Использование языков описания процессоров высокого уровня для генерации платформо-зависимых частей операционной системы

Ю. Фонин

Аннотация. В статье проанализированы языки описания архитектуры процессора. Рассмотрены основные элементы микропроцессора. Выделены базовые элементы архитектуры процессора, необходимые для генерации ядра операционной системы, их параметры и свойства.

1. Введение

В настоящее время аппаратное обеспечение все чаще и чаще разрабатывается под конкретное системное программное обеспечение. Такой подход позволяет добиться оптимизации конечной программно-аппаратной системы по многим критериям: производительность, стоимость, потребляемая мощность. Основным достоинством данного подхода является возможность адаптировать аппаратные ресурсы под требования конкретного программного обеспечения.

Процесс разработки носит интерактивный характер. При каждом изменение в аппаратном обеспечении требуется модификация системного программного обеспечения (компилятор, компоновщик, операционная система). Внесение таких изменений вручную может занимать до нескольких человеко-месяцев и тем самым значительно замедлять процесс разработки системы.

Представленное в статье формальное описание архитектуры позволяет разрабатывать системы для автоматического анализа процессора и генерации системного программного обеспечения. Данное описание строится по аналогии с промежуточным представлением, используемым компиляторами языков высокого уровня с учетом специфики архитектуры процессора.

В качестве генератора такого описания может выступать компилятор языка описания архитектуры (ЯОА). Подобные языки используются для разработки процессоров, причем как для создания непосредственно чипа или его программного симулятора, так и для автоматической генерации соответствующего программного обеспечения (компилятора, ассемблера, отладчика).

Статья состоит из введения, трех разделов и заключения. Во втором разделе статьи представлен краткий обзор существующих языков описания

архитектур, приводятся основные конструкции таких языков. В разделе 3 перечисляются базовые элементы процессора, а также свойства и характеристики элементов, необходимые для генерации операционной системы. Последний раздел содержит описание базовых элементов операционной среды, полученных в процессе структурно-функционального проектирования архитектуры процес-сора. В заключении приводится план будущих направлений развития темы.

2. Языки описания архитектуры процессора

В настоящее время существуют три различных стиля для описания архитектуры процессора: структурный, поведенческий и смешанный [1]. Зачастую при описании конкретной архитектуры используются все три типа стилей.

Структурное описание используется в основном для проектирования цифровых схем, поведенческое, как правило, – для моделирования, так как содержит конструкции, которые невозможно реализовать в виде схемы.

2.1. Структурное описание (*structural description*)

Структурное описание соответствует стилю описания, используемому в языках регистровых передач, когда архитектура представляется в виде иерархии связанных компонентов. Для спецификации аппаратуры на уровне регистровых передач (*register transfer level — RTL*) используются языки категории *HDL* (*Hardware Description Languages*), наиболее известными из которых являются *Verilog* [2] и *VHDL* [3].

В качестве примера языка структурного описания приведем язык *MIMOLA* [4]. Описание процессора на языке *MIMOLA* представляется в виде набора блоков, таких как арифметико-логическое устройство, регистры и шины для доступа к памяти, а также набора соединений между блоками.

Анализировать такое описание для генерации компилятора или операционной системы сложно. Поэтому в языке *MIMOLA* предусмотрена возможность ввода дополнительной спецификации компонентов, а именно, предназначение регистров или расположение памяти программ:

```
LOCATION_FOR_PROGRAMCOUNTER PCReg;
LOCATION_FOR_INSTRUCTIONS IM[0..1023];
```

Такие спецификации позволяют получить дополнительную информацию об элементах процессора, необходимую для автоматической генерации компиляторов, операционных систем и создания отладочного программного обеспечения. В частности, в приведенном выше примере указано, что регистр *PCReg* является регистром счетчика команд, а область памяти *IM[0..1023]* предназначена для хранения инструкций.

Однако отсутствие представления инструкций процессора в явном виде делает процесс автоматической генерации модулей операционной системы крайне сложным.

2.2. Поведенческое описание (*behavioral description*)

Для решения задачи автоматической генерации компонентов операционной системы предназначены языки поведенческого класса *ADL* (*Architecture Description Languages*), в которых функциональность каждой инструкции описывается в явном виде. Рассмотрим более подробно некоторые из таких языков.

К языкам класса ADL относится язык *nML* [5]. В *nML* система команд процессора описывается с помощью атрибутных грамматик. Атрибуты включают в себя поведение (*action*), ассемблерный синтаксис (*syntax*) и отображение в машинные коды (*image*). В исходном варианте *nML* отсутствовали механизмы описания многотактовых команд. В последних версиях *nML* поддерживается синтез *VHDL*-описания.

В *nML* инструкция или группа инструкций описывается в виде отдельного блока, например:

```
op num_instruction(a:num_action, src:SRC, dst:DST)
action
{
      temp_src = src;
      temp_dst = dst;
      a.action;
      dst = temp_dst;
}
op num_action = add | sub | mul | div
op add()
action = {
      temp_dst = temp_dst + temp_src
}
...
```

Данный пример содержит функциональное описание блока инструкций. Описывается группа инструкций *num_instruction*, выполняющих базовые арифметические операции: сложение, вычитание, умножение и деление. При вызове данной инструкции выполняются следующие операции:

1. Значение аргумента *src* заноситься в переменную *temp_src*;
2. Значение аргумента *dst* заноситься в переменную *temp_dst*;
3. В зависимости от значения аргумента *a* выполняется одна из четырех возможных операций (*a.action*);
4. Результат операций сохраняется в dst.

Множество возможных операций определяется типом аргумента *a* (*num_action*). В приведенном примере это множество состоит из четырех операций: *add* | *sub* | *mul* | *div*. Для каждой операции разработчик должен описать ее функциональность в блоке *action* (см. пример описания операции *add*).

Как видно из примера, функциональность инструкции или группы инструкций в *nML* может быть описана в виде иерархии функциональных блоков или групп

функциональных блоков. В приведенном примере такой группой является группа из блоков *add*, *sub*, *mul* и *div*. Подобное описание позволяет получить в явном виде дерево каждой инструкции, описывающее ее функциональность, которое может быть использовано для генерации компилятора или операционной системы.

Последователем *nML* стал язык *Sim-nML* [6]. Основное отличие состоит в присутствии в грамматике описания команд дополнительного атрибута использования ресурсов (*uses*), что позволяет специфицировать использование ресурсов и, тем самым, обнаруживать конфликты между командами. В рамках проекта *Sim-nML* были разработаны генераторы, позволяющие автоматически генерировать программные коды симулятора, ассемблера и дисассемблера на основе спецификации процессора на языке *Sim-nML*.

2.3. Смешанное описание

Языки смешанного типа похожи на поведенческие языки, однако они позволяют при описании инструкций специфицировать использование конкретных аппаратных блоков, например, арифметико-логического устройства или специальных умножителей, а также описывать конвейер инструкций. В смешанных языках, как и в поведенческих, инструкции описываются в явном виде. Но при создании дерева инструкций могут возникать трудности, связанные с извлечением информации о функциональности внешних блоков.

Язык *EXPRESSION* [7] позволяет создавать интегрированное описание структуры и поведения подсистемы процессор-память. Спецификация на *EXPRESSION* состоит из шести секций (первые три отвечают за поведение, последние три – за структуру):

- спецификация операций: набор атомарных команд с кодами, описанием параметров и семантики (поведения);
- описание формата команды: команда состоит из ячеек, ответственных за определенный функциональный модуль, которые могут заполняться атомарными операциями для параллельного выполнения;
- отображение общих операций компилятора на машинные операции, описанные в первой секции; данное описание используется в кодогенераторе компилятора;
- описание компонентов: функциональные устройства, шины, порты и т.п.;
- описание конвейера и связей компонентов;
- описание иерархии памяти: регистровая память, кэш, SRAM, DRAM.

Язык *LISA* [8] разрабатывался в качестве средства описания аппаратуры для генерации симуляторов. К ключевым характеристикам *LISA* можно отнести подробное описание конвейера на уровне операций с возможностью задания зависимостей и блокировок. Конвейерные конфликты задаются явно. Каждая команда специфицируется в виде набора операций, которые определяются как регистровые пересылки за время одного такта синхронизации. Описание

аппаратуры на языке *LISA* состоит из двух основных частей: спецификации ресурсов и описания операций. Описание операций в свою очередь содержит следующие секции:

- *DECLARE*: определение объектов и групп через другие объекты и операции – фактически, правила грамматики;
- *CODING*: описание бинарного кодирования операции;
- *SYNTAX*: описание ассемблерного синтаксиса и параметров;
- *BEHAVIOR* и *EXPRESSION*: описание поведения операции в виде кода на языке *C/C++*;
- *ACTIVATION*: описание задержек (*timings*) и поведения конвейера.

Языки *LISA* и *EXPRESSION* позволяют детально описать архитектуру процессора, что важно для синтеза микросхемы. С другой стороны, поскольку инструкции процессора описаны в явном виде, создание систем автоматической генерации операционных систем на основе *EXPRESSION* или *LISA*-описаний процессора существенно проще, чем создание таких систем на основе структурных языков. К недостаткам решения на основе этих языков по сравнению с поведенческими языками следует отнести относительную трудоемкость описания архитектуры, связанную с необходимостью детально описывать структурную составляющую. В этом смысле языки *EXPRESSION* и *LISA* стоят между чистыми поведенческими *ADL*-решениями (типа *nML*) и структурными описаниями уровня *HDL*.

2.4. Выводы

Анализ современных *ADL* высокого уровня показывает, что из *ADL*-описания процессоров можно получить сведения о структуре и функциональности процессора. Фактически, вопрос состоит только в степени сложности использования того или иного языка категории *ADL* для автоматической генерации операционной системы. Отметим также, что для генерации операционной системы требуется лишь некоторое подмножество информации об архитектуре, которая может быть получена из описания процессора. Поэтому целесообразно разработать некий формат промежуточного представления (*ПП*) процессора, аналогичный представлению, применяемому в компиляторах языков программирования высокого уровня. Такое представления, в первую очередь, должно быть удобным и полным для генерации ОС, а с другой стороны, должна существовать возможность создания *ПП* на основе конструкций языков *ADL*.

3. Компоненты промежуточного представления архитектуры, необходимые для автоматической генерации ОС

Для автоматической генерации ОС на основе *ADL*-описания процессора необходимо, выделить список базовых элементов процессора, которые, с

одной стороны, можно получить из *ADL*-описания процессора, а с другой стороны, являются необходимыми для генерации ОС.

3.1. Структурное и поведенческое описание процессора

Структурные элементы описываются набором свойств, соответствующих типу элемента. Функциональные элементы описываются деревом функциональности. Узлами дерева являются микрооперации, указатели на структурные элементы или множества элементов. Функциональный базис определяет множество возможных элементарных операций, которые могут быть применены при описании инструкций.

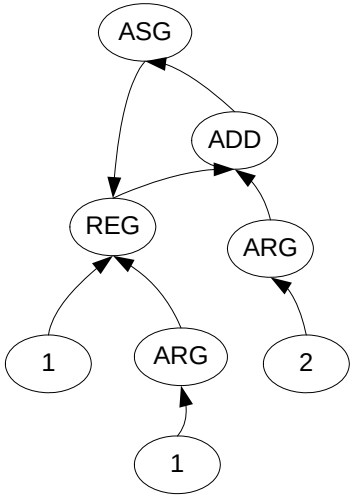

Рис. 1. Пример функционального дерева для инструкции `ADD Ri, #IMM`

На Рис 1 приведен пример представления инструкции сложения регистра с константой. При описании инструкции использованы следующие микрооперации: `ADD (op1,op2)` – сложение двух операндов; `ASG (dst,src)` – присвоение элементу *dst* значения выражения *src*; `REG (lst,idx)` – регистр с индексом *idx* из списка регистров с номером *lst*; `ARG(NUM)` – значение аргумента инструкции с номером *NUM*.

3.2. Базовые элементы архитектуры микропроцессора

Архитектура процессора описывается на основе набора базовых элементов.

Рассмотрим более подробно базовые элементы, приведенные на Рис.2.

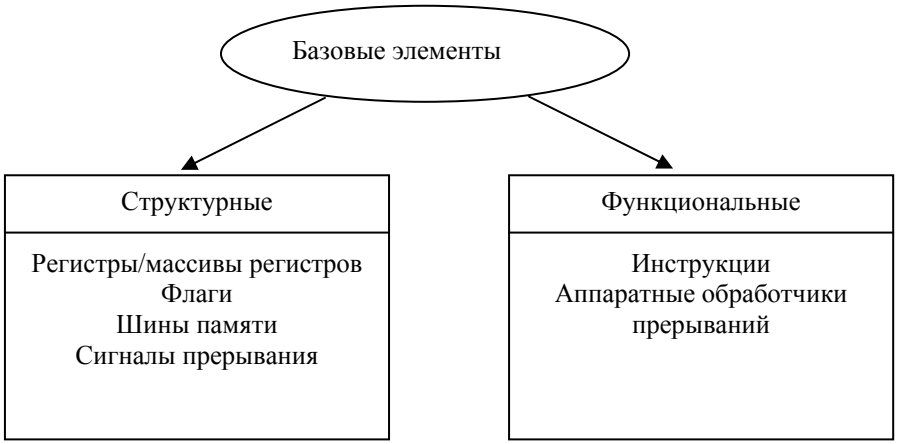

Рис. 2. Базовые элементы процессора.

3.2.1. Регистры

Регистры процессора предназначены для хранения промежуточных результатов вычислений (регистры данных), адресов (адресные регистры) и информации о режиме работы процессора (системные регистры). Каждый регистр описывается следующими параметрами:

- размер регистра в битах;
- ассемблерное имя (если есть);
- список подрегистров (если есть).

Множество регистров, используемых для хранения промежуточных данных пользовательской программы, называется контекстом. Если операционная система поддерживает *многопоточность* (*multi-threading*), то ее частью является функция переключения контекста. При переключении контекста операционная система должна сохранить регистры прерываемой задачи и восстановить регистры загружаемой задачи.

3.2.2. Флаги

Флаги – это элементарные однобитовые ячейки памяти. Флаги могут быть использованы для хранения результатов арифметических операций, например, бита переполнения или знака результата. Кроме того, специальные флаги могут использоваться для управления системой прерываний. Часто флаги являются битами регистров, и изменение состояния флага возможно только через изменение значения регистра.

Флаги характеризуются следующими свойствами:

- имя регистра, которому принадлежит флаг;

- номер бита в регистре;
- ассемблерное имя.

Регистр флагов также является частью контекста. Однако возможна ситуация, при которой одна часть флагов одного регистра принадлежит контексту, а другая часть определяет состояние процессора в целом и не принадлежат контексту. В таком случае при переключении контекста операционной системой должна быть восстановлена только та часть регистра флага, которая определяет состояние исполняемой программы.

3.2.3. Память процессора

Информацию о доступной процессору памяти можно описать в виде набора из трех компонентов: шины, блоки памяти и соединения. Рассмотрим составляющие каждого из компонентов.

Шины

Шина предоставляет интерфейс для работы процессора с памятью и описывается следующим набором характеристик:

- размер шины адреса (в битах);
- размер шины данных (в битах).

Блоки памяти

Блок памяти определяет имеющуюся физическую память и описывается следующим набором характеристик:

- размер блока в байтах;
- размер одной адресуемой ячейки памяти (в битах);
- тип памяти: только для чтения или для чтения и записи.

Соединения

Соединение описывает отображение блоков физической памяти на адресное пространство и определяется следующим набором характеристик:

- блок памяти;
- шина памяти, через которую осуществляется обращение к блоку;
- стартовый адрес, с которого начинается блок.

Информация о блоках памяти необходима для конфигурации менеджера памяти операционной системы. Как правило, операционная система сама отвечает за выделение памяти под стек. Кроме того, во многих современных ОС (включая ОС для встроенных систем) поддерживается динамическое выделение памяти. Таким образом, при портировании ОС на новую архитектуру необходимо иметь информацию о размере, адресах и типах блоков физической памяти, доступной процессору.

3.2.4. Прерывания

Прерывания описываются следующим набором характеристик:

- номер сигнала – запроса на прерывание;

- приоритет прерывания или регистр, значение которого определяет приоритет прерывания;
- процедура обработки запроса на прерывания.

Одной из задач операционной системы является корректная обработка прерываний процессора. Во многих современных ОС для встроенных систем имеются стандартные функции пролога и эпилога. Функция пролога сохраняет регистры и флаги процессора, которые могут быть модифицированы при запуске стандартных функций. Функция эпилога восстанавливает регистры процессора, сохраненные функцией пролога, а также осуществляет корректный возврат из прерывания. Наличие в ОС функций пролога и эпилога позволяет реализовывать драйверы устройств, а также стандартные обработчики прерываний (исключения и программные прерывания) на языке высокого уровня, не задумываясь об особенностях архитектуры конкретного процессора.

Еще одной функцией ОС, связанной с обработкой прерываний, является динамическая инициализация обработчиков прерываний, а также динамическое управление приоритетами прерываний. Операционная система предоставляет набор функций для подключения обработчика прерывания и установки приоритета того или иного прерывания.

Очевидно, что для реализации перечисленных функций необходима информация о прерываниях, аппаратных обработчиках прерываниях и приоритетах прерываний или способах установки приоритетов.

3.2.5. Инструкции

Одним из основных компонентов архитектуры является набор инструкций. Каждая инструкция имеет следующий набор свойств:

- длина кода инструкции – сколько байт в памяти занимает инструкция;
- шаблон бинарного кода инструкции;
- список аргументов;
- ассемблерный синтаксис – шаблон инструкции для языка ассемблера;
- дерево, описывающее функциональность инструкции;
- список режимов, в которых данная инструкция может быть выполнена;
- количество тактов, необходимых для выполнения инструкции.

Аргумент инструкции:

- стартовый бит в коде инструкции;
- длина аргумента в битах;
- шаблон аргумента для ассемблера.

Информация о наборе инструкций процессора необходима для реализации функций операционной системы.

4. Выбор базовых элементов для построения ядра ОС

Проведенный в разд. 2 анализ языков описания процессоров показывает, что в процессе структурно-функционального проектирования архитектуры процессора появляется также возможность выявления базовых элементов, необходимых для автоматической генерации ОС.

Из модели описания архитектуры процессора *явным* образом можно выделить следующую информацию:

- список регистров процессора;
- список флагов процессора;
- список сигналов «запрос на прерывание»;
- операции, выполняемые процессором при возникновении прерывания;
- адресные пространства памяти;
- список инструкций для генерации кода.

Данный набор элементов архитектуры процессора является необходимым, но не достаточным для построения ядра ОС. Помимо перечисленных выше элементов, для построения операционной системы необходимо дополнительно описать следующие элементы:

- список регистров контекста – множество регистров, которые должны быть сохранены при сохранении состояния задачи;
- указатель контекста – регистр, который содержит адрес в памяти для сохранения контекста;
- регистры состояния системы прерываний;
- регистры управления системой прерываний;
- счетчик команд.

Для ряда процессоров перечисленная информация может быть получена из анализа архитектуры, однако это возможно не всегда. Например, в процессоре *ARM7* любой из 16 регистров общего назначения может выполнять функции указателя контекста. В таких случаях необходима дополнительная спецификация элементов.

Также существуют параметры, которые невозможно получить явным или неявным путем из описания архитектуры процессора. Такие параметры должны быть определены разработчиком при проектировании системы:

- список *volatile*-регистров, значения которых не восстанавливаются после вызова функций;
- распределение приоритетов прерываний, которое может быть определено только пользователем в соответствии с потребностями всей системы.

5. Заключение

В статье рассмотрены особенности выделения и описания базовых элементов архитектуры процессора, необходимых для генерации ядра ОС, приведен обзор языков описания архитектуры процессора, а также описаны ключевые моменты, позволяющие достичь эффективного решения поставленной задачи.

Дальнейшее развитие технологии автоматической генерации ОС на основе описания архитектуры процессора заключается в разработке методологии анализа структуры и функциональности процессора и создании алгоритмов генерации функций ОС на основе данного анализа.

Литература

1. Architecture Description Languages for Retargetable Compilation. Wei Qin, Sharad Malik Department of Electrical Engineering Princeton University.
2. IEEE Standard Hardware Description Language Based on the Verilog® Hardware Description Language, IEEE Std 1364-1995.
3. IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1987.
4. The integrated design of computer systems with MIMOLA. Peter Marwedel University of Kiel.