УДК 004.337, 004.052.42

Оснащение систем на кристалле средствами эмуляции сбоев в памяти

О.В. Мамутова, О.В. Ненашев, А.С. Филиппов

Санкт-Петербургский государственный политехнический университет

System on Chip Instrumentation for Emulated Memory Fault Injection

O.V. Mamoutova, O.V. Nenashev, A.S. Filippov

Saint-Petersburg State Polytechnic University

Представлен новый метод автоматизированного оснащения системы на кристалле агентами внесения неисправностей типа «одиночный сбой» для блоков встроенной памяти. Метод основан на использовании гибридных моделей устройств и может быть применен на этапе прототипирования для любых маршрутов проектирования и форматов исходных описаний. Тестирование на системе с процессором OpenRISC1200 показало высокое быстродействие и существенную экономию накладных расходов на дополнительные аппаратные ресурсы кристалла при использовании данного метода.

Ключевые слова: автоматизация проектирования; встроенная память; внесение неисправностей; реинжиниринг; одиночный сбой.

The emulation of fault injection at the prototyping stage for Systems-on-Chip (SoC) is an efficient method for evaluating the system reaction to SEU effects. A new method of the automated SoC instrumentation for emuleted fault injection into embedded memory blocks has been presented. This approach has been successfully tested for the system with the OpenRISC1200 processor and has demonstrated the low hardware and engineering time overheads.

Keywords: automated instrumentation; embedded memory; fault injection; hardware reengineering; SEU

Для современных микросхем одиночные сбои памяти являются одной из основных угроз работоспособности при воздействии излучения в космосе [1]. Сбой инвертирует значения битов в памяти. Исправление этой ошибки происходит при записи нового значения. Однако, если до момента записи операционный узел системы прочитает неверное значение, может произойти отказ всей системы. Поэтому актуальна задача борьбы с последствиями одиночных сбоев.

Решение проблемы одиночных сбоев при проектировании надежной системы на кристалле (далее – система) заключается, с одной стороны, в синтезе средств по борьбе со сбоями, а с другой – в анализе воздействия сбоев на систему. В ходе синтеза используются известные методы: схемотехнически усиленные ячейки памяти, структурное резервирование, помехоустойчивое кодирование, периодическое просеивание массивов памяти, по-

вторное исполнение программ. Для анализа реакции системы на ошибку применяются методы, основанные на внесении в память системы различного рода неисправностей [2]. Внесение неисправностей позволяет выявить «узкие места» в надежности системы, классифицировать возникающие эффекты и проверять средства борьбы с неисправностями.

В настоящей работе представлен новый метод автоматизированного оснащения систем средствами внесения неисправностей типа «одиночный сбой» в блоки встроенной памяти. Подход применим на этапе прототипирования систем с помощью программируемых логических интегральных схем (ПЛИС) и позволяет проводить верификацию надежности систем с памятью и процессорными ядрами. Практическим результатом работы является прототип программы автоматизированного оснащения систем средствами внесения неисправностей в память на базе программируемого средства реинжиниринга устройств PHRT (Programmable Hardware Reengineering Toolkit).

Анализ существующих подходов при внесении сбоев в память. Время жизни ошибки в памяти сопоставимо с временем исполнения программы, поэтому анализ влияния сбоя на систему следует проводить в масштабе времени исполнения вычислительной нагрузки [3]. По этой причине для внесения сбоев в блоки памяти эффективна эмуляция внесения неисправностей с использованием ПЛИС. Ошибка может быть введена в систему двумя способами: с помощью частичной реконфигурации ПЛИС или дополнительными средствами, которыми оснащается исходная система. Метод оснащения системы в отличие от частичной реконфигурации не ограничен определенной аппаратной платформой и применим для произвольных ПЛИС.

Оснащение заключается в добавлении к исследуемой системе агентов внесения неисправностей в виде мутантов или саботажников. Мутант – это модифицированный узел исходной системы, саботажник – новый узел, подключаемый между узлами исходной системы. Большинство существующих подходов оснащения позволяет вносить сбои только в отдельные триггеры и лишь некоторые – в блоки встроенной памяти [4]. Основная проблема подходов состоит в том, что в исходной системе обычно отсутствует доступ к отдельным битам встроенной памяти помимо ее основного интерфейса. Предлагаемый подход основан на использовании подключаемых к блокам памяти саботажников под управлением встроенного процессора, что позволяет минимизировать дополнительные аппаратные затраты, [5].

Идея использования имеющегося в системе процессорного ядра для служебных целей не нова и часто применяется для тестирования. Существуют подходы, называемые встроенным тестированием под управлением процессора [6], или встроенной программной самодиагностикой [7]. Однако для задач внесения неисправностей не используются мощности имеющихся в системе процессорных ядер, так как для этого требуется решить задачу синхронизации тестов и основной функциональности системы. Один из редких примеров – частичная реконфигурация под управлением процессора, описанная в работе [8]. Предлагаемый подход реализует иной вариант: процессорное ядро используется для управления распределенной системой саботажников.

Еще одна сложность оснащения памяти обусловлена необходимостью изменения исходного кода на языке описания аппаратуры (Hardware description language – HDL). Очевидно, что оснащение вручную занимает много времени и ведет к возможным ошибкам, поэтому требуется автоматизация процесса оснащения. Поэтому предлагаемый подход реализует автоматизированное оснащение систем с использованием гибридной модели [9] на базе исходного описания и низкоуровневого структурного описания, получаемого при синтезе и содержащего синтезированные блоки памяти.

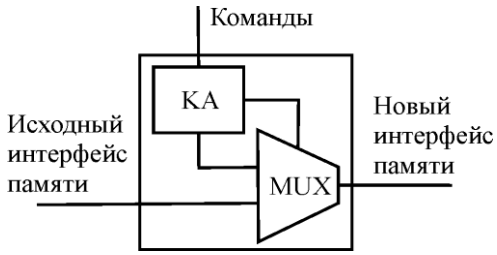

Рис.1. Саботажник для блока памяти

Саботажники, управляемые встроенным процессором. При оснащении памяти в разрыв сигналов ее основного интерфейса подключается саботажник (рис.1), который содержит мультиплексор (MUX) и систему управления в виде конечного автомата (КА). Мультиплексор осуществляет переключение между основным интерфейсом памяти и выходами конечного автомата, управляющего мультиплексированием, и по команде внесения сбоя осуществляет процедуру чтение–изменение–запись для инверсии битов памяти. Саботажники управляются одним из процессорных ядер исследуемой системы, что позволяет реализовать необходимый алгоритм внесения неисправностей как пользовательскую программу для процессора.

Методика оснащения системы средствами внесения ошибок в память. На рис.2 показано оснащение системы с блоками встроенной памяти: внутри процессора, на системной шине и внутри некоторого периферийного блока. Для получения команд от процессора саботажники подключены напрямую к системной шине или через промежуточный адаптер, дающий доступ ко всем саботажникам.

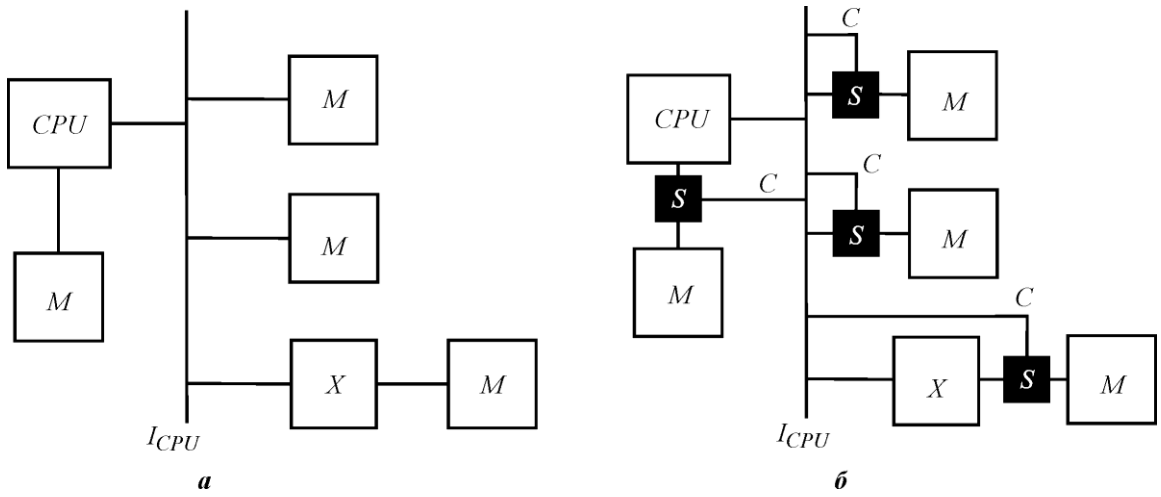

Рис.2. Пример оснащения системы с целью внесения неисправностей в память:
а – исходная система; *б* – модифицированная система с саботажниками

Определим инфраструктуру внесения сбоев в память следующим множеством:

$$FI_M = (M, S, I_{CPU}, C, A_{SW}),$$

где M – множество оснащаемых блоков памяти; S – множество саботажников, подключенных к блокам памяти M; I_{CPU} – интерфейс процессорного ядра; C – множество соединений между саботажниками S и интерфейсом I_{CPU}; A_{SW} – программные реализации алгоритмов проведения экспериментов.

При встраивании средств внесения неисправностей в систему предлагается следующий порядок действий:

1) выбор в иерархии проекта части системы, подлежащей исследованию;
2) поиск блоков памяти M в выбранной части системы;
3) оснащение выбранных блоков памяти M саботажниками S;
4) подключение саботажников S к интерфейсу процессора I_{CPU} (соединения C).

В большинстве существующих методов оснащения модифицируются исходные HDL-коды, но такой подход имеет существенные ограничения для встроенной блочной памяти. Первое ограничение – в том, что высокоуровневое описание вследствие оптимизаций при синтезе может не соответствовать конечным результатам. Это затрудняет поиск блоков памяти. Второе ограничение обусловлено сложностью подключения сигналов от саботажников к процессору сквозь уровни иерархии проекта, так как требуется модификация всех элементов иерархии устройства. Третье ограничение состоит в том, что существует множество различных языков описания аппаратуры. Это усложняет создание универсального средства оснащения.

Перечисленные проблемы предлагается решать за счет использования одновременно исходного высокоуровневого описания системы и результатов его синтеза. При этом иерархия проекта из исходного описания применяется для разделения системы на интересующие исследователя части, информация об экземплярах синтезированной памяти позволяет находить блоки памяти, а низкоуровневое структурное описание предоставляет плоское пространство сигналов для организации новых связей между блоками системы.

За основу взята гибридная модель представления устройств из средства PHRT [9], которая формируется на основе низкоуровневых структурных описаний и поддерживает элементы высокоуровневых HDL. Это позволяет представить в одной модели исходную систему, аппаратные компоненты инфраструктуры внесения сбоев FI_M и библиотеки встраиваемых саботажников. При помощи статической параметризации и условной генерации реализуются саботажники S, описанные во внешних библиотеках. Гибридная модель совместима с большинством форматов нетлистов и HDL (EDIF, VHDL, Verilog и SystemVerilog) и не зависит от исходного описания системы.

Подход к автоматизации реинжиниринга на базе PHRT. Автоматизация предлагаемого подхода выполнена на базе средства PHRT, использующего одноименную модель. Приложение PHRT является расширяемым средством автоматизации проектирования (САПР) для автоматизации частных задач анализа и трансформации устройств. PHRT применяет гибридную модель устройств и может быть интегрировано с прочими САПР для использования их возможностей в маршрутах проектирования. Схема работы PHRT приведена на рис.3.

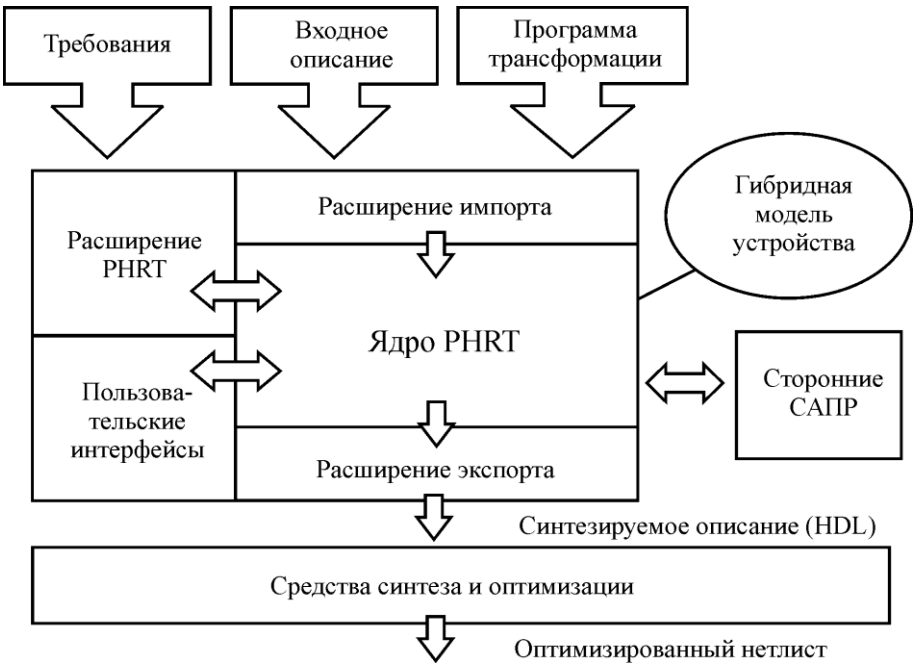

Рис.3. Общая схема реинжиниринга устройств с использованием PHRT

Ядро средства предоставляет набор базовых функций для работы с гибридной моделью; пользовательские алгоритмы реинжиниринга реализуются в PHRT в виде расширений. Существующие расширения позволяют решить следующие задачи, возникающие при встраивании саботажников в исследуемую систему:

- импорт/экспорт данных в формате EDIF, VHDL, XML;
- базовые операции над гибридной моделью (добавление и удаление элементов, переименование, доступ к свойствам и пр.);
- соединение компонентов на разных уровнях иерархии;
- анализ (оценка площади кристалла, временной анализ и пр.);
- интеграция с САПР Quartus II для прототипирования на ПЛИС (импорт/экспорт проектов, частичный синтез компонентов устройства).

Удобство внутреннего представления и имеющиеся базовые функции, независимость от формата исходного описания устройства, а также функциональная гибкость за счет возможности расширения обусловили выбор PHRT в качестве базы для реализации поставленной в работе задачи. Данное средство не предлагает готового решения для оснащения памяти средствами внесения неисправностей, поэтому потребовалась разработка соответствующего функционального расширения.

Реализация алгоритма оснащения памяти системы в PHRT. Для автоматизации алгоритма оснащения разработано функциональное расширение memfault_inject. Методика использования расширения, где все шаги выполняются автоматически под управлением сценариев на языке TCL, приведена на рис.4. Подобные сценарии могут быть реализованы для частных конфигураций устройств и памяти.

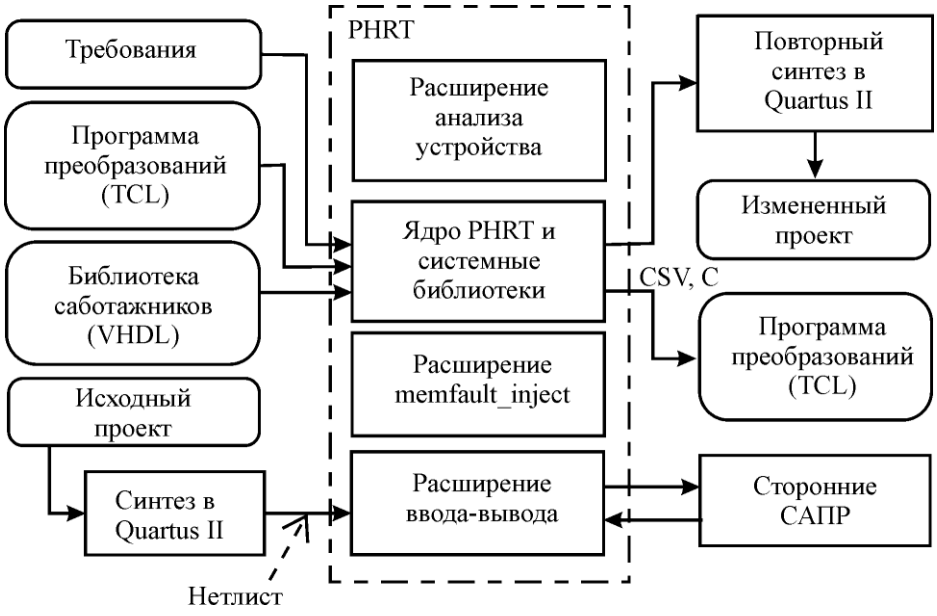

Рис.4. Процесс преобразования устройства при встраивании саботажников

Процесс начинается с синтеза исходной системы в Quartus II и получения низкоуровневого структурного описания (нетлиста). Затем осуществляется импорт этого описания в PHRT и построение гибридной модели на базе исходного высокоуровневого описания и полученного нетлиста. Далее программа проводит поиск блоков памяти по имени (“altsyncram”) и добавляет к ним саботажники S, после чего все саботажники подключаются к интерфейсу процессора I_{CPU}. Приведем часть сценария, реализующую алгоритм посредством вызова операций из расширений PHRT:

```
# Load extensions and import device
phrt::system::load_extensions \
      {quartus vhdl_netlist memfault_inject }
phrt::vhdl_netlist::import openRISC1200.vhd /test/

# Find memory by mask and instrument it
set mem_list \
      [phrt:find_refs /test/or1200_top -R ALTSYNCRAM]
phrt::memfault_inject:instrument $mem_list

# Output modified netlist
phrt::vhdl_netlist::export /test out.vhd
```

Для выполнения модификации системы в сценарии вызывается расширение memfault_inject, которое оперирует библиотечным компонентом саботажника и включает в себя функции добавления саботажника к заданному интерфейсу памяти. Так как саботажники описаны на VHDL и имеют статические параметры (generic), PHRT осуществляет их синтез посредством вызова Quartus II для формирования структурной модели. Далее происходит оснащение блоков памяти M и подключение саботажников S к процессорному ядру (см. рис. 2), после чего формируется список адресов саботажников в пространстве I_{CPU}, используемый при реализации алгоритмов экспериментов A_{SW}.

Оценка эффективности реализованного метода. Для демонстрации применимости предложенного метода проведено тестирование и оценка его эффективности с использованием ПЛИС CycloneII. В качестве тестового устройства использована система на базе программного процессора с открытым исходным кодом OpenRISC1200. Для тестовой системы проведено несколько вариантов оснащения: оснащение одного блока памяти вручную, автоматизированное оснащение того же блока памяти и автоматизированное оснащение всех блоков, которые при синтезе были заменены на встроенную в ПЛИС синхронную память (ALTSYNCRAM). Исходное описание процессорного ядра содержит 13 таких блоков. (Исходные коды аппаратных блоков, использованных для оснащения, доступны в репозитории github.com/Mamoutova/memory_fault_injection.)

Автоматизированное оснащение проводилось средствами PHRT. В таблице приведены ключевые характеристики системы до и после оснащения (максимальная частота синхронизации F_{max} и затраты аппаратных ресурсов ПЛИС). Также рассмотрена система, прошедшая ввод-вывод в PHRT без дополнительных модификаций. Синтез проводился в САПР Quartus II в одинаковых условиях: на одном компьютере, в одной версии САПР и при одинаковых настройках синтеза и оптимизации.

Сравнение результатов синтеза

Синтезированное устройство	F_{max}, МГц	LCELL	Регистры	Встроенная память, бит
Исходная система	39,31	8055	3105	53888
Немодифицированная система после PHRT I/O	37,18	8247	3105	53888
Один блок памяти, ручная модификация	39,31	8215	3142	53888
Один блок памяти, автоматизированное оснащение	37,12	8310	3187	53888
Полностью оснащенная система, 13 блоков памяти	36,53	8615	3528	53888

Из таблицы видно, что модификации синтезируемой системы ведут к снижению максимальной тактовой частоты, так как оснащаемая память находится на критическом пути тактирования и использует ту же частоту, что и процессор. Для приведенного примера в худшем случае потеря составляет около 7%, что существенно лучше иных методов внесения неисправностей. Таким образом, не теряется основное преимущество эмуляции при внесении неисправностей, а именно скорость проведения экспериментов и возможность проведения длительных тестов памяти.

Анализ аппаратных затрат показал, что ни один из представленных вариантов не требует дополнительных ресурсов встроенной памяти в ПЛИС (блоки M4K). В то же время реализация саботажников требует дополнительных триггеров и комбинаторной логики на шинах адреса и данных, но за счет отсутствия систем генерации тестовых векторов и периферийных блоков достигается существенная экономия по сравнению с другими подходами. Разница между ручным и автоматизированным описаниями объясняется использованием при конечном синтезе разных исходных описаний: вручную модифицированного Verilog-кода и сгенерированного PHRT нетлиста.

Для проверки работоспособности метода использованы простые тесты памяти, состоящие из последовательных обращений чтение–модификация–проверка для всех битов оснащенных блоков памяти. Поскольку управление сбоями осуществляется процессорным ядром, исполняющим основную программу, то конфликты при доступе в память невозможны. В иных случаях конфликты должны разрешаться саботажником.

Запуск ряда тестовых программ из набора EEMBC показал, что автоматизированное оснащение системы не нарушает основных функций процессора. Задача полной верификации модифицированной системы в работе не решалась, но она может быть решена сторонними САПР или с помощью дополнительных расширений PHRT.

Таким образом, предложенный метод автоматизированного оснащения систем на кристалле является эффективным способом подготовки системы к экспериментам по внесению неисправностей. Полученные результаты подтверждают применимость предложенного метода для оценки устойчивости системы к сбоям в памяти для реальных процессорных систем. Апробация на реальных устройствах показала, что метод обеспечивает высокую эффективность по аппаратным ресурсам и скорости проведения экспериментов.

Направление дальнейшей работы – исследование других архитектур процессорных ядер со встроенной помехоустойчивой памятью с целью сравнения различных архитектур надежной памяти в условиях реальной вычислительной нагрузки и разных шаблонов сбоев. При этом ставится задача разработки новых тестовых программ для оценки надежности, поддержанных внесением неисправностей. Такие тестовые программы применимы для проведения исследований с целями сравнения различных архитектур систем на кристалле со встроенной памятью и выбора варианта с лучшим соотношением надежность–производительность.

Литература

1. ***Максименко С.Л., Мелехин В.Ф., Филиппов А.С.*** Анализ проблемы построения радиационностойких информационно-управляющих систем // Информационно-управляющие системы. – 2012. – № 2. – С. 18–25.

2. Fault injection and dependability evaluation of fault-tolerant systems / ***J. Arlat et al.*** // IEEE Transactions on Computers. – 1993. – Vol. 42. – P.913–923.

3. ***Benso A., DiCarlo S.*** The art of fault injection // J. of Control Engineering and Applied Informatics. – 2011. – Vol. 13. – P.9–18.

4. A new approach to accelerate SEU sensitivity evaluation in circuits with embedded memories / ***M. Portela-Garcia et al.*** // Proc. SPIE VLSI Circuits and Systems IV. – 2009. – Vol. 7363.

5. ***Мамутова О.В.*** Использование встроенного процессора для управления эмуляцией внесения неисправностей типа «сбой» в блоки памяти // Университетский научный журнал. – 2013. – Вып. 5. – С. 185–193.

6. ***Kenney J.*** Using a processor-driven test bench for functional verification of embedded SoCs // All configurable systems development articles. – Oct. – 2006.

7. Microprocessor software-based self-testing / ***M. Psarakis et al.*** // Design & Test of Computers. – 2010. – Vol. 27. – P. 4–19.

8. ***Dutton B.F., Ali M., Stroud C.E., Sunwoo J.*** Embedded processor based fault injection and SEU emulation for FPGAs // Proc. Embedded Systems and Applications. – 2009. – P. 183–189.

9. ***Ненашев О.В.*** Расширяемый инструментарий для автоматизации реинжиниринга цифровых систем на кристалле // Университетский научный журнал. – 2013. – Вып. 5. – С. 194–203.

Статья поступила
22 января 2014 г.

Мамутова Ольга Вячеславовна – старший преподаватель кафедры компьютерных систем и программных технологий (КСПТ) Санкт-Петербургского государственного политехнического университета. *Область научных интересов:* встраиваемые системы, надежность вычислительных систем для космического применения. **E-mail: mamoutova@kspt.icc.spbstu.ru**

Ненашев Олег Вячеславович – аспирант кафедры КСПТ Санкт-Петербургского государственного политехнического университета. *Область научных интересов:* встраиваемые системы, автоматизация проектирования и непрерывная интеграция.

Филиппов Алексей Семенович – кандидат технических наук, доцент кафедры КСПТ Санкт-Петербургского государственного политехнического университета. *Область научных интересов:* программируемая логика, автоматизированное проектирование встраиваемых систем.