СИСТЕМЫ И СРЕДСТВА ИНФОРМАТИКИ

Выпуск 22 № 2 Год 2012

СОДЕРЖАНИЕ

СИСТЕМЫ И СРЕДСТВА ИНФОРМАТИКИ, 2012. Т. 22. № 2. С. 29–37

АВТОМАТИЗАЦИЯ ОТЛАДКИ ПРОГРАММ ДЛЯ РЕКУРРЕНТНОГО ОПЕРАЦИОННОГО УСТРОЙСТВА*

***Р. А. Зеленов*[1], *А. А. Прокофьев*[2], *В. Н. Волчек*[3]**

Аннотация: Показаны проблемы, встающие перед разработчиком низкоуровневых программ, и способы их решения на примере среды проектирования программного обеспечения для рекуррентного операционного устройства (РОУ). Описывается реализация средств уменьшения числа ошибок на этапе программирования, а также инструмента автоматической локализации ошибок, приводящих к генерации исключений в аппаратуре.

Ключевые слова: интегрированная среда разработки; автоматизация отладки; потоковая архитектура; рекуррентность

1 Введение

В области разработки низкоуровневых программ, предназначенных для выполнения на микропроцессорах, средства поддержки программирования представлены различными средами разработки либо для ассемблерных языков, либо для языков программирования микроконтроллеров. Данные программные продукты включают в себя оболочку, предоставляющую удобный интерфейс пользователя, компиляторы и отладочные средства. Инструменты отладки, как правило, предоставляют следующую функциональность: пошаговый просмотр, отображение информации о текущих значениях переменных и трассировка всего хода вычислений.

Каких-либо автоматизированных средств локализации ошибок, приводящих к возникновению исключений или прерываний в процессоре, в широком применении не используется. Следовательно, процесс отладки ложится полностью на плечи программиста и эффективность отладки зависит от его опыта и квалификации.

В Институте проблем информатики Российской академии наук ведется разработка многоядерной потоковой рекуррентной вычислительной системы (МПРВС)

*Работа выполнена при частичной финансовой поддержке по Программе фундаментальных исследований ОНИТ РАН на 2011 г., проект 1.5.

[1]Институт проблем информатики Российской академии наук, graf.developer@gmail.com

[2]Институт проблем информатики Российской академии наук, a.a.prokofyev@mail.ru

[3]Институт проблем информатики Российской академии наук, v_volchek@inbox.ru

с нетрадиционной архитектурой [1]. Многоядерная потоковая рекуррентная вычислительная система реализована в виде двухуровневой системы: управляющий уровень, базирующийся на фон-неймановских принципах, и операционный уровень, реализованный в виде РОУ, базирующийся на потоковых (dataflow) принципах. Именно для РОУ требуется разработка низкоуровневых программ-капсул [2], которые представляют собой наборы элементов самодостаточных данных (ЭСД, операндов). Элементы самодостаточных данных — это совокупность данных и инструкций для их обработки.

Для целей программирования капсул была разработана система капсульного программирования и отладки (СКАТ) [3], которую можно отнести к интегрированным средам разработки (IDE). Возможности СКАТ соответствуют функциональности большинства IDE, существующих в мире, т. е. она содержит в себе средства создания и отладки программ, в данном случае капсул.

2 Возможности и недостатки системы капсульного программирования и отладки

В настоящее время СКАТ имеет следующие функциональные возможности:

(1) создание капсул с помощью визуального конструктора капсул (ВКК);

(2) отладка капсул:

- просмотр хода вычислительного процесса:
  - файл-отчет выполнения моделирования;
  - временные диаграммы;
  - пошаговый просмотр всех ключевых регистров и памяти;
- пошаговая отладка — изменение значений определенных регистров в режиме пошагового просмотра;

(3) подготовка файлов для выполнения моделирования вычислительного процесса:

- генерация служебных файлов, необходимых для запуска моделирования в симуляторе ModelSim;
- создание, редактирование, добавление и удаление файлов VHDL-модели РОУ;
- формирование содержимого буферной памяти (БП) [4];

(4) выполнение моделирования в ModelSim [4].

Подробное описание указанных возможностей представлено в [3].

Система капсульного программирования и отладки уникальна тем, что при ее разработке закладывались принципы, предоставляющие программисту возможность реконфигурации данной системы при изменениях в спецификации РОУ.

Это выражается в возможности быстрого изменения формата операндов и настройки процедуры отладки в соответствии с тем, какими аппаратными ресурсами обладает РОУ.

Однако СКАТ не лишена недостатков, которые выражаются в следующем:

– на этапе формирования капсулы отсутствуют средства, снижающие вероятность появления ошибок, связанных с неверной семантикой;

– отсутствует инструментарий, позволяющий автоматизировать процесс локализации ошибок, приводящих к появлению исключений в аппаратуре, и тем самым сокращать время отладки капсул.

Текущий этап разработки МПРВС требует модернизации СКАТ таким образом, чтобы вышеуказанные недостатки были устранены. Основная цель развития функциональных возможностей системы заключается в снижении вероятности появления ошибок в капсулах, но если ошибки все же возникли во время выполнения, то необходимо ускорить процесс их исправления путем внедрения автоматизированного средства локализации ошибок. В данной статье будут рассмотрены проблемы, связанные с реализацией данной модернизации, и способы их решения.

3 Снижение вероятности появления ошибок в капсулах

Ошибки во время выполнения программ могут возникать по нескольким причинам [5]:

(1) неверный синтаксис или семантика языка;

(2) неверная реализация алгоритма программистом;

(3) ошибки в инструментах разработки;

(4) ошибки в аппаратуре, на которой эти программы выполняются.

В данной статье ошибки, обусловленные причинами 2–4 , рассматриваться не будут. Как правило, синтаксический анализ программы проводится в начале этапа трансляции. Однако ВКК, с помощью которого создаются капсулы, исключает появление синтаксически неверных инструкций. Данный компонент использует всегда актуальное XML-представление формата операндов.

Семантические ошибки, т. е. логически неверное использование капсульного языка, более вероятны. Следовательно, в СКАТ должны быть встроены инструменты, помогающие избегать появления такого рода ошибок. Однако построение функционала, позволяющего полностью избежать появления ошибок в программах, невозможно.

Для того чтобы разработать инструментарий, позволяющий снизить вероятность появления ошибок на этапе создания капсулы, необходимо:

– определить множество возможных ошибок;

– оценить целесообразность их обнаружения на этапе создания капсулы;

– оценить требуемые для обнаружения ошибок вычислительные и временные ресурсы;

– в случае принятия решения об обнаружении определенных ошибок на этапе создания капсулы, требуется разработать алгоритм работы данной функциональности.

Во время работы над спецификацией РОУ было проведено исследование, результатом которого стало формирование множества возможных исключений, которые могут генерироваться аппаратурой во время выполнения капсулы. Данное множество формировалось исходя из возможных ошибок, обнаруживаемых в функциональных блоках РОУ. Можно привести перечень данных ошибок:

– Значения функциональных полей в операнде не специфицированы. Этот случай исключается из рассмотрения в рамках данной задачи, так как подобная ситуация возможна по причине либо неправильного функционирования ВКК, который не должен давать пользователю сформировать операнд с не специфицированными значениями функциональных полей, либо из-за ошибок в работе аппаратуры.

– Значения функциональных полей двух операндов, которые будут участвовать в какой-либо операции (сцепившиеся операнды) [6], взаимно противоречивы, т. е. РОУ не может корректно работать с данными значениями.

– Отсутствие определенных операндов в той части капсулы, в которой они должны быть расположены.

– Неверная настройка распределения операндов по секциям.

– Некоторые функциональные поля в операндах имеют такие значения, что не ясен дальнейший путь передачи данных.

Взаимную противоречивость функциональных полей сцепившихся операндов возможно обнаруживать в СКАТ на этапе создания капсулы только лишь для первой пары операндов каждой секции. Это обусловлено тем фактом, что во время выполнения капсулы вторая и последующие пары операндов для каждой секции выбираются исходя из контекста вычислительного процесса, а для его формирования требуется запустить данный вычислительный процесс. Следовательно, вводить функционал обнаружения данного типа ошибок на этапе создания капсулы нецелесообразно по причине низкой эффективности данной логики для капсул, которые содержат больше восьми операндов. Капсулы, как правило, содержат гораздо большее число операндов.

Отсутствие операндов в строго определенных частях капсулы можно считать нарушением структуры капсулы. Данная ошибка возможна в большинстве

случаев только вследствие неправильного программирования, следовательно, целесообразно возложить проверку данных ситуаций на СКАТ до выполнения капсулы в РОУ. Здесь следует ввести некоторое разделение ошибок данного типа.

В первом случае нарушение структуры (принятого формата) капсулы возможно при несоблюдении правил расположения операндов в определенных частях капсулы. Например, некоторые управляющие операнды могут находиться только в начале капсулы.

Во втором случае возможна ситуация, когда в операнде из управляющей части капсулы указаны номера-ссылки на операнды в несуществующей части капсулы — выход за пределы диапазона количества операндов в капсуле либо указание на ту часть капсулы, в которой не может быть данных операндов.

Неверная настройка распределения операндов по секциям, а также значения функциональных полей, которые не позволяют определить направление пересылки данных, могут быть обнаружены только во время выполнения капсулы, так как имеет место зависимость от контекста вычислительного процесса.

Таким образом, принимается решение о том, что на этапе создания капсулы целесообразно обнаруживать только ошибки, связанные с нарушением структуры капсулы. В условиях применения современного аппаратного обеспечения персональных компьютеров вычислительные ресурсы, требуемые для реализации данной проверки, невелики по сравнению со временем выполнения моделирования капсулы в РОУ и затрачиваемых при этом оперативной памяти и процессорных мощностей.

Переходя к реализации обнаружения вышеуказанных ошибок, необходимо учитывать, что СКАТ разрабатывалась с требованием гибкой настройки на изменяющуюся спецификацию РОУ. Следовательно, несмотря на жесткое определение формата капсулы в данный момент, следует закладывать возможность его быстрого изменения. Таким образом, формат капсулы необходимо специфицировать в таком виде, который обеспечит возможность его изменения без перекомпиляции СКАТ.

Проверку на соответствие капсулы специфицированному формату лучше всего проводить перед началом трансляции ее в БП. Это обусловлено тем, что при программировании вначале могут создаваться содержательные операнды, над которыми будут происходить вычисления, а уже после этого — конфигурационные и управляющие, которые разместятся в начале капсулы.

Проанализировав различные варианты реализации спецификации на формат капсулы, разработчики СКАТ пришли к выводу, что лучше всего для этого подходит использование языка регулярных выражений. Суть предлагаемого способа заключается в том, что с помощью регулярного выражения описывается корректная структура капсулы. Затем в строку собираются все сокращенные обозначения операндов капсулы и проверяется соответствие получившейся строки

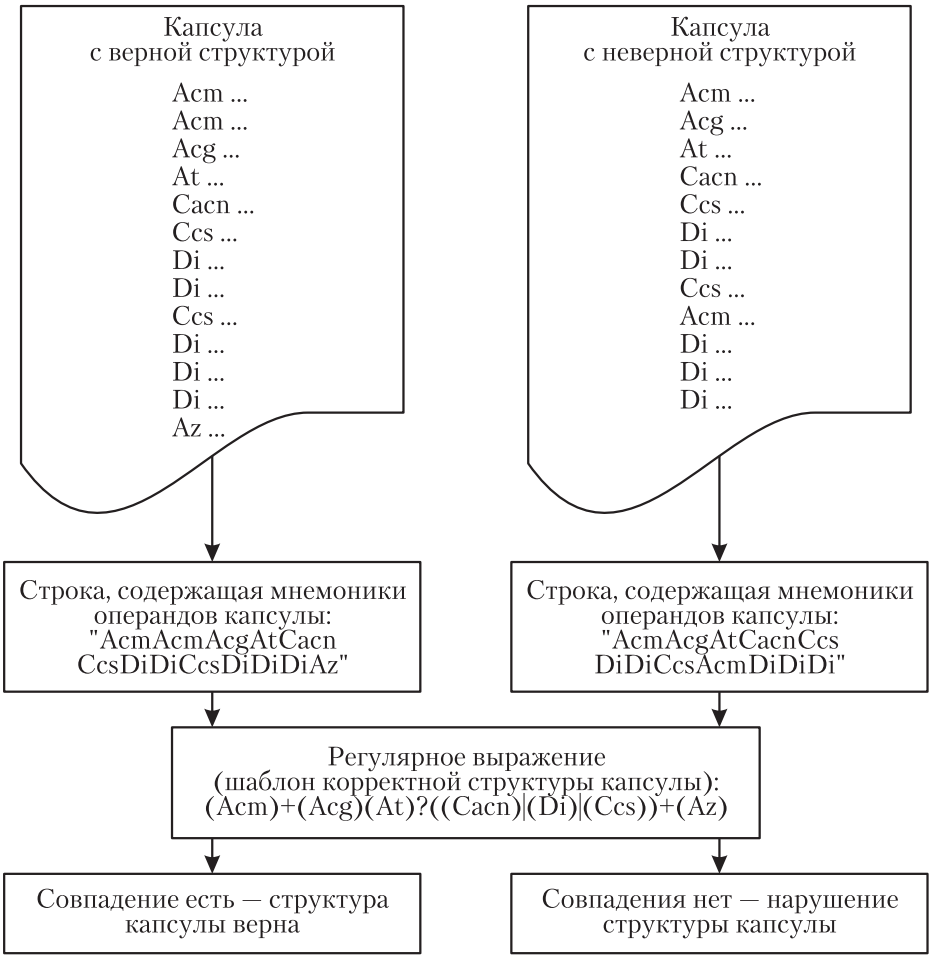

Рис. 1 Пример анализа структуры капсулы на соответствие шаблону

шаблону. Если соответствие найдено, то это означает, что структура капсулы верна.

Пример данной проверки показан на рис. 1. Операнды в капсулах обозначены сокращениями — мнемоникой, многоточие означает функциональные поля. В данном случае важно не назначение операндов, а тот факт, что корректная капсула должна начинаться с Acm-операндов, а заканчиваться операндом Az.

Для того чтобы определить, корректно ли указаны ссылки на операнды в управляющей части капсулы, необходимо проанализировать значения этих ссылок. В случае если они содержат величину, выходящую за границу количества

операндов в капсуле, это считается ошибкой. Также ошибочной будет ситуация, когда ссылки указывают на операнды, тип которых недопустим в данном месте (например, ссылка на операнд из управляющей части капсулы).

В случае обнаружения вышеуказанных ошибок пользователю должно выдаваться сообщение, содержащее тип ошибки и указание на операнд, содержащий ошибочные данные.

4 Автоматизация процесса локализации обнаруженных ошибок

В РОУ реализована логика обнаружения и обработки исключений, возникающих в ходе вычислительного процесса. Эти исключения могут возникнуть во время моделирования выполнения капсулы по причине вышеописанных ошибок. Следовательно, необходимо иметь в СКАТ инструментарий, позволяющий программисту быстрее локализовать обнаруженные ошибки. Описываемый инструмент называется компонентом обработки исключительных ситуаций (КОИС).

Общий алгоритм работы КОИС можно представить в виде трех этапов:

(1) сбор отладочной информации об исключении;

(2) анализ информации и локализация ошибки;

(3) если место возникновения ошибки определено, то выдача пользователю этой, а также всей собранной отладочной информации.

Под отладочной информацией подразумевается тип ошибки, время и место ее обнаружения. Эта информация хранится во внутренних регистрах блока обработки исключений в РОУ, следовательно, может быть получена при анализе файла-отчета, созданного в результате моделирования выполнения капсулы в ModelSim.

Именно на этапе анализа собранной отладочной информации возможно внедрение автоматизированных средств локализации обнаруженных ошибок. После проведения исследования возможности создания подобного инструментария, было принято решение о том, что это реализуемо в разумные сроки, и был разработан алгоритм, согласно которому должен функционировать поиск мест возникновения ошибок.

Суть предлагаемого алгоритма заключается в том, что во время обработки файла-отчета, полученного после моделирования, могут быть обнаружены все зафиксированные аппаратурой ошибки в виде сгенерированных исключений. По типу каждого исключения может быть определен конкретный вид ошибки. Как правило, ошибки могут возникнуть по причине таких значений функциональных полей в операндах, которые приводят к нарушению правильной семантики программы. Далее происходит поиск в капсуле исходных операндов, содержащих неверные значения функциональных полей.

В СКАТ файл-отчет моделирования транслируется во внутреннее представление [3], и оно используется для осуществления пошаговой отладки, во время которой пользователю предоставляются в графическом (удобном для восприятия) виде значения данных в определенных узлах РОУ на каждом шаге выполнения капсулы. Если обнаружено появление исключения, то значения, по которым был определен факт исключения, подсвечиваются.

Пользователь может запустить автоматический поиск места возникновения ошибки. Он работает по принципу обратной трассировки, т. е. на шаге, на котором обнаружено исключение, собирается информация о текущем операнде и начинается обратный проход по шагам и отслеживание преобразований, которые породили данный операнд. Упрощенно процесс можно описать следующим образом:

1. Шаг № 3. Обнаружено исключение. Функциональное поле ФП операнда О3 содержит неверное, с точки зрения семантики, значение. Операнд О3 находится в функциональном блоке ФБ3.
2. Анализ шага № 2. Операнд О3 является результатом действий над операндами О2 и О1 в функциональном блоке ФБ2. Функциональное поле ФП попало в О3 из О2.
3. Анализ шага № 1. Операнд О2 находится в функциональном блоке ФБ1. Сюда он попал из БП, которая содержит в себе капсулу.
4. Поиск в исходной капсуле операнда О2. Именно его функциональное поле ФП стало причиной исключения в ФБ3 на шаге № 3.
5. Осуществляется выдача пользователю сообщения о том, что ошибка находится в ФП в операнде О2 в исходной капсуле и необходимо его изменить.

Приведенная последовательность вычисления места возникновения ошибки сильно упрощена, так как зачастую ошибочные значения функциональных полей находятся сразу в двух операндах и необходимо проводить обратную трассировку для каждого из них. Чем больше номер шага, тем, соответственно, больше итераций требуется для прохода через все шаги к искомому операнду в исходной капсуле.

5 Заключение

В ходе проведенной модернизации в СКАТ были внедрены следующие функции:

1. Проверка соответствия структуры капсулы шаблону, заданному в виде регулярного выражения.
2. Проверка корректности ссылок на операнды, которые содержатся в управляющей части капсулы.

3. Автоматический поиск в капсуле исходного операнда, содержащего такие значения функциональных полей, которые приводят к генерации исключения в РОУ в процессе выполнения капсулы.

В результате вышеуказанных изменений была исключена возможность запуска на выполнение в РОУ капсулы, имеющей нарушение в своей структуре, а также значительно сократилось время отладки в случаях, когда необходимо найти и исправить ошибку, приводящую к генерации исключений.

Литература

1. *Степченков Ю. А., Петрухин В. С.* Особенности гибридного варианта реализации на ПЛИС рекуррентного обработчика сигналов // Системы и средства информатики. Доп. вып. — М.: ИПИ РАН, 2008. С. 118–129.
2. *Зеленов Р. А., Степченков Ю. А., Волчек В. Н., Хилько Д. В., Шнейдер А. Ю., Прокофьев А. А.* Система капсульного программирования и отладки // Системы и средства информатики. — М.: ТОРУС ПРЕСС, 2010. Вып. 20. № 1. С. 24–30.
3. *Степченков Ю. А., Волчек В. Н., Петрухин В. С., Прокофьев А. А., Зеленов Р. А.* Механизмы обеспечения поддержки алгоритмов цифровой обработки речевых сигналов в рекуррентном обработчике сигналов // Системы и средства информатики. — М.: ТОРУС ПРЕСС, 2010. Вып. 20. № 1. С. 30–46.
4. ModelSim Tutorial. http://www.actel.com/documents/modelsim_tutorial_ug.pdf.
5. *Тэллес М., Хсих Ю.* Наука отладки. — М.: Кудиц-образ, 2003. 560 с.
6. *Степченков Ю. А., Волчек В. Н., Петрухин В. С., Прокофьев А. А., Зеленов Р. А.* Цифровой сигнальный процессор с нетрадиционной рекуррентной потоковой архитектурой // Проблемы разработки перспективных микро- и наноэлектронных систем — 2010: Сб. трудов. — М.: ИППМ РАН, 2010. 694 с.

ВСТРОЕННЫЕ СРЕДСТВА ОТЛАДКИ РЕКУРРЕНТНОГО ОПЕРАЦИОННОГО УСТРОЙСТВА*

А. А. Прокофьев[1], *Р. А. Зеленов*[2], *В. Н. Волчек*[3]

Аннотация: Рассмотрены проблемы обеспечения наблюдаемости и управляемости внутренних состояний рекуррентного операционного устройства (РОУ). Рассмотрены вопросы внедрения средств, позволяющих разработчику максимально быстро обнаружить, локализовать и исправить ошибки в РОУ, предложены наиболее эффективные, учитывающие архитектурную новизну и специфику его организации решения.

Ключевые слова: отладка; аппаратура; рекуррентность; потоковая архитектура

1 Введение

В Институте проблем информатики Российской академии наук ведется разработка многоядерной потоковой рекуррентной вычислительной системы (МПРВС) с нетрадиционной архитектурой. Данная архитектура является развитием потокового подхода. В текущей реализации осуществляется взаимодействие двух уровней: функции управляющего уровня возложены на фон-неймановский процессор, а в качестве операционного уровня выступает РОУ, имеющее в своем составе четыре процессорных ядра. Управляющий уровень осуществляет взаимодействие с периферийными устройствами и организует подготовку данных для дальнейшей их обработки в РОУ [1, 2].

В качестве исполняемых программ для РОУ выступают капсулы [3]. Это программы, представляющие собой набор элементов самодостаточных данных (операндов), содержащих данные и инструкции для их обработки, а также инструкции настройки режимов работы РОУ. Капсула хранит рекуррентно сжатый алгоритм решения конкретной задачи, который разворачивается в ходе выполнения в РОУ.

Реализуется МПРВС в базисе ПЛИС (программируемой логической интегральной схемы), в качестве языка описания аппаратуры используется VHDL.

*Работа выполнена при частичной финансовой поддержке по Программе фундаментальных исследований ОНИТ РАН на 2011 г., проект 1.5.

[1]Институт проблем информатики Российской академии наук, a.a.prokofyev@mail.ru

[2]Институт проблем информатики Российской академии наук, graf.developer@gmail.com

[3]Институт проблем информатики Российской академии наук, v_volchek@inbox.ru

На текущем этапе проведена верификация как отдельных функциональных блоков, так и всего РОУ в целом. Подготовлен набор капсул, реализующих алгоритмы, требующиеся для решения задачи распознавания слов диктора [4]. На следующем этапе предстоит комплексная отладка разработанного программного и аппаратного обеспечения на инструментальной плате фирмы Altera. Так как МПРВС является системой реального времени и корректность ее функционирования зависит от времени выполнения отдельных программ и скорости работы аппаратуры, невозможно ограничиться лишь отладкой VHDL-модели, необходимо также иметь возможность проверки и отладки устройства непосредственно в аппаратуре (в реальных условиях).

Процесс отладки в аппаратуре является итерационным. Сначала требуется проверить работоспособность каждого из блоков в отдельности, заменяя сопряженные с ним блоки заглушками (имитирующими входные воздействия, например, путем подачи на входы отлаживаемого блока определенной последовательности данных). Затем проверяется взаимодействие отдельных пар блоков, после чего можно приступать к комплексной отладке.

В данной статье рассматриваются вопросы обеспечения наблюдаемости и управляемости внутренних состояний РОУ, а также проблемы реализации функций отладки, связанные с особенностями архитектуры МПРВС, а именно:

- графодинамическим представлением алгоритмов — отсутствуют понятия адреса команды и данных [5];
- параллельным выполнением ветвей графа алгоритма.

2 Функции отладки в рекуррентном операционном устройстве

Функции отладки в РОУ направлены на поиск и устранение ошибок в программной и аппаратной частях системы. Источниками ошибок могут служить как неверные логические предпосылки при создании спецификаций, сложность учета всех возможных факторов, влияющих на работу системы, так ошибки инженеров, занимающихся проектированием и реализацией аппаратуры и программного обеспечения. Чтобы упростить процесс выявления и локализации ошибок, необходимо реализовать определенный набор функций, которые позволят провести отладку.

Существует пять основных приемов комплексной отладки микропроцессорной системы [6]:

(1) останов функционирования системы при возникновении определенного события;

(2) чтение/изменение содержимого памяти или регистров системы;

(3) пошаговое отслеживание поведения системы;

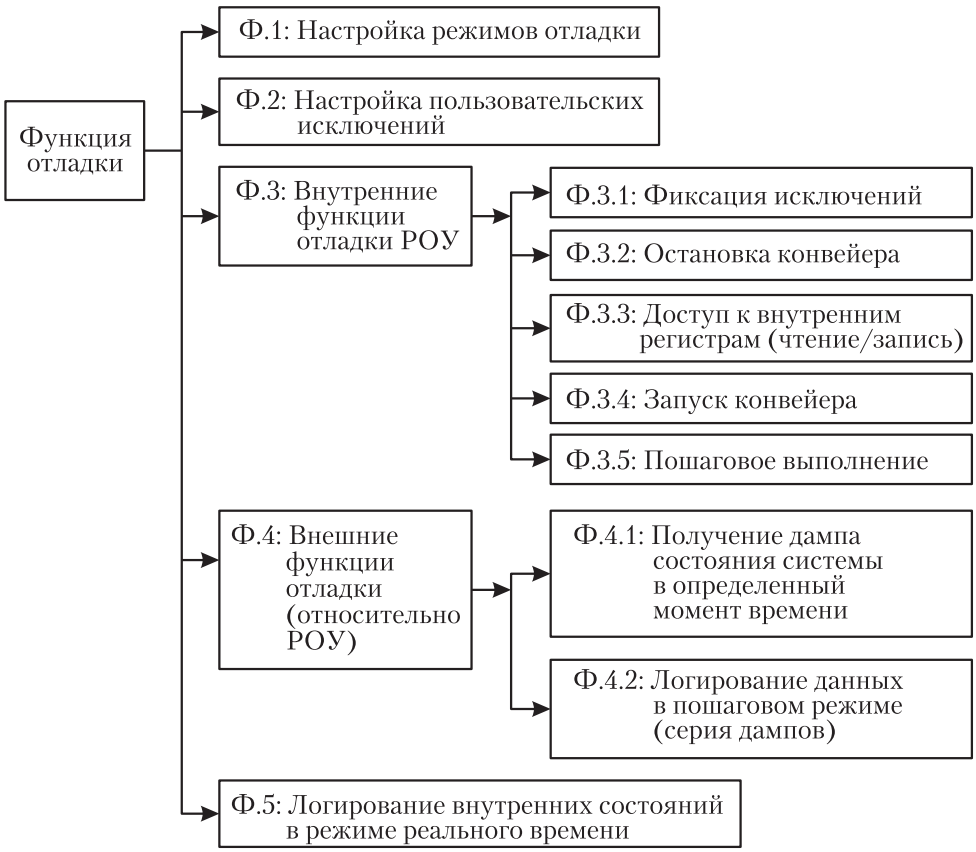

Рис. 1 Структура функции отладки

(4) отслеживание поведения системы в реальном времени;

(5) временно́е согласование программ.

На рис. 1 представлена структура функций отладки, которые обеспечивают возможность использования приемов, описанных выше. Реализация этих функций должна удовлетворять критериям:

– полноты отладочной информации;

– легкости получения и анализа информации;

– минимизации аппаратных затрат на реализацию с целью снизить влияние на характеристики устройства.

3 Реализация встроенных средств отладки рекуррентного операционного устройства

Функции отладки и обнаружения исключительных ситуаций объединены в РОУ в виде единого функционального блока отладки и фиксации исключений (БОФИ). Такая реализация позволяет оперативно управлять состоянием конвейера, избегать потери данных и нарушения хода вычислений при возникновении исключения. Для обеспечения доступа к внутренним регистрам и памяти внутри РОУ организована единая отладочная шина. По ней передается следующая информация:

1. Флаг работы в режиме отладки (1 бит). При единичном значении работа конвейера приостанавливается, давая возможность последовательно обратиться ко всем необходимым внутренним регистрам и памяти.
2. Флаг разрешения записи (1 бит). При единичном значении разрешает изменение значения регистра или ячейки памяти, к которой осуществляется обращение.
3. Адрес (32 бита). Определяет, с каким конкретно регистром или ячейкой памяти осуществляется взаимодействие.
4. Данные для записи (32 бита).
5. Считанные данные (32 бита).

Введение режимов отладки требует обязательной корректировки VHDL-кода, а именно частей, описывающих синхронные процессы. На рис. 2 дополнительная логика выделена графически. Условно описание процесса можно разбить на три блока:

(1) аппаратный сброс (строки 2–4, рис. 2);

(2) режим отладки (строки 5–24);

(3) нормальный режим работы (строки 25–27).

Рассмотрим часть, описывающую режим отладки, подробнее. Из условия на строке 5 видно, что режим отладки активируется путем установки единичного значения сигнала debug_mode. Данный сигнал должен быть глобальным и одновременно останавливать все синхронные процессы в устройстве, чтобы не затронуть ход вычислений. В режиме отладки обеспечивается возможность получить доступ к внутренним регистрам: в зависимости от адреса на шину данных (debug_out_data) подается либо значение регистра, либо высокий импеданс. Обязательным условием является обеспечение уникальности адресов с целью предотвращения коллизий.

```
 1  PROCESS (clk,rst,debug_mode) BEGIN
 2     IF rst='1'THEN
 3        tag_data_reg   <= (others => '0');
 4        debug_out_data <= (others => 'Z');
 5     ELSIF clk'event AND clk='1' AND debug_mode='1' THEN
 6        IF debug_address(31 DOWNTO 30)=core_num THEN
 7           CASE debug_address(29 DOWNTO 0) IS
 8              WHEN "00000000000000000000000010000" =>
 9                 debug_out_data <= tag_data_reg(31 DOWNTO 0);
10                 IF debug_wren='1' THEN
11                    tag_data_reg <= tag_data_reg(34 DOWNTO 32)
12                                  & debug_in_data;
13                 END IF;
14              WHEN "00000000000000000000000010001" =>
15                 debug_out_data <= "00000000000000000000000000000000"
16                                 & tag_data_reg(34 DOWNTO 32);
17                 IF debug_wren='1' THEN
18                    tag_data_reg <= debug_in_data(2 DOWNTO 0)
19                                  & tag_data_reg(31 DOWNTO 0);
20                 END IF;
21              WHEN others =>
22                 debug_out_data <= (others => 'Z');
23           END CASE;
24        END IF;
25     ELSIF clk'event AND clk='1' THEN
26        -- Тело процесса
27     END IF;
28  END PROCESS;
```

Рис. 2 Пример синхронного процесса

Из рис. 2 видно, что адресное пространство логически разделено:

– старшие биты указывают номер вычислительного ядра (строка 6);

– средняя часть определяет регистр (строки 8, 15);

– младшие биты определяют часть регистра, если его длина превышает 32 разряда.

Таким образом, БОФИ получает доступ к остановке и запуску конвейера (реализуются функции Ф.3.2 и Ф.3.4 с рис. 1), а также возможность чтения / записи внутренних регистров (Ф.3.3). Пошаговое выполнение (Ф.3.5) является ничем иным, как автоматическим переходом в режим отладки после запуска устройства в нормальном режиме в течение определенного количества тактов. Очевидно, что в режиме отладки работа БОФИ должна контролироваться извне, так как процедуры считывания данных выполняются специальным ПО или программистом. Кроме этого, необходимо обеспечить возможность настройки режимов отладки (Ф.1).

Для доступа к БОФИ с хост-компьютера используется интерфейс JTAG [7], а для доступа со стороны управляющего уровня — 32-разрядные шины данных и адреса. Взаимодействие управляющего уровня с БОФИ необходимо для реализации служебных процедур, когда требуется доступ к внутренним регистрам РОУ.

В нормальном режиме работы РОУ БОФИ воспринимает поступающие данные как инструкции (установка настроечных регистров либо требование перейти в режим отладки). В режиме же отладки интерфейсные шины напрямую соединяются с внутренними шинами отладки, возврат в нормальный режим инициализируется подачей нулевого значения адреса.

Что касается фиксации исключений (Ф.3.1), то в результате анализа функциональных блоков РОУ выявлены и специфицированы ситуации, которые могут негативно повлиять на ход вычислительного процесса. К ним относятся ошибки, возникающие в результате обнаружения недопустимых значений функциональных полей; попыток передачи данных на одну шину из различных источников; из-за противоречивости значений функциональных полей в парах операндов; при обращении к несуществующей ячейке памяти; при переполнении в результате вычислений и т. п. Все подобные ситуации внесены в перечень исключений с классификацией по типам, приоритетности и катастрофичности последствий. Наличие ошибки, входящей в перечень выявленных исключений, фиксируется непосредственно в каждом из функциональных блоков и передается в БОФИ, где, в свою очередь, генерируется информационный код, содержащий в себе данные о месте, типе и времени ошибки. При необходимости может быть произведена остановка конвейера и сгенерирован сигнал прерывания.

Так как в архитектуре РОУ (потока данных) отсутствует понятие адреса команд, определение места на потоковом графе алгоритма, в котором находится вычислительный процесс, составляет определенные трудности. Требуется введение дополнительных аппаратных счетчиков, стартующих по определенным системным событиям:

- счетчик шагов выполнения капсулы — стартует по началу выполнения капсулы, инкрементируется после выполнения очередного вычислительного шага;
- индексные регистры — служат для считывания операндов из капсулы, по их значениям можно определить, какая капсула выполняется в данный момент;
- счетчик циклов выполнения алгоритма — запускается при старте вычислений, инкрементируется после прохождения очередного цикла вычислений.

Сочетание значений этих счетчиков позволяет определить временную составляющую в информации о зарегистрированном в ходе вычислений исключении.

С целью упрощения отладки логика фиксации исключительных ситуаций расширена возможностью настраивать условия пользовательских исключений (Ф.2):

– достижение определенного узла в графе вычислительного процесса;
– возникновение какого-либо отладочного события, например:
  - запись операнда в память совпадений;
  - перенаправление операндов;
  - формирование перехода;
  - исполнение кода операции.

Отладочные условия настраиваются и активируются через описанный выше интерфейс БОФИ.

4 Сбор отладочной информации

Сбор информации о ходе вычислительного процесса и предоставление ее пользователю в удобном для анализа виде является необходимой частью процесса отладки. Объема доступной на кристалле ПЛИС памяти недостаточно для хранения огромных массивов данных, представляющих собой дамп состояний внутренних регистров РОУ, полученных в течение нескольких вычислительных шагов. Единственный возможный выход — использование внешней памяти. Это может быть, например, оперативная память на отладочной плате либо память жесткого диска хост-компьютера. В любом случае без передачи управления на внешние относительно РОУ устройства обойтись невозможно (Ф.4), поэтому после фиксации исключения осуществляется останов конвейера и путем генерации прерывания контроль над РОУ передается управляющему уровню.

Управляющий уровень принимает решение о дальнейших действиях:

– сохранить состояния определенного набора регистров в оперативную память и продолжить вычисления (Ф.4.1);
– продолжить вычисления в пошаговом режиме, выполняя логирование состояний системы после каждого шага (Ф.4.2);
– передать управление на хост-компьютер. Программа-отладчик в автоматическом режиме либо при участии пользователя начнет сбор информации. При необходимости также может быть выполнена корректировка внутренних состояний регистров и памяти.

В случае необходимости сбора отладочной информации в режиме реального времени (без остановки конвейера) наиболее эффективным решением является применение встроенного логического анализатора, предоставляемого разработчиками ПЛИС [8]. Применение данного инструмента базируется на использовании незадействованной в проекте памяти ПЛИС. Такая память — ограниченный ресурс, поэтому на время отладки целесообразно по максимуму сократить используемую проектом память (например, уменьшить объем буферной памяти

и памяти совпадений). К плюсам логического анализатора относятся простота использования и отсутствие необходимости вносить в проект какие-либо изменения. Недостатком можно считать то, что каждое переопределение логируемых данных требует перепрограммирования ПЛИС.

5 Визуализация процесса отладки

Лишь обладать информацией о ходе вычислительного процесса недостаточно, необходимо предоставить ее пользователю в удобном и наглядном виде. Оптимальным вариантом будет, если от пользователя не потребуется дополнительных усилий для освоения инструментов встроенной отладки. Поэтому в качестве визуального интерфейса взаимодействия со встроенными средствами отладки РОУ целесообразно использовать интерфейс отладки капсул системы капсульного программирования и отладки (СКАТ) [3].

Взаимодействие СКАТ с аппаратурой осуществляется при помощи интерфейса JTAG [7]. С точки зрения пользователя, отладка в аппаратуре выглядит точно так же, как и отладка капсул на VHDL-модели РОУ. Единственное отличие заключается в том, что изменения внутренних состояний могут быть выполнены только на текущем шаге (при пошаговой отладке) либо не могут быть применены совсем (если СКАТ работает в режиме анализа собранной информации).

Так как работа носит научно-исследовательский характер и имеется большая вероятность внесения изменений в аппаратную часть РОУ, нужно обеспечить возможность легкой подстройки СКАТ под новую структуру аппаратуры. Поэтому информация об адресах внутренних регистров и памяти РОУ предоставляется для СКАТ в виде XML-файла, что позволяет легко модифицировать перечень доступных для отладки регистров без перекомпиляции.

6 Автоматизация диагностирования ошибок

Процесс отладки, а именно установка причины возникновения ошибки, имеет достаточно большую аналитическую интеллектуальную составляющую. Чтобы снизить трудоемкость и ускорить процесс определения, на каком уровне возникает ошибка (управляющий уровень, РОУ, интерфейс взаимодействия уровней), диагностирование можно автоматизировать с помощью СКАТ следующим образом:

1. СКАТ считывает из аппаратуры текущее состояние РОУ, включая часть буферной памяти, соответствующую капсуле, в ходе выполнения которой была зафиксирована ошибка.
2. Структура капсулы (типы операндов и значения функциональных полей) сравнивается с эталоном (капсулой, подготовленной пользователем). Если

отклонений не найдено, то переход к шагу 3. В противном случае можно сделать вывод, что ошибка возникла в процессе записи шаблона капсулы в буферную память, следовательно, необходимо проверить работу управляющего уровня и его синхронизацию с РОУ.

3. Капсула, абсолютно идентичная той, что была выполнена в аппаратуре, запускается на программной модели РОУ.
4. Результат выполнения в модели сравнивается с данными, считанными из аппаратуры. Если они совпадают, то ошибка, скорее всего, не связана с физической реализацией и имеет систематический характер, а дальнейшую отладку можно проводить на VHDL-модели РОУ.

Таким образом, в автоматическом режиме можно выяснить, влияет ли реализация в аппаратуре на появление ошибки, что помогает резко сократить область поиска.

7 Заключение

Разработанные средства отладки РОУ позволяют ускорить процесс обнаружения и локализации ошибок. Это достигается за счет введения в аппаратуру дополнительного блока отладки, взаимодействие с которым осуществляется через интерфейс JTAG, таким образом обеспечивается доступ к внутренним регистрам и памяти устройства. С помощью визуального интерфейса программы СКАТ пользователь легко может в пошаговом режиме анализировать внутреннее состояние РОУ и при необходимости модифицировать его. Средства автоматизации диагностирования ошибок, встроенные в СКАТ, позволяют ускорить локализацию ошибки за счет определения уровня на котором она возникла: управляющий уровень либо интерфейс его взаимодействия с РОУ; ошибка в логике работы РОУ или в капсуле; ошибка, связанная с особенностями реализации в аппаратуре (например, ошибки синхронизации).

Литература

1. *Степченков Ю. А., Петрухин В. С.* Особенности гибридного варианта реализации на ПЛИС рекуррентного обработчика сигналов // Системы и средства информатики. Доп. вып. — М.: ИПИ РАН, 2008. С. 118–129.
2. *Степченков Ю. А., Волчек В. Н., Петрухин В. С., Прокофьев А. А., Зеленов Р. А.* Цифровой сигнальный процессор с нетрадиционной рекуррентной потоковой архитектурой // Проблемы разработки перспективных микро- и наноэлектронных систем — 2010: Сб. трудов. — М.: ИППМ РАН, 2010. 694 с.
3. *Зеленов Р. А., Степченков Ю. А., Волчек В. Н., Хилько Д. В., Шнейдер А. Ю., Прокофьев А. А.* Система капсульного программирования и отладки // Системы и средства информатики. — М.: ТОРУС ПРЕСС, 2010. Вып. 20. № 1. С. 24–30.

4. *Степченков Ю. А., Волчек В. Н., Петрухин В. С., Прокофьев А. А., Зеленов Р. А.* Механизмы обеспечения поддержки алгоритмов цифровой обработки речевых сигналов в рекуррентном обработчике сигналов // Системы и средства информатики. — М.: ТОРУС ПРЕСС, 2010. Вып. 20. № 1. С. 30–46.
5. *Филин А. В.* Динамический подход к выбору архитектуры вычислительных устройств обработки сигналов // Системы и средства информатики. — М.: Наука, 2001. Вып. 11. С. 247–261.
6. *Воробьев Н. В., Горбунов В. Л., Горячев А. В. и др.* Микропроцессоры: В 3-х кн. Кн. 3. Средства отладки, лабораторный практикум и задачник: Уч-к для втузов / Под ред. Л. Н. Преснухина. — М.: Высшая школа, 1986. 351 с.
7. Altera, Virtual JTAG (sld_virtual_jtag). Megafunction User Guide. http://www.altera.com/literature/ug/ug_vir tualjtag.pdf.
8. Altera, Quartus II Handbook v11.1.0. Vol. 3: Verification. http://www.altera.com/literature/hb/qts/qts_qii5v3.pdf.

МОДЕЛЬ ПОТОКОВОЙ АРХИТЕКТУРЫ НА ПРИМЕРЕ РАСПОЗНАВАТЕЛЯ СЛОВ*

Д. В. Хилько[1], *Ю. А. Степченков*[2]

Аннотация: Рассмотрена имитационная модель потоковой рекуррентной архитектуры (МПРА), реализуемой в виде сверхбольших интегральных схем (СБИС) на кристалле ПЛИС (программируемой логической интегральной схемы) фирмы Altera. Предлагаемая модель позволяет решать задачи отладки функциональных блоков архитектуры до их внедрения в аппаратуру и служит инструментом разработки и отладки программ. Описаны ключевые аспекты архитектуры, специализированного языка программирования и их отражение в модели.

Ключевые слова: имитационная модель; потоковая архитектура; рекуррентность; программирование

1 Введение

В ИПИ РАН ведутся работы по созданию нетрадиционной рекуррентной архитектуры, предназначенной для реализации параллельных вычислений ограниченной размерности в области цифровой обработки сигналов. Для экспериментальной апробации предлагаемой архитектуры разрабатывается VHDL-модель и СБИС на ее основе — рекуррентный обработчик сигналов (РОС). Он исполняется в гибридном, двухуровневом варианте с ведущим фон-неймановским процессором на управляющем (верхнем) уровне (УУ) и рядом потоковых процессоров на нижнем уровне — рекуррентном операционном устройстве (РОУ) [1]. Апробация РОС осуществляется на примере программы распознавателя слов.

Архитектура РОУ радикально отличается по основным моментам не только от классической архитектуры фон Неймана, но и от других нетрадиционных параллельных архитектур. Для существующих традиционных и нетрадиционных компьютерных архитектур характерно наличие двух потоков: активного потока инструкций и пассивного потока данных. В РОУ оба потока сливаются в один общий поток самодостаточных данных, в котором, помимо собственно

*Работа выполнена при частичной финансовой поддержке по Программе фундаментальных исследований ОНИТ РАН на 2012 г. (проект 1.5) и Программе фундаментальных исследований Президиума РАН (проект 16).

[1]Институт проблем информатики Российской академии наук, dhilko@yandex.ru

[2]Институт проблем информатики Российской академии наук, YStepchenkov@ipiran.ru

обрабатываемых данных, содержится управляющая и служебная информация, необходимая для их обработки (в существующих архитектурах кодируемая в форме инструкций) [2, 3].

Для достижения максимальной производительности предлагаемой архитектуры разрабатывается соответствующее программное обеспечение (ПО), оптимизированное для выполнения в ее среде. В качестве основного инструмента программирования РОС выступает специализированная среда СКАТ [4], использующая VHDL-модель как основное средство исполнения и отладки программ. Однако эта версия СКАТ обладает ограниченным набором утилит автоматизации процесса разработки ПО. Специфические особенности, не свойственные другим архитектурам, делают невозможным применение в полном объеме известных методов и технологий программирования для разработки программ на существующем низкоуровневом языке. Необходим специализированный язык программирования высокого уровня и компилятор для него.

Дополнительным препятствием для успешного программирования в среде МПРА является отсутствие завершенной, полнофункциональной реализации РОС (VHDL-модели, СБИС) в данной работе исследовательского характера.

В результате анализа указанных проблем было решено создать дополнительную, обособленную модель РОС, которая позволила бы отлаживать его программы, анализировать его поведение и локализовывать блоки и функции VHDL-модели, требующие доработки.

2 Ключевые особенности рекуррентного обработчика сигналов и языка его программирования

Рекуррентный обработчик сигналов относится к классу потоковых архитектур, но, в отличие от них, тегируемые данные являются самодостаточными (рекуррентно разворачивающимися). Теги содержат некоторую начальную сжатую информацию, обеспечивающую выполнение требуемой процедуры. Каждый следующий шаг процедуры рекуррентно самоопределяется, в том числе с учетом результата предыдущего шага.

В разрабатываемой архитектуре в состав центрального процессорного устройства (ЦПУ) включено автономное устройство преобразования тегов (ПТ), которое обладает возможностью саморазвертки рекуррентного вычислительного процесса (ВП). Устройство ПТ представляет собой относительно простую комбинационную схему, содержащую средства настройки на предметную область. В памяти находится только набор операндов (элементов самодостаточных данных) и начальных значений их функциональных полей, устройством ПТ динамически подвергаемых рекуррентной развертке. Такое представление программы называется капсулой. Более подробно с предлагаемой архитектурой можно ознакомиться в работе [2].

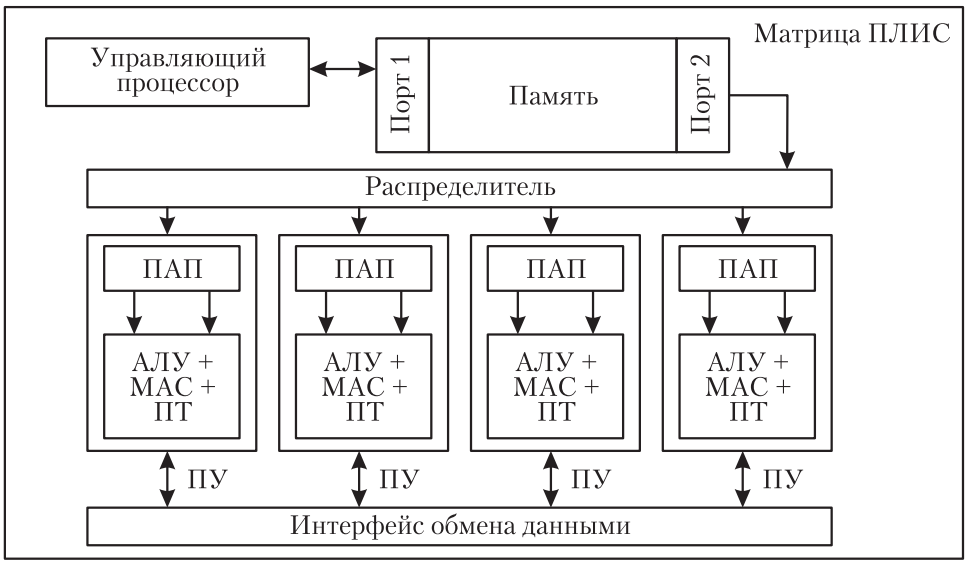

Рис. 1 Структура РОС

В настоящее время РОС реализуется в однокристальном исполнении и имеет четыре процессорных ядра (ПЯ), способных функционировать параллельно. Таким образом, можно говорить о параллелизме на уровне ПЯ. Однако это не единственный уровень параллелизма, задействованный в РОС. Специфика исполнения архитектуры позволяет выделить достаточно глубокий конвейер, каждая из ступеней которого может функционировать независимо. Также можно выделить параллелизм на самом низком уровне — в вычислительных блоках и регистрах. В настоящем исполнении РОС поддерживает несколько разновидностей суперскалярных режимов вычислений [5].

На рис. 1 приведена структура двухуровневой архитектуры РОС, с подробным описанием которой можно ознакомиться в работе [1].

Как было представлено выше, РОС имеет свой уникальный язык программирования, называемый капсульным. В последующих разделах будет приведен пример программы на капсульном языке. Предлагаемому языку в чистом виде (без вспомогательных утилит и подходов) свойственны те же недостатки, что и другим языкам ассемблерного типа:

1. Поскольку алгоритм является параллельным и рекуррентно свернутым, проследить ход ВП практически невозможно в отсутствие специальных навыков или инструментальных средств.
2. При создании архитектуры и капсульного языка преследовалась цель минимизировать размер потока самодостаточных данных. С одной стороны,

это достигается благодаря рекуррентной свертке, а с другой — при помощи специальных конструкций языка. Поэтому по структуре и размеру капсулы невозможно заранее оценить время ее исполнения (в условных вычислительных шагах). При условии, что капсула написана корректно, минимальная длина — ее преимущество, оборачивающееся потерей наглядности.

Поиск возможных путей устранения этих недостатков привел к необходимости построения моделей программ. Развитие точных методов в программировании вызвало возникновение различных формальных моделей программ, в том числе и моделей параллельных программ. Среди них можно выделить: операторные схемы программ, сети Петри, UCLA-графы и др. [6]. Эти модели позволяют представить алгоритм в виде графовых структур, демонстрирующих последовательные и параллельные участки как вычислительного, так и управляющего процесса.

Развитием моделей параллельных программ для РОС стала концепция графодинамики [7]. Согласно этой концепции программа представляется в виде генерируемой последовательности графов ВП. Содержание графов зависит от настройки преобразователя тегов на предметную область. В РОС задано жесткое ограничение на функциональность преобразователя тегов, благодаря чему возможна обратная трассировка от конечного графа (развернутого алгоритма) к свернутому представлению (капсуле). Такая концепция определяет последовательность разработки капсул: построение динамических графов с последующей сверткой конечного графа в капсулу.

3 Имитационная модель рекуррентного обработчика сигналов

Разработка модели преследует несколько целей, наиболее значимые из которых: отладка и доработка VHDL-модели РОС и ее реализации в виде СБИС; разработка и отладка ПО в условиях постоянно совершенствующейся спецификации архитектуры; исследование реакций модели на эволюционное развитие МПРА.

Для построения модели РОС был выбран метод имитационного моделирования, позволяющий охватить все интересующие разработчиков аспекты функционирования РОС как системы в целом, а также функционирование ПО в его среде.

Предлагаемый вариант архитектуры РОС требует представления множества алгоритмов разрабатываемого ПО в виде двухуровневой структуры. К первому уровню можно отнести те алгоритмы, реализация которых нецелесообразна в среде РОУ (например, алгоритмы управления данными, алгоритмы управления ВП и др.). Ко второму уровню относятся алгоритмы, реализующие вычислительные задачи высокой степени параллелизма и сложности. Это является еще одной причиной разработки имитационной модели.

В настоящее время ведутся работы над двумя версиями модели РОС, различными по своей функциональности и назначению. Первая, условно названная «черный ящик» — упрощенная модель, фактически преобразованная в автомат. Вторая, названная «имитатор РОС» — полнофункциональная модель, описывающая поведение основных блоков РОС и позволяющая отслеживать протекание ВП.

А. Модель «черный ящик»

Эта версия модели разрабатывается с целью автоматизации процессов отладки «имитатора РОС», VHDL-модели и аппаратной однокристальной реализации РОС, а также алгоритмов программы распознавателя слов, исполняемых в среде РОС.

Модель является одним из компонентов полнофункциональной версии и позволяет определить корректность или некорректность входного набора данных для заданной входной программы (капсулы) и в случае его корректности — вычислить соответствующий выходной набор данных. Полученный выходной набор отправляется на сравнение с результатами работы «имитатора РОС» или аппаратной реализации РОС (в зависимости от объекта отладки).

Концептуальная структура модели представлена на рис. 2. Она содержит два основных компонента: анализатор входной капсулы и вычислитель.

Анализатор входной капсулы осуществляет ее синтаксический разбор с целью поиска различных операндов, несущих входные данные от УУ. При обнаружении всех необходимых операндов формируется комплект текущих входных данных.

Вычислитель является простым автоматом, осуществляющим поиск комплекта выходных данных в матрице для заданной капсулы (алгоритма) и текущего комплекта входных данных. В случае если соответствия не найдено, фиксируется

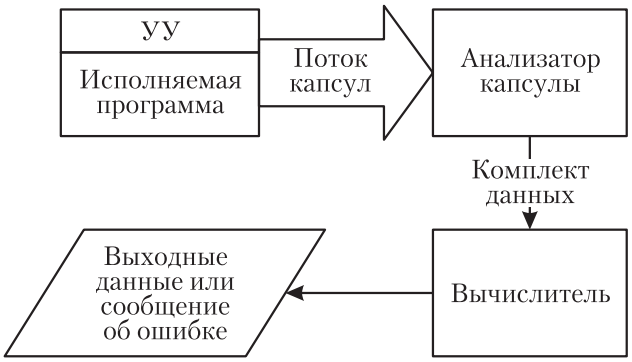

Рис. 2 Структура модели «черный ящик»

факт несоответствия структуры капсулы формату потока данных, поступающих с УУ.

Важно отметить, что указанная матрица формируется заранее программистом, в текущей версии модели — вручную. При дальнейшем развитии в состав модели будет введена утилита формирования матриц.

Б. Модель «имитатор РОС»

Данная версия модели на текущем этапе разработки ориентирована на отладку ПО, разрабатываемого для РОС. Разработчики стремятся к тому, чтобы структура модели максимально соответствовала спецификации РОС на логическом уровне. Данный подход позволяет использовать модель для исследования поведения функциональных блоков и локализации объектов архитектуры, если появится необходимость их доработки. Другими словами осуществляется глубокий анализ капсул и элементов архитектуры и проверка:

(1) соответствия структуры капсулы алгоритма формату потока данных, поступающих с УУ;

(2) соответствия поведения «имитатора РОС» и/или аппаратной реализации РОС требованиям, заложенным в спецификации;

(3) корректности и эффективности реализации исполняемой капсулы;

(4) целесообразности и эффективности функционирования конкретных механизмов архитектуры.

Расширенная версия данной модели предоставляет следующие возможности:

(1) мультипроектность — возможность работы сразу с несколькими проектами отлаживаемых программных систем (например, с несколькими различными проектами распознавателя слов);

(2) создание, редактирование и исполнение программ (капсул) на языке капсульного программирования;

(3) подключение и использование специализированного компилятора для языка капсульного программирования (в настоящее время находится в разработке);

(4) автоматическое и пошаговое моделирование процесса исполнения капсулы или последовательности капсул; эта функция позволит также осуществлять полную трассировку ВП, а при необходимости и корректировать его с сохранением всех изменений в лог;

(5) использование модели «черный ящик» для автоматизированного самотестирования и контроля результатов вычисления капсул.

Концептуальная структура модели, содержащая шесть основных компонентов, представлена на рис. 3.

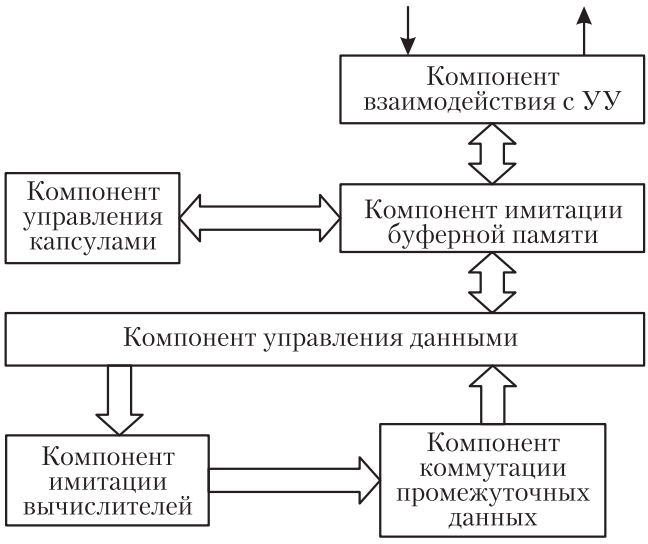

Рис. 3 Структура модели «имитатор РОС»

Компонент взаимодействия с УУ обеспечивает интерфейс, совместимый с ПО УУ. Введение этого компонента позволяет подключать имитационную модель к реальному управляющему процессору.

Компонент имитации буферной памяти совместно с компонентом управления капсулами реализует блок «Память» (см. рис. 1): обеспечивает хранение внутреннего представления капсулы в памяти модели и организует доступ к нему.

Компонент управления капсулами предоставляет сервисные функции для работы с ними (создание, редактирование), а также преобразует исходную капсулу во внутреннее представление модели.

Компонент управления данными — ключевое логическое звено модели — реализуется в виде блока «Распределитель» и четырех блоков «ПАП» (см. рис. 1). Он обеспечивает рассылку операндов в четыре параллельных потока вычислений и образование пар операндов.

Компонент имитации вычислителей, содержащий четыре блока «АЛУ + МАС + ПТ» (см. рис. 1), организует дешифрацию и вычисления в каждом блоке «Вычислитель» в псевдопараллельном режиме (за один такт синхронизации модели последовательно отрабатывают четыре вычислителя).

Компонент коммутации промежуточных данных (блок «Интерфейс обмена данными» на рис. 1) осуществляет сбор и перераспределение промежуточных данных между выходными буферами модели и компонентом управления данными.

Реализация всех указанных функций и компонентов позволит создать универсальное программное средство, пригодное как для отладки ПО, так и для отладки и усовершенствования архитектуры РОС.

4 Пример применения модели

В качестве примера приводится фрагмент капсулы вычисления четырех натуральных логарифмов. Здесь капсула составлена корректно и заполнена корректным набором данных. Ожидаемый результат модели — выходной набор данных. Фрагмент капсулы приводится без комментариев, так как эти данные являются ноу-хау и не раскрываются разработчиками.

```
@d0=-1 Acg: Cx=3 Cp=4 Cc=d Cr= Ce=ecs Cs=ei;
@d0=-1 At: Tc= Tm= Tu= Tr= Tn=0 To=0 Ts= Te= Ta= [Sm=2 Dr=1111 ];
@d0=-1 Cacn: [Oc=*sa Ou=l Sm=0 Dr=1111 Ds=o De= ];
@d0=-1 Ccs: [Oc= Ou=k Sm=0 Dr=1111 Ds= De=] Cj=ri Cl⇒ Cd=Cb=Cs=0 Cm=;
@d0 = 0 Di: V = 6454 [Oc = *sa Ou = l Sm = 1 Dr = 0001 Ds = n De =];
@d0 = 1 Di: V = 16002 [Oc = *sa Ou = l Sm = 1 Dr = 0010 Ds = n De =];
@d0 = 2 Di: V = 11170 [Oc = *sa Ou = l Sm = 1 Dr = 0100 Ds = n De =];
@d0 = 3 Di: V = 8917 [Oc = *sa Ou = l Sm = 1 Dr = 1000 Ds = n De =];
                          _  _  _
@d0=-1 Di: V=ra [Oc=e Ou=k Sm=0 Dr=1111 Ds=o De= ];
                          _  _  _
@d0=-1 Di: V=rc [Oc=e Ou=e Sm=0 Dr=1111 Ds=n De= ];
@d0=-1 Az:
```

В данном примере входные данные выделены курсивом. Если для наглядности обозначить

$$g1 = 6454\,,\ \ g2 = 16\,002\,,\ \ g3 = 11\,170\,,\ \ g4 = 8917\,,$$

то отладочные запуски распознавателя слов с целью извлечения промежуточных значений натуральных логарифмов указанных величин приводят к следующим результатам:

$$\ln(g1) = 7187\,,\ \ \ln(g2) = 21\,947\,,\ \ \ln(g3) = 13\,659\,,\ \ \ln(g4) = 10\,409\,.$$

Результаты запуска модели «черный ящик» отражены на рис. 4 и совпадают с ожидаемыми.

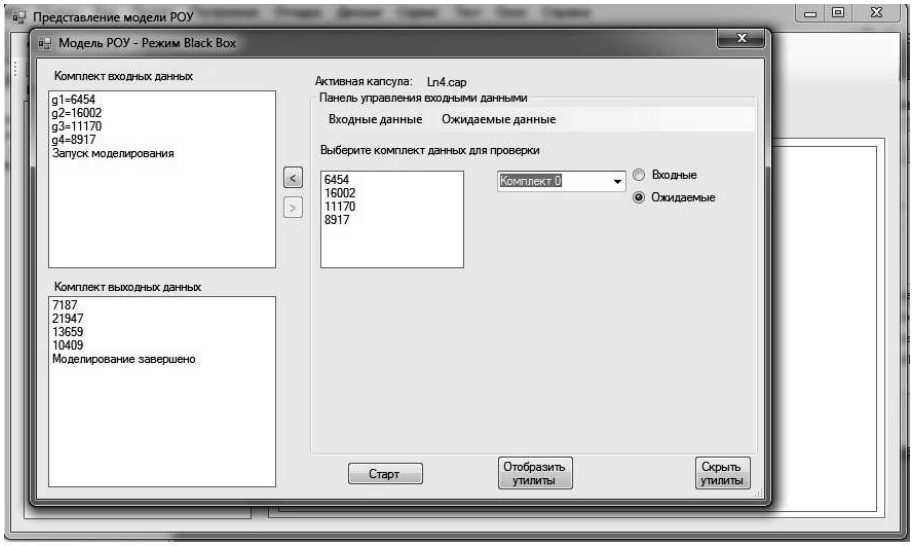

Рис. 4 Результат запуска модели «черный ящик»

5 Заключение

Для повышения эффективности и качества процесса разработки полнофункциональной VHDL-модели РОС и СБИС на ее основе создана имитационная модель РОС, отражающая требования спецификации.

Модель наглядно демонстрирует процессы, протекающие внутри РОС, служит хорошим средством обнаружения ошибок в ПО и поддерживает два режима работы, каждый из которых имеет уникальное назначение и область применения. Оба режима созданы, чтобы предоставлять удобный инструментарий программистам и разработчикам аппаратной реализации РОС.

Разработчики модели надеются, что она станет мощным инструментом отладки и проектирования сложных внутренних механизмов РОС.

Литература

1. *Степченков Ю. А., Петрухин В. С.* Особенности гибридного варианта реализации на ПЛИС рекуррентного обработчика сигналов // Системы и средства информатики. Доп. вып. — М.: ИПИ РАН, 2008. С. 118–129.
2. *Степченков Ю. А., Петрухин В. С., Филин А. В.* Рекуррентное операционное устройство для процессоров обработки сигналов // Системы и средства информатики. — М.: Наука, 2001. Вып. 11. С. 283–315.

3. *Хилько Д. В., Степченков Ю. А.* Вопросы программируемости многоядерной вычислительной архитектуры с единым потоком для эффективной реализации рекуррентных вычислений // Многоядерные процессоры и параллельное программирование; Системы обработки сигналов на базе ПЛИС и цифровых сигнальных процессоров: Сб. ст. Регион. науч.-практ. конф. — Барнаул: Алтайский ун-т, 2011. С. 86–92.
4. *Зеленов Р. А., Степченков Ю. А., Волчек В. Н., Хилько Д. В., Шнейдер Ю. А., Прокофьев А. А.* Система капсульного программирования и отладки // Системы и средства информатики. — М.: ТОРУС ПРЕСС, 2010. Вып. 20. № 1. С. 24–30.
5. *Степченков Ю. А., Волчек В. Н., Петрухин В. С., Прокофьев А. А., Зеленов Р. А.* Механизмы обеспечения поддержки алгоритмов цифровой обработки речевых сигналов в рекуррентном обработчике сигналов // Системы и средства информатики. — М.: ТОРУС ПРЕСС, 2010. Вып. 20. № 1. С. 31–47.
6. *Питерсон Дж.* Теория сетей Петри и моделирование систем / Пер. с англ. — М.: Мир, 1984. 264 с.
7. *Филин А. В.* Динамический подход к выбору архитектуры вычислительных устройств обработки сигналов // Системы и средства информатики. — М.: Наука, 2001. Вып. 11. С. 247–261.