее для ошибок, но тем проще выявить факт наличия ошибки.

Метод (краткое описание)			Достоинства	Недостатки	Особенности
7. Проверка потока управления (control-flow checking)	7.1. Проверка количества инструкций (checking of number of fetched instructions) [10] (Проверяется соответствие количества выполненных инструкций и их заранее подсчитанного количества для каждого программного блока)		Оперативность выявления ошибок. Не оказывает влияния на производительность системы, так как реализована на базе сторожевого процессора. Относительная простота ее реализации, а значит, и невысокая стоимость решения.	Не способна выявить некоторые типы ошибок, например, перестановку инструкций, а также некоторые нештатные переходы между блоками кода. Для реализации требуется поддержка со стороны аппаратуры для возможности подсчета инструкций.	Для достижения эффективности применяется с другими методами обнаружения ошибок.
	7.2. Сигнатурный анализ (signatures techniques) [8, С. 155–169; 11–15]	7.2.1. Назначенные сигнатуры (assigned-signatures techniques) (Проверяется поток управления путем сопоставления информации о разрешенных переходах между блоками и сигнатурами (уникальными метками) текущего и предыдущего блоков. Сигнатуры назначаются)	Оперативное выявление ошибок и их локализация. Реализация проще, чем для метода вычисленных сигнатур.	Невозможность выявления ошибок, происходящих внутри блока инструкций.	Широкий спектр решений. Разные решения требуют разной степени модификации целевой программы и (или) аппаратуры, а также разного количества дополнительных системных ресурсов. Требуется поддержка со стороны аппаратного обеспечения. Методы ориентированы на поиск ошибок в потоке управления, но не ошибок в данных. Таким образом, желательно сочетать их с методами, выявляющими такие нарушения, например, использовать резервирование [13, 14].
		7.2.2. Вычисленные сигнатуры (derived-signatures techniques) (Помимо проверки правильности следования блоков инструкций позволяет проверить правильность инструкций и порядок их следования в блоке вычислением сигнатуры блока из кодов инструкций и ее сравнением с заранее вычисленной сигнатурой блока. Сигнатуры вычисляются на основе кодов инструкций блока)	Оперативное выявление ошибок и их локализация. Возможность выявления ошибок, возникающих внутри блока инструкций. Высокая эффективность выявления ошибок в потоке управления [16].	Требует дополнительных вычислительных мощностей, для расчета текущего значения сигнатуры блока инструкций.	
	7.3. Проверка временных ограничений (time checking) [7, С. 100–114; 17–25] (Проверяется выполнение временных ограничений, накладываемых на различные операции и процессы)		Простота реализации и минимальность затрат. Высокая универсальность метода. Подходит для поиска как программных, так и аппаратных ошибок и неисправностей. Возможность оперативного выявления ошибок.	Отсутствие сигнала об ошибке не может гарантировать ее отсутствия в системе (не может самостоятельно выявлять многие типы ошибок).	Широко используемый метод детектирования ошибок. Чем точнее определены временные ограничения, тем выше эффективность метода. Его сочетание с другими методами позволяет добиться их большей эффективности [16]. Особую значимость имеет для систем реального времени. Реализуется на базе сторожевого таймера.

Таблица 2. Методы проверки корректности выполнения целевого алгоритма

В случае детектирования ошибок с помощью одного из описанных ранее методов сторожевой механизм инициирует действия, направленные на восстановление нормальной работы встраиваемой системы или перевода ее в безопасный режим. В ходе работы с источниками были выделены возможные варианты действий, приведенные в табл. 3. Описанные в ней действия могут выполняться как специальным механизмом восстановления, которому сторожевой механизм отправляет сигнал активации, так и самим сторожевым механизмом.

Видится целесообразным сочетание нескольких вариантов действий, представленных в табл. 3. Например, в системе для более быстрого восстановления в качестве основной может использоваться коррекция. Если коррекция невозможна или не дает результата, может применяться перезагрузка системы. В случае перезагрузки системы сторожевой механизм может вести учет количества перезагрузок. Если их число превысит некоторое допустимое значение (при зацикливании перезагрузки), сторожевой механизм может выполнить остановку системы или же переключить систему в режим минимальной функциональности.

Действие (краткое описание)	**Достоинства**	**Недостатки**
1. Возврат к точке восстановления (backward error recovery) [7, С. 144–149, С. 151–169] (Устранение ошибки за счет возврата системы в состояние, предшествующее ее возникновению. При реализации метода может быть использована перезагрузка системы (reset recovery) [26, 27])	Прост в реализации. Позволяет восстанавливать систему, даже не имея точных оценок вреда, причиненного системе ошибкой. Универсален (подходит для устранения большинства ошибок и применим во многих системах). Не требует точной локализации и идентификации самой ошибки.	Большие затраты на реализацию: необходима организация системы периодического сохранения информации для возможности последующего восстановления системы; сама информация для восстановления требует дополнительных, порой весьма значительных, ресурсов памяти. Не все элементы системы можно вернуть в исходное состояние (например, вышедшую из строя линию передачи данных). Опасность зацикливания процедуры восстановления в случае постоянных неисправностей, которые могут приводить к повторному появлению тех же ошибок.
2. Коррекция (forward error recovery) [7, С. 144–151] (Устранение ошибки за счет корректирования состояния системы)	Не требует дополнительных ресурсов памяти для хранения информации о предыдущем состоянии системы. Выполняется быстрее, чем возврат к точке восстановления. Может помочь в случае компонентов, предыдущее состояние которых невозможно восстановить (например, после оценки ущерба может быть выполнена попытка обхода поврежденного участка линии передачи по запасной линии).	Требует точной предварительной оценки степени ущерба, а также локализации и идентификации ошибок. Не универсален, так как тесно связан с особенностями конкретной системы.
3. Остановка системы (fail-silent recovery) [26] (Перевод системы в безопасное состояние и ее дезактивация)	Исключает опасность зацикливания восстановления даже в случае постоянных неисправностей. Позволяет исключить переход системы в неуправляемый режим, грозящий опасными последствиями.	Для дальнейшего восстановления системы все еще нужны дополнительные действия, например, вмешательство оператора. Неприемлем для систем, которые управляют критически важными процессами и не могут быть остановлены.
4. Частичное восстановление (limp-home recovery) [26] (Поддержание минимально необходимой функциональности системы)	Позволяет обеспечить непрерывное функционирование критически важных систем даже в случае возникновения в них ошибок и (или) неисправностей.	Наиболее сложный для реализации вариант, требующий дополнительных ресурсов.

Таблица 3. Действия при обнаружении ошибок

Способы повышения эффективности сторожевых механизмов

Существуют разные подходы к повышению эффективности сторожевых механизмов. Часть из них сформулирована в виде требований и рекомендаций по их проектированию и реализации. Например, с требованиями и рекомендациями к сторожевым таймерам можно ознакомиться в источниках [17–20, 22, 23]. Следование перечисленным в них правилам позволит избежать многих ошибок и получить действительно эффективное решение.

Одним из направлений повышения эффективности сторожевых механизмов является снижение их избыточности. Оно связано с решением сложной задачи обеспечения необходимого и достаточного уровня контроля при минимуме затраченных средств. Решение этой задачи целесообразно начинать с градации неисправностей и ошибок по степени серьезности их последствий [34, С. 452–453]. Данная информация может быть использована в дальнейшем для выделения в системе наиболее важных объектов и процессов, требующих более тщательного контроля со стороны сторожевых механизмов. В [31] предложено выделение критического пути в программе. В программе могут выделяться наиболее важные блоки кода на основе частоты исполнения (чем чаще исполняется, тем важнее) [14, 29, 30], размера блока [29] (чем больше, тем важнее), частоты исполнения и размера блока [29], а также выбора пользователя [14]. Также возможно выделение наиболее важных переменных на основе вычисления некоторой метрики [13, 14].

Другое направление повышения эффективности связано с глубоким и комплексным знанием особенностей аппаратного и программного обеспечения целевой системы. Оно является необходимым при реализации некоторых методов проверки. Например, в [10] для подсчета количества выполненных инструкций используется встроенный в процессор монитор (performance monitoring feature). В [15] дублирование основного процессора поддерживается встроенной в него функцией «основной/проверяющий» (master/checker). Для получения информации об исполняемых процессором инструкциях в [29] предложено использование встроенных в него средств, применяемых при отладке. В [22] для возможности контроля ожидающих задач используется их опрос. Реализация механизма опроса ожидающих задач возможна при наличии соответствующих возможностей в операционной системе.

Эффективность сторожевых механизмов может быть повышена за счет динамического получения и обновления информации, используемой ими при работе. Например, в [29] в памяти сторожевого процессора хранится динамически обновляемая информация о наиболее часто исполняемых блоках инструкций. Это позволяет контролировать большие по объему программы без увеличения памяти сторожевого процессора. В [17] предложен метод сбора статистики по запускам различных задач с целью использования этой статистики в сторожевом механизме. Такой подход может помочь в случае с детектированием ошибок в задачах, запуск которых не детерминирован.

Для повышения эффективности сторожевых механизмов широко используется комбинирование различных методов поиска и устранения ошибок и неисправностей. В методе из [11] сочетаются сигнатурный анализ и проверка корректности с использованием логических утверждений (assertions). В решениях из [13, 14] к методу сигнатурного анализа добавлен метод дублирования наиболее важных переменных. В [13] также предусмотрено дублирование передачи наиболее значимых данных. В источнике [10] описывается механизм, в котором реализованы методы подсчета количества инструкций, сигнатурного анализа и проверки временных ограничений. В [15] к нему добавлено дублирование основного процессора и троирование сторожевого процессора.

Неисправности и ошибки могут происходить на разных уровнях системы, поэтому эффективный сторожевой механизм также должен быть многоуровневым. В [12] проверка программ осуществляется на нескольких уровнях – уровне отдельных инструкций, уровне процедур и уровне процессов. Это повышает возможности системы по выявлению ошибок, позволяет сократить время их обнаружения и повысить точность их локализации и идентификации. Процесс восстановления также может быть разделен на уровни. Этот принцип используется в многоуровневых сторожевых таймерах [21]. В [7, С. 144–149] приводится пример шести уровней восстановления, используемых в системе ESS.

Самостоятельным блоком проблем является сокращение времени восстановления системы после обнаружения нештатной ситуации. Это комплексная проблема, которая должна решаться, в первую очередь, на общесистемном уровне с анализом алгоритмов реализации прикладной функциональности, микроархитектуры системной платформы и адекватного объема сторожевых механизмов.

Проблемы реализации сторожевых механизмов во встраиваемых системах

Проектирование и реализация эффективных сторожевых механизмов требует от разработчика большого количества сил, времени и знаний. К сожалению, на практике проектированию сторожевых механизмов уделяется недостаточно внимания, особенно в проектах с так называемыми «стандартными» требованиями по надежности. Это не способствует их качественной реализации, и на выходе получается ненадежное или даже нерабочее решение. Некорректно построенный сторожевой механизм может даже снизить исходный уровень надежности. Объясняется такая ситуация отсутствием четкой и компактной методики внедрения сторожевых механизмов в функциональность проектируемой встраиваемой системы.

Это подтверждает анализ как специальной литературы, так и большого числа проектов и изделий в сегменте встраиваемых систем. Кроме того, наряду с методиками разработчику нужны инструменты, которые помогали бы генерировать эффективное решение для конкретной системы на основании ее особенностей и существующих методов обнаружения ошибок и восстановления работы системы. Эти инструменты должны учитывать ограничения методов, позволяя грамотно их комбинировать для достижения требуемой эффективности.

Свойства значительной части разрабатываемых встраиваемых систем, такие как многоуровневая распределенная организация и гетерогенность входящих в их состав физических и виртуальных процессоров, позволяют определить следующие приоритетные задачи в части повышения качества и снижения трудоемкости применения сторожевых механизмов.

1. Разработка модели сторожевых механизмов в привязке к типовым уровням организации встраиваемых систем и к технологиям, характерным для соответствующих уровней [35, 36].
2. Унификация таких моделей в направлении поддержки перспективных методик проектирования категории HW/SW Codesign (аппаратно-программное сопряженное проектирование) [37], в которых акцент сделан на расширении фазы проектирования, инвариантного к способу конечной реализации.

3. Выделение кросс-уровневых сторожевых механизмов и предложение для них языка описания (спецификирования) и интеграции в технологические и инструментальные средства создания основных уровней организации встраиваемых систем, например, на базе аспектного подхода [38].

Целый ряд сложных задач необходимо решить в рамках создания технологий использования сторожевых механизмов в составе встраиваемых систем с реконфигурируемой архитектурой [39–42]. Еще одной задачей следует считать развитие методов оценки качества реализации интегрируемых в проект сторожевых механизмов, представленных на уровне детализации, максимально приближенном к уровню инженерных решений. Сегодня получили развитие в основном методы расчета показателей надежности для решений верхнего, архитектурного уровня представления вычислительных систем. В рамках высокоуровневого представления необходимо уметь выделять критически важные локальные механизмы и обеспечивать их качественную верификацию на последующих этапах инженерной реализации.

Заключение

Компоненты, отвечающие за корректное и надежное функционирование любой вычислительной системы, являются сегодня обязательными. Широкий диапазон требований к встраиваемым системам по критерию цена/качество заставляет проектировщиков использовать очень разные по сложности сторожевые механизмы. Это сторожевые таймеры, сторожевые и сервисные процессоры, системные мониторы с аппаратной, программной или комбинированной реализацией. Наличие в составе массовых и специализированных микроконтроллеров, а также в ядрах операционных систем заготовок для реализации сторожевых механизмов при отсутствии понятной и доступной методики их применения приводит в большом числе случаев к провалу проектов или выпуску продуктов низкого качества. Устранение проектных ошибок, связанных с нестабильной или ненадежной работой изделия, – очень затратная задача.

Необходима систематизация обширного международного опыта построения и использования сторожевых механизмов во встраиваемых системах, представление этого опыта в русскоязычной литературе. Наряду с этим видится перспективным предложение массовому разработчику спектра шаблонов инженерных решений сторожевых механизмов в рамках популярных микропроцессорных семейств и системных вычислительных платформ. Актуально дальнейшее развитие работ по автоматизированной интеграции таких шаблонов в проектируемые встраиваемые системы. Речь идет о специализированных инструментах, которые должны включаться в индустриальные маршруты проектирования и в соответствующие системы автоматизированного проектирования встраиваемых систем, систем на кристалле и встроенного программного обеспечения.

Литература

1. Leveson N.G., Turner C.S. An investigation of the Therac-25 accidents // Computer. 1993. V. 26. N 7. P. 18–41. doi: 10.1109/MC.1993.274940
2. Ganssle J. A Designer's Guide to Watchdog Timers [Электронный ресурс]. Режим доступа: http://www.digikey.com/en/articles/techzone/2012/may/a-designers-guide-to-watchdog-timers, свободный. Яз. англ. (дата обращения: 01.11.2016).
3. Kobayashi H., Shiraishi K., Tsuchiya H. et. al. Evaluation of LSI soft errors induced by terrestrial cosmic rays and alpha particles [Электронный ресурс]. Режим доступа: http://www.rcnp.osaka-u.ac.jp/~annurep/2001/genkou/sec3/kobayashi.pdf, свободный. Яз. англ. (дата обращения: 01.11.2016).
4. Soft errors in electronic memory - a white paper [Электронный ресурс]. Режим доступа: http://tezzaron.com/media/soft_errors_1_1_secure.pdf, свободный. Яз. англ. (дата обращения: 01.11.2016).
5. Cataldo A. SRAM soft errors cause hard network problems [Электронный ресурс]. Режим доступа: http://www.eetimes.com/document.asp?doc_id=1143781, свободный. Яз. англ. (дата обращения: 01.11.2016).
6. Knight J.C. Safety critical systems: challenges and directions // Proc. 24th Int. Conf. on Software Engineering. 2002. P. 547–550. doi: 10.1145/581339.581406
7. Lee P.A., Anderson T. Fault Tolerance: Principles and Practice. Vienna: Springer Vienna, 1990. 320 p. doi: 10.1007/978-3-7091-8990-0
8. Goloubeva O., Rebaudengo M., Sonza Reorda M., Violante M. Software-Implemented Hardware Fault Tolerance. NY: Springer, 2006. 227 p. doi: 10.1007/0-387-32937-4
9. Koren I., Krishna C.M. Fault Tolerant Systems. San Francisco: Morgan Kaufmann Publ., 2007. 400p.
10. Rajabzadeh A., Miremadi S.G., Mohandespour M. Error detection enhancement in COTS superscalar processors with performance

References

1. Leveson N.G., Turner C.S. An investigation of the Therac-25 accidents. *Computer*, 1993, vol. 26, no. 7, pp. 18–41. doi: 10.1109/MC.1993.274940
2. Ganssle J. *A Designer's Guide to Watchdog Timers*. Available at: http://www.digikey.com/en/articles/techzone/2012/may/a-designers-guide-to-watchdog-timers (accessed: 01.11.2016).
3. Kobayashi H., Shiraishi K., Tsuchiya H. et. al. *Evaluation of LSI soft errors induced by terrestrial cosmic rays and alpha particles*. Available at: http://www.rcnp.osaka-u.ac.jp/~annurep/2001/genkou/sec3/kobayashi.pdf (accessed: 01.11.2016).
4. *Soft errors in electronic memory - a white paper*. Available at: http://tezzaron.com/media/soft_errors_1_1_secure.pdf (accessed: 01.11.2016).
5. Cataldo A. *SRAM soft errors cause hard network problems*. Available at: http://www.eetimes.com/document.asp?doc_id=1143781 (accessed: 01.11.2016).
6. Knight J.C. Safety critical systems: challenges and directions. *Proc. 24th Int. Conf. on Software Engineering*, 2002, pp. 547–550. doi: 10.1145/581339.581406
7. Lee P.A., Anderson T. *Fault Tolerance: Principles and Practice*. Vienna, Springer Vienna, 1990, 320 p. doi: 10.1007/978-3-7091-8990-0
8. Goloubeva O., Rebaudengo M., Sonza Reorda M., Violante M. *Software-Implemented Hardware Fault Tolerance*. NY, Springer, 2006, 227 p. doi: 10.1007/0-387-32937-4
9. Koren I., Krishna C.M. *Fault Tolerant Systems*. San Francisco: Morgan Kaufmann Publ., 2007, 400p.
10. Rajabzadeh A., Miremadi S.G., Mohandespour M. Error detection enhancement in COTS superscalar processors with performance monitoring features. *Journal of Electronic Testing*, 2004, vol. 20, no. 5, pp. 553–567. doi: 10.1023/B:JETT.0000042519.31454.1b
11. Goloubeva O., Rebaudengo M., Sonza Reorda M., Violante M.

monitoring features // Journal of Electronic Testing. 2004. V. 20. N 5. P. 553–567. doi: 10.1023/B:JETT.0000042519.31454.1b
11. Goloubeva O., Rebaudengo M., Sonza Reorda M., Violante M. Soft-error detection using control flow assertions // Proc. 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems. 2003. P. 581–588. doi: 10.1109/DFTVS.2003.1250158
12. Majzik I., Holh W., Pataricza A., Sieh V. Multiprocessor checking using watchdog processors // Computer Systems Science and Engineering. 1996. V. 11. N 5. P. 301–310.
13. Benso A., Di Carlo S., Di Natale G., Prinetto P. A watchdog processor to detect data and control flow errors // Proc. 9th IEEE On-Line Testing Symposium (IOLTS), 2003. P. 144–148. doi: 10.1109/OLT.2003.1214381
14. Bergaoui S., Vanhauwaert P., Leveugle R. IDSM: an improved disjoint signature monitoring scheme for processor behavioral checking // Proc. 15th Latin American Test Workshop - LATW, 2014. P. 1–6. doi: 10.1109/LATW.2014.6841915
15. Rajabzadeh A. A 32-bit COTS-based fault-tolerant embedded system // Proc. 11th IEEE On-Line Testing Symposium (IOLTS), 2005. P. 205v206. doi: 10.1109/IOLTS.2005.5
16. Djambazova E., Djambazov K. Processor control-flow error-detection techniques-model and evaluation tool // Cybernetics and Information Technologies. 2001. V. 1. N 2. P. 3–18.
17. Ganssle J. Great Watchdog Timers for Embedded Systems [Электронный ресурс]. Режим доступа: http://www.ganssle.com/watchdogs.htm, свободный. Яз. англ. (дата обращения: 01.11.2016).
18. Тимофеев В. 5.4 Сторожевой таймер [Электронный ресурс]. Режим доступа: http://www.pic24.ru/doku.php/osa/articles/wdt, свободный. Яз. рус. (дата обращения: 01.11.2016).
19. Japenga B. Guidelines for Creating Robust Embedded Systems. Part 7 – Creating Robust Watchdog Timers. Great Watchdog Timers [Электронный ресурс]. Режим доступа: http://www.microtoolsinc.com/RobustGuidelines7.pdf, свободный. Яз. англ. (дата обращения: 01.11.2016).
20. Чакраварти С., Томар Р., Арора М. Сторожевой таймер для отказоустойчивых систем // Электронные компоненты. 2008. № 12. С. 15–20.
21. Lamberson J. Single and Multistage Watchdog Timers [Электронный ресурс]. Режим доступа: http://www.sensoray.com/downloads/appnote_826_watchdog_1.0.0.pdf, свободный. Яз. англ. (дата обращения: 01.11.2016).
22. Gehlot P. Watchdog timer for robust embedded systems // Electronics for you. 2015. N 2. P. 60–62.
23. Murphy N. Watchdog timers // Embedded Systems Programming. 2000. V. 13. N 12. P. 112–124.
24. Pohronska M., Krajcovic T. Fault-tolerant embedded systems with multiple FPGA implemented watchdogs [Электронный ресурс]. Режим доступа: https://pdfs.semanticscholar.org/4577/acea8ac928150d119053a43d2ab9ff207cd6.pdf, свободный. Яз. англ. (дата обращения: 01.11.2016).
25. El-Attar A.M., Fahmy G. An improved watchdog timer to enhance imaging system reliability in the presence of soft errors // Proc. IEEE Int. Symposium on Signal Processing and Information Technology, 2007. P. 1100–1104. doi: 10.1109/ISSPIT.2007.4458184
26. Pont M.J., Ong R.H.L. Using watchdog timers to improve the reliability of single-processor embedded systems: Seven new patterns and a case study // Proc. First Nordic Conference on Pattern Languages of Programs. 2002. P. 159–200.
27. Cunha J.C. Reset-driven fault tolerance // Lecture Notes in Computer Science. 2002. V. 2485. P. 102–120. doi: 10.1007/3-540-36080-8_13
28. Schlaepfer E. Comparison of Internal and External Watchdog Timers [Электронный ресурс]. Режим доступа: https://www.maximintegrated.com/en/app-notes/index.mvp/id/4229, свободный. Яз. англ. (дата обращения: 01.11.2016).
29. Du B., Reorda S., Sterpone L., Parra L. et al. Online test of control flow errors: anew debug interface-based approach // IEEE Transactions on Computers. 2016. V. 65. N 6. P. 1846–1855. doi: 10.1109/TC.2015.2456014
30. Boroomandnezhad T., Azgomi M.A. An efficient control-flow checking technique for the detection of soft-errors in embedded software // Computers & Electrical Engineering. 2013. V. 39. N 4. P. 1320–1332. doi: 10.1016/j.compeleceng.2013.03.015

Soft-error detection using control flow assertions. *Proc. 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003, pp. 581–588. doi: 10.1109/DFTVS.2003.1250158
12. Majzik I., Holh W., Pataricza A., Sieh V. Multiprocessor checking using watchdog processors. *Computer Systems Science and Engineering*, 1996, vol. 11, no. 5, pp. 301–310.
13. Benso A., Di Carlo S., Di Natale G., Prinetto P. A watchdog processor to detect data and control flow errors. *Proc. 9th IEEE On-Line Testing Symposium, IOLTS*, 2003, pp. 144–148. doi: 10.1109/OLT.2003.1214381
14. Bergaoui S., Vanhauwaert P., Leveugle R. IDSM: an improved disjoint signature monitoring scheme for processor behavioral checking. *Proc. 15th Latin American Test Workshop - LATW*, 2014, pp. 1–6. doi: 10.1109/LATW.2014.6841915
15. Rajabzadeh A. A 32-bit COTS-based fault-tolerant embedded system. *Proc. 11th IEEE On-Line Testing Symposium, IOLTS*, 2005, pp. 205v206. doi: 10.1109/IOLTS.2005.5
16. Djambazova E., Djambazov K. Processor control-flow error-detection techniques-model and evaluation tool. *Cybernetics and Information Technologies*, 2001, vol. 1, no. 2, pp. 3–18.
17. Ganssle J. *Great Watchdog Timers for Embedded Systems*. Available at: http://www.ganssle.com/watchdogs.htm (accessed: 01.11.2016).
18. Timofeev V. 5.4 *Watchdog Timer*. Available at: http://www.pic24.ru/doku.php/osa/articles/wdt (accessed: 01.11.2016).
19. Japenga B. *Guidelines for Creating Robust Embedded Systems. Part 7 – Creating Robust Watchdog Timers. Great Watchdog Timers*. Available at: http://www.microtoolsinc.com/RobustGuidelines7.pdf (accessed: 01.11.2016).
20. Chakravarty S., Tomar R., Arora M. Watchdog Timer for Fault-Tolerant Systems. *Elektronnye Komponenty*, 2008, no. 12, pp. 15–20.
21. Lamberson J. *Single and Multistage Watchdog Timers*. Available at: http://www.sensoray.com/downloads/appnote_826_watchdog_1.0.0.pdf (accessed: 01.11.2016).
22. Gehlot P. Watchdog Timer for Robust Embedded Systems. *Electronics for you*, 2015, no. 2, pp. 60–62.
23. Murphy N. Watchdog timers. *Embedded Systems Programming*, 2000, vol. 13, no. 12, pp. 112–124.
24. Pohronska M., Krajcovic T. *Fault-tolerant embedded systems with multiple FPGA implemented watchdogs*. Available at: https://pdfs.semanticscholar.org/4577/acea8ac928150d119053a43d2ab9ff207cd6.pdf, свободный. Яз. англ. (accessed: 01.11.2016).
25. El-Attar A.M., Fahmy G. An improved watchdog timer to enhance imaging system reliability in the presence of soft errors. *Proc. IEEE Int. Symposium on Signal Processing and Information Technology*, 2007, pp. 1100–1104. doi: 10.1109/ISSPIT.2007.4458184
26. Pont M.J., Ong R.H.L. Using watchdog timers to improve the reliability of single-processor embedded systems: Seven new patterns and a case study. *Proc. First Nordic Conference on Pattern Languages of Programs*, 2002, pp. 159–200.
27. Cunha J.C. Reset-driven fault tolerance. *Lecture Notes in Computer Science*, 2002, vol. 2485, pp. 102–120. doi: 10.1007/3-540-36080-8_13
28. Schlaepfer E. *Comparison of Internal and External Watchdog Timers*. Available at: https://www.maximintegrated.com/en/app-notes/index.mvp/id/4229 (accessed: 01.11.2016).
29. Du B., Reorda S., Sterpone L., Parra L. et al. Online test of control flow errors: anew debug interface-based approach. *IEEE Transactions on Computers*, 2016, vol. 65, no. 6, pp. 1846–1855. doi: 10.1109/TC.2015.2456014
30. Boroomandnezhad T., Azgomi M.A. An efficient control-flow checking technique for the detection of soft-errors in embedded software. *Computers & Electrical Engineering*, 2013, vol. 39, no. 4, pp. 1320–1332. doi: 10.1016/j.compeleceng.2013.03.015
31. Abdi A., Asghari S.A., Pourmozaffari S., Taheri H., Pedram H. An effective software implemented data error detection method in real time systems. *Advances in Computer Science, Engineering & Applications*, 2012, vol. 106, pp. 919–926. doi: 10.1007/978-3-642-30157-5_91
32. Avizienis A. et al. Basic concepts and taxonomy of dependable

31. Abdi A., Asghari S.A., Pourmozaffari S., Taheri H., Pedram H. An effective software implemented data error detection method in real time systems // Advances in Computer Science, Engineering & Applications. 2012. V. 106. P. 919–926. doi: 10.1007/978-3-642-30157-5_91
32. Avizienis A. et al. Basic concepts and taxonomy of dependable and secure computing // IEEE Transactions on Dependable and Secure Computing. 2004. V. 1. N 1. P. 11–33. doi: 10.1109/TDSC.2004.2
33. Hooman J., Hendriks T. Model-based run-time error detection // Lecture Notes in Computer Science. 2007. V. 5002. P. 225–236. doi: 10.1007/978-3-540-69073-3_24
34. Geffroy J.C., Motet G. Design of Dependable Computing Systems. Dordrecht: Springer, 2002. 672 p. doi: 10.1007/978-94-015-9884-2
35. Пенской А.В. Архитектурное документирование встроенных систем с многоуровневой конфигурацией // Изв. вузов. Приборостроение. 2015. V. 57. N 7. P. 527–532.
36. Platunov A., Kluchev A., Penskoi A. HLD methodology: the role of architectural abstractions in embedded systems design // Proc. 14th Geo Conference on Informatics, Geoinformatics and Remote Sensing. 2014. P. 209–218.
37. Teich J. Hardware/software codesign: the past, the present, and predicting the future // Proc. IEEE. 2012. V. 100. P. 1411–1430. doi: 10.1109/JPROC.2011.2182009
38. Platunov A., Nickolaenkov A. Aspects in the design of software-intensive systems // Mediterranean Conference on Embedded Computing (MECO). 2012. P. 84–87.
39. Jozwiak L., Nedjah N. Modern architectures for embedded reconfigurable systems - A survey // J. of Circuits, Systems, and Computers. 2009. V. 18. N 2. P. 209–254. doi: 10.1142/s0218126609005034
40. Jozwiak L., Nedjah N., Figueroa M. Modern development methods and tools for embedded reconfigurable systems: a survey // Integration, the VLSI j. 2010. V. 43. N 1. P. 1–33. doi: 10.1016/j.vlsi.2009.06.002
41. Платунов А.Е. Реконфигурируемые встраиваемые системы и системы на кристалле // Известия высших учебных заведений. Приборостроение 2014. Т. 57. № 4. С. 49–52.
42. Hartenstein R. The relevance of reconfigurable computing / In: J.M.P. Cardoso, M. Hübner (eds.), Reconfigurable Computing: From FPGAs to Hardware/Software Codesign. Springer, 2011. 296 p.

and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 2004, vol. 1, no. 1, pp. 11–33. doi: 10.1109/TDSC.2004.2
33. Hooman J., Hendriks T. Model-based run-time error detection. *Lecture Notes in Computer Science*, 2007, vol. 5002, pp. 225–236. doi: 10.1007/978-3-540-69073-3_24
34. Geffroy J.C., Motet G. *Design of Dependable Computing Systems*. Dordrecht: Springer, 2002, 672 p. doi: 10.1007/978-94-015-9884-2
35. Penskoi A. V. Architectural specification of embedded systems with multi-level configuration. *Izvestiya Vysshikh Uchebnykh Zavedeniy. Priborostroenie*, 2015, vol. 58, no. 7, pp. 527–532 (in Russian).
36. Platunov A., Kluchev A., Penskoi A. HLD methodology: the role of architectural abstractions in embedded systems design. *Proc. 14th Geo Conference on Informatics, Geoinformatics and Remote Sensing*, 2014, pp. 209–218.
37. Teich J. Hardware/software codesign: the past, the present, and predicting the future. *Proc. IEEE*, 2012, vol. 100, pp. 1411–1430. doi: 10.1109/JPROC.2011.2182009
38. Platunov A., Nickolaenkov A. Aspects in the design of software-intensive systems. *Mediterranean Conference on Embedded Computing, MECO*, 2012, pp. 84–87.
39. Jozwiak L., Nedjah N. Modern architectures for embedded reconfigurable systems - a survey. *J. of Circuits, Systems, and Computers*, 2009, vol. 18, no. 2, pp. 209–254. doi: 10.1142/s0218126609005034
40. Jozwiak L., Nedjah N., Figueroa M. Modern development methods and tools for embedded reconfigurable systems: a survey. *Integration, the VLSI j*, 2010, vol. 43, no. 1, pp. 1–33. doi: 10.1016/j.vlsi.2009.06.002
41. Platunov A.E. Reconfigurable embedded systems and system-on-chip. *Izvestiya Vysshikh Uchebnykh Zavedeniy. Priborostroenie,* 2014, vol. 57, no. 4, pp. 49–52. (in Russian).
42. Hartenstein R. The relevance of reconfigurable computing. In: J.M.P. Cardoso, M. Hübner (eds.), *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*. Springer, 2011, 296 p.

Авторы

Платунов Алексей Евгеньевич – доктор технических наук, профессор, профессор, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, aeplatunov@gmail.com

Стерхов Александр Сергеевич – студент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, ink-sa@mail.ru

Authors

Alexey E. Platunov – D.Sc., Full Professor, ITMO University, Saint Petersburg, 197101, Russian Federation, aeplatunov@gmail.com

Alexander S. Sterkhov – student, ITMO University, Saint Petersburg, 197101, Russian Federation, ink-sa@mail.ru