Особенности программной реализации имитационной модели потоковой рекуррентной архитектуры*

Д. В. Хилько[59], Ю. И. Шикунов[60], Ю. А. Степченков[61]

Аннотация: Статья посвящена программной реализации имитационной модели, входящей в состав аппаратно-программных средств моделирования и отладки вычислительного устройства, построенного на основе многоядерной потоковой рекуррентной архитектуры. Приводится описание архитектуры имитационной модели, особенностей программной реализации ключевых компонентов. Рассматривается механизм организации взаимодействия между блоками и компонентами модели, функционирующими параллельно. Также рассматриваются технические возможности текущей версии программой реализации, перспективы их развития. Демонстрируются некоторые этапы функционирования модели на примере вычисления тестового алгоритма.

Ключевые слова: потоковая архитектура; имитационное моделирование; рекуррентность

Введение

В Институте проблем информатики ФИЦ ИУ РАН на протяжении ряда лет ведутся исследования в области нетрадиционных архитектур вычислительных систем и разработки принципиально новых концепций организации вычислительного процесса. Результатом данных исследований является многоядерная потоковая рекуррентная архитектура (МПРА),

*Работа выполнена при частичной финансовой поддержке РФФИ в рамках научного проекта № 13-07-12068

[59] ФИЦ ИУ РАН (ИПИ РАН); 1987 г.р.; dhilko@yandex.ru

[60] ФИЦ ИУ РАН (ИПИ РАН); 1995 г.р.; yishikunov@yandex.ru

[61] ФИЦ ИУ РАН (ИПИ РАН); 1951 г.р.; к.т.н.; ystepchenkov@ipiran.ru

опытный образец которой реализуется в виде гибридной двухуровневой архитектуры рекуррентного обработчика сигналов (ГАРОС) [1, 2].

В целях обеспечения разработки опытного образца была создана платформа аппаратно-программного моделирования и отладки, названная GAROS IDE [3]. Одним из основных компонентов данной платформы является программная имитационная модель (ИМ) рекуррентного обработчика сигналов (РОС), реализованная в программе СИМПРА [4]. На Рис. 1 приведена архитектура имитационной модели РОС.

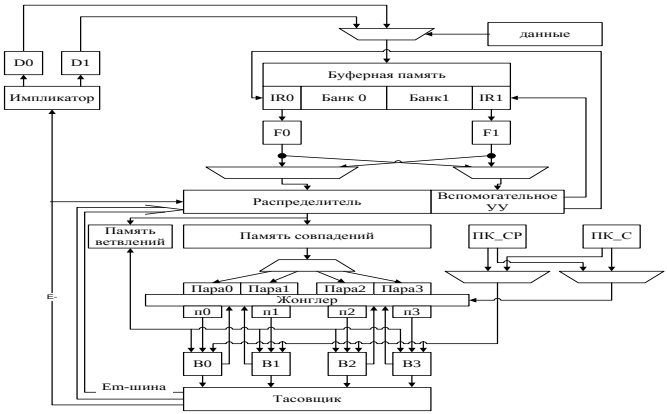

Рис. 1. Архитектура имитационной модели

Указанная модель используется для отладки специализированного ПО, функционирующего в среде ГАРОС, а также аппаратной VHDL-модели. Подробное описание ее составных блоков может быть найдено в статье [5]. Настоящая работа посвящена описанию основных особенностей и возможностей программной реализации разработанной модели.

1. Программная реализация модели

1.1. Общее описание компонент модели

Существуют несколько основных подходов разработки

ИМ. Для реализации данной ИМ была выбрана дискретно-событийная парадигма моделирования. Суть данного подхода заключается в представлении процесса функционирования системы, как хронологической последовательности событий.

В качестве программной среды реализации ИМ была выбрана платформа *.NET Framework*, а в качестве языка программирования – *C#*. Для каждого компонента модели созданы соответствующие классы, реализующие основные элементы дискретно-событийного подхода (часы, списки действий и событий, набор состояний и т.д,), а также механизм межкомпонентного взаимодействия, описываемый в разделе 1.3.

Разработчики РОС разбили вычислительный процесс на три ступени конвейера. На Рис. 1 можно выделить три основных группы компонентов, образующих указанные ступени конвейера: Буферная память + Распределитель + Вспомогательное УУ (ступень 1), Память совпадений + Память ветвлений + Памяти констант (ступень 2), Вычислители + Тасовщик + Импликатор (ступень 3). Совокупное состояние компонентов ступени определяет состояние этой ступени.

Наиболее функционально сложным компонентом модели является «Распределитель», в задачи которого входит управление входным и промежуточными потоками данных и их преобразование для последующей обработки на ступенях 2 и 3 конвейера. Для данного компонента было разработано большое количество режимов работы, опций конфигурации и пр. Высокая функциональная нагрузка на «Распределитель» является его главной отличительной особенностью от остальных компонент, поэтому в разделе 1.2 приводится краткое описание его программной реализации.

1.2. Компонент «Распределитель»

Ввиду особенностей реализации событийной модели в языке C#, события модели были названы действиями. Представление состояний на языке C#

```
public enum DistributorStates
{   Stopped = 0,    // Начальное состояние S0
   InitState = 1,  // Состояние начальной инициализации S1
```

```
    DataWaiting = 2,// Состояние ожидания данных S2
    Grape1Proc = 3, // Состояние обработки первой полугорсти S3
    Grape2Proc = 4, // Состояние формирования второй полугорсти S4
    KeyProc = 5,    // Состояние формирования горсти ключом S5
    MGrapeFinished = 6,// Финальное состояние «Распределителя» S6
    Finished = 7,   // Финальное состояние обработки капсулы S7
    Halted = 8,     // Состояние приостановки работы S8
    Error = 9       // Состояние останова по ошибке S9
}  // Дискретные состояния распределителя
```

Представление действий на языке C#

```
public enum DistrActionName
{   Stop = 0,              // Остановить моделирование
    Init = 1,             // Инициализировать компонент
    Start = 2,            // Старт моделирования
    Hold = 3,             // Приостановить моделирование
    Resume = 4,           // Возобновить моделирование
    Step = 5,             // Выполнить логический шаг
    Configure = 6,        // Установить параметры конфигурации
    CallBS = 7,           // Запросить данные БП
    WriteFReg = 8,        // Записать в регистр F0 или F1
    ReadFReg = 9,         // Читать регистр F0 или F1
    UnPack = 10,          // Распаковать упакованный операнд
    WriteFIFO = 11,       // Записать в FIFO
    ReadFIFO = 12,        // Считать из FIFO
    WriteToBranchMemory = 13,// Записать в память ветвлений
    AddToGrape1 = 14,     // Добавить к первой полугорсти
    AddToGrape2 = 15,     // Добавить ко второй полугорсти
    MergeEIGrapes = 16    // Выбор или объединение E, I-горстей
}  // Внутренние действия распределителя
```

Кроме того, была построена диаграмма состояний компонента «Распределитель», представленная на Рис. 2.

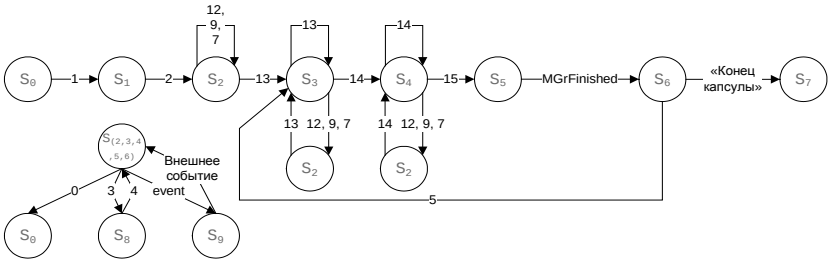

Рис. 2. Диаграмма состояний Распределителя

На диаграмме числами указаны номера действий (событий), в результате которых осуществляется переход из одного состояния в другое, а «капсула» – это наименование программы,

исполняемой в модели.

1.3. Межкомпонентное взаимодействие

Конвейерная реализации РОС потребовала синхронизировать все его ступени на каждом вычислительном шаге. Для решения этой задачи был применен механизм блокировок. Для этого было разработано три функциональных примитива: *DoReadBusData*, *DoStep*, *DoFinalizeStep* и группа состояний для работы с ними. Данные примитивы реализуют основные точки блокировки, позволяя управлять вычислительными потоками и синхронизировать их на каждом шаге.

В качестве точки старта вычислительного шага выступает состояние *DataReady* (см. Рис. 3), сигнализирующее о готовности данных для обработки. Затем, оно порождает исполнение события *DoReadBusData*, реализующего чтение данных из входных шин компонент, после чего для каждого уровня конвейера выполняется событие *DoStep*. По окончании исполнения все процессы ожидают перехода системы в состояние *Finalize*, после чего исполняется событие *DoFinalizeStep*, реализующее очистку необходимых регистров и запись данных на шины для дальнейшего считывания на следующем шаге.

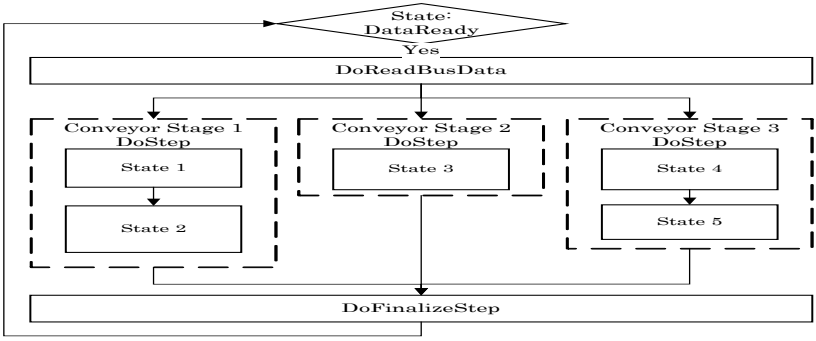

Рис. 3. Механизм межкомпонентного взаимодействия

Данная схема описывает верхний уровень взаимодей-

ствия между ступенями конвейера. В тоже время, если некоторые блоки компонента также могут функционировать параллельно, применяется аналогичная схема взаимодействия.

При разработке ИМ была применена последовательная реализация исполнения параллельных процессов для упрощения отладки логики в каждом отдельном процессе.

2. Возможности и перспективы развития ИМ

Разработанная ИМ входит в состав программных средств платформы GAROS IDE и является одним из ключевых ее компонентов. Текущая версия ИМ обладает следующими техническими возможностями.

1) Позволяет выполнять как одиночный шаг моделирования, так и заданное количество шагов.

2) Имеется возможность осуществлять переход к заданному шагу моделирования (одному из выполненных ранее).

3) Автоматически завершает свое выполнение по достижении условий окончания моделирования, устанавливаемых исполняемой программой.

4) Для каждого сеанса моделирования можно определить степень детализации наблюдаемых результатов.

5) Доступен просмотр состояния любого компонента на каждом шаге моделирования.

6) Визуализация результатов моделирования осуществляется в текстовой форме.

7) Сопоставлять результаты программного моделирования и аппаратного моделирования средствами GAROS IDE.

Достичь подобной гибкости удалось при помощи развитого инструментария ведения лога, хранящего в формализованном виде исчерпывающую информацию о сеансе моделирования для всех компонентов и шагов. В текущей версии логирование осуществляется в соответствии с формируемым пользователем XML-шаблоном, на основе которого строится словарь логируемых сущностей. Сохраняемая информация имеет высокую степень структурированности и формализации.

Потребности разработчиков РОС постоянно возрастают по мере развития и отладки архитектуры. Это обуславливает необходимость расширения функциональных возможностей ИМ и среды GAROS IDE. Имеются следующие перспективы развития.

1) Разработка подсистемы визуализации результатов моделирования, при условии сохранения текстового вывода при необходимости более высокой детализации.

2) Доработка режимов логирования с целью повышения вариативности наблюдений результатов моделирования.

3) Разработка языковых нотаций и подсистем описания новых компонент Им с целью превращения ее в полноценную подсистему имитационного моделирования.

3. Демонстрация процесса моделирования

Процесс моделирования демонстрирует один из алгоритмов распознавания изолированных слов – полосовая фильтрация, – который был реализован в виде специальной программы: капсулы и отлажен при помощи ИМ. На Рис. 4 приведен завершающий шаг моделирования.

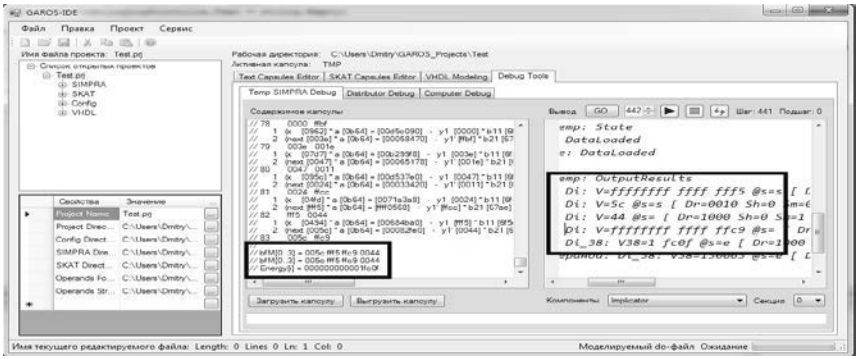

Рис. 4. Завершающий шаг моделирования

Выводы

В статье рассмотрены особенности программной реализа-

ции ключевых компонент и механизмов имитационной модели РОС. Описаны технические возможности модели, функционирующей в рамках аппаратно-программной платформы GAROS IDE. Также продемонстрирована работа модели на тестовом алгоритме.

В соответствии с потребностями разработчиков определены перспективы развития модели до полноценной подсистемы имитационного моделирования, которая станет мощным инструментарием программного моделирования средствами платформы GAROS IDE.

Список литературы

1. *Степченков Ю. А., Петрухин В. С.* Особенности гибридного варианта реализации на ПЛИС рекуррентного обработчика сигналов // Системы и средства информатики, 2008. Доп. вып. С. 118–129.

2. *Хилько Д. В., Степченков Ю. А.* Модель потоковой архитектуры на примере распознавателя слов // Системы и средства информатики. 2012, Т. 22. Вып. 2. С. 8–57.

3. *Хилько Д. В., Шикунов Ю. И.* Разработка инструментальной среды проектирования программного обеспечения для рекуррентно-потоковой модели вычислений // Четвертая школа молодых ученых ИПИ РАН, 2013. Сборник докладов. С. 65–77.

4. *Хилько Д. В., Степченков Ю. А.* Средства имитационного моделирования потоковой рекуррентной архитектуры (СИМПРА). Версия 2. Свидетельство о государственной регистрации программы для ЭВМ № 2014610123.

5. *Хилько Д. В., Степченков Ю. А., Шикунов Ю. И.* Средства имитационного моделирования многоядерной потоковой рекуррентной архитектуры // Многоядерные процессы, параллельное программирование, ПЛИС, системы обработки сигналов: сборник научных статей Всероссийской научно-практической конференции, Барнаул, 28 февраля 2014 г. – С. 58–69.