УДК 004.273

Р. И. Попов

# ВЫСОКОУРОВНЕВЫЙ СИНТЕЗ ЦИФРОВЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ ИНФРАСТРУКТУРЫ LLVM

Представлена система высокоуровневого синтеза цифровых схем, разработанная на базе открытой компиляторной инфраструктуры LLVM и используемая при проектировании специализированной аппаратуры. Приведено описание простой программы автоматической генерации микроархитектуры вычислительных устройств на основе алгоритма на языке высокого уровня. Рассмотрены основные этапы синтеза.

***Ключевые слова:*** *высокоуровневый синтез, специализированные процессоры, конвейеризация вычислений, потоковые процессоры.*

**Введение.** Одной из перспективных областей исследования современных средств автоматизации проектирования электронных устройств, объединяющей проблемы компьютерной архитектуры и теории компиляторов, является высокоуровневый синтез. Задача систем высокоуровневого синтеза (High-Level Synthesis — HLS) заключается в отображении поведенческой

модели алгоритма, описанной на языке высокого уровня, на специализированную аппаратную архитектуру [1]. Исследования в этой области ведутся с середины 1980-х гг., но только в последние несколько лет благодаря тенденции к увеличению степени интеграции микросхем такие системы начали завоевывать реальное признание в инженерной практике.

В настоящее время существуют несколько коммерческих HLS-систем, однако для проведения серьезных исследований требуются значительные инвестиции в создание программного каркаса системы. В этой связи важно, что основные этапы синтеза совпадают с этапами компиляции, что позволяет синтезирующее средство разработать на базе компилятора с открытым исходным кодом.

В настоящей статье приведены результаты исследований по созданию такого средства на базе открытой компиляторной инфраструктуры LLVM (низкоуровневой виртуальной машины — Low-Level Virtual Machine), основанной на промежуточном представлении программы в форме с единственным статическим присваиванием. Такая форма легко преобразуется в граф потока данных и управления программой (Control and Data Flow Graph — CDFG), который используется при синтезе аппаратуры.

Основными особенностями рассматриваемой системы являются:

— использование потоковой вычислительной модели без централизованного устройства управления;

— разработка на базе открытой компиляторной инфраструктуры.

**Этапы синтеза и модель представления аппаратуры.** Задача высокоуровневого синтеза находится на стыке теории компиляторов и микроархитектуры вычислительных устройств. На начальных этапах синтеза осуществляются традиционные компиляторные оптимизации, после чего выполняются микроархитектурные аппаратные оптимизации и генерируется спецификация схемы на уровне регистровых пересылок. Схема, отображающая основные этапы высокоуровневого синтеза, показана на рис. 1.

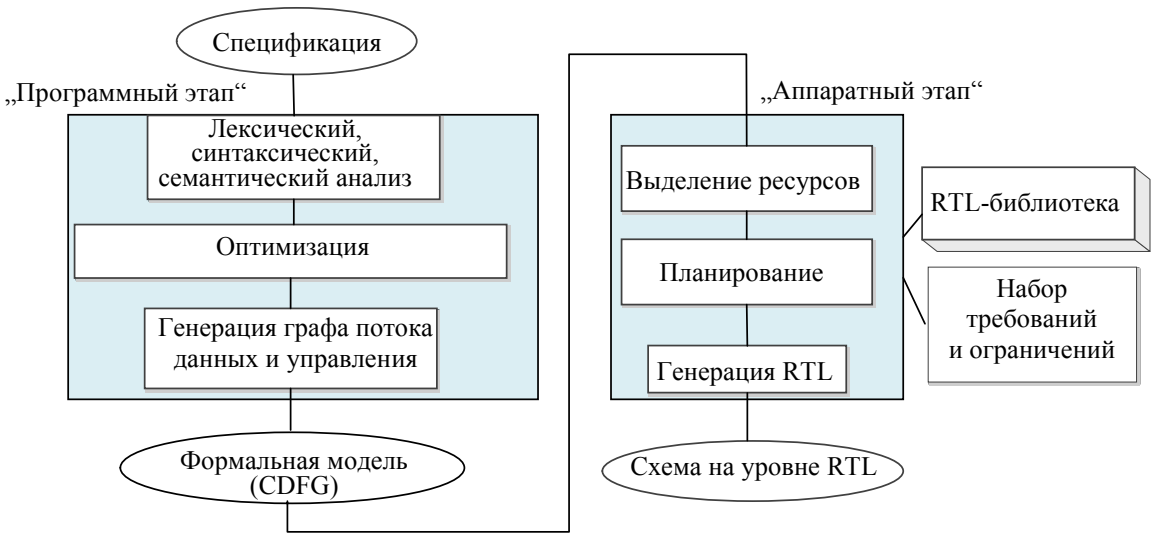


*Рис. 1*

Задачи синтеза, обозначенные как „Программный этап“, практически полностью совпадают с задачами, решаемыми традиционным компилятором. Поэтому большинство коммерческих HLS-систем создано на базе существующих компиляторных инфраструктур, таких как Front-End C/C++ фирмы “Edison Design Group” (США).

Традиционно в качестве аппаратной модели в HLS-системах используется модель конечного автомата с трактом данных (Finite State Machine plus Datapath — FSMD). В такой модели вычисления выполняются в тракте данных, состоящем из арифметических блоков и мультиплексоров, а автомат реализует граф потока управления программой. Альтернативным

подходом является генерация потока управления непосредственно в тракте данных путем введения специализированных аппаратных блоков, реализующих циклы и ветвления. Такой подход позволяет создавать системы с максимальной пропускной способностью, но имеет, однако, и такой недостаток, как большая площадь схемы.

При создании системы высокоуровневого синтеза, описываемой в настоящей статье, был использован второй из представленных подходов, реализованный в виде потоковой вычислительной модели [2]. В рамках этой модели данным присваиваются признаки их достоверности. Наличие таких признаков позволяет организовать циклы и ветвления непосредственно в графе потока данных, а также облегчает процесс реализации вычислительного конвейера.

**Синтез аппаратуры с использованием LLVM-кода.** Промежуточное представление LLVM основано на форме с единственным статическим присваиванием, в этой форме значение каждой переменной присваивается однократно. Аналогичные представления используются во многих компиляторах благодаря возможности проведения корректного анализа и различных преобразований [3]. В основе работы любого компилятора на базе LLVM лежит ряд процедур по оптимизации промежуточного кода, каждая из таких процедур оптимизации называется „проходом“. Синтез аппаратуры реализован как „специальный проход“.

„Синтезирующий проход“ состоит из двух этапов: на первом этапе LLVM-код преобразуется в граф потока данных и управления в выбранной вычислительной модели, на втором этапе осуществляется конвейеризация вычислений в зависимости от заданных на входе системы требований. Рассмотрим каждый из этапов „синтезирующего прохода“.

*Преобразование LLVM-кода в CDFG.* Программа в LLVM-коде представляется в виде графа потока управления, узлы которого, называемые базовыми блоками, содержат линейные участки программы. Для примера рассмотрим следующую функцию на языке С:

```
char demo (unsigned char memory[], char len)
{
   unsigned char max = 0;
   for (char i = 0; i < len ; i++)
     if (memory[i] > mx) mx = memory[i];
   return mx;
}
```

Пропустив такую функцию через парсер Clang (Front-End для языка С из состава LLVM), на его выходе получим LLVM-код, который можно визуально отобразить в виде графа, показанного на рис. 2. Данная функция разбивается на три базовых блока, для каждого из которых показан код на LLVM-ассемблере.

Путем преобразования переменных в дуги графа базовые блоки легко преобразуются в граф потока данных. В качестве примера на рис. 3 показан граф потока данных для тела цикла (операции преобразования типов не показаны). Помимо узлов, в которых осуществляются арифметические операции над данными (сложение, сравнение, выбор большего числа из двух), в графе также присутствуют специальные функциональные узлы: **`phi`**, **`guard`** и **`mem load`**. Узел **`mem load`** является портом к внешней памяти; узлы **`phi`** и **`guard`** служат для инициализации и завершения цикла. Узел **`phi`** используется для инициализации начальных значений переменных: когда на внешний порт поступают данные с признаком достоверности, они пропускаются через узел **`phi`**, после чего внешний порт переходит в закрытое состояние, а узел **`phi`** начинает пропускать данные, поступающие в результате итераций внутреннего цикла. В приведенном примере узлы **`phi`** присваивают итератору цикла **`i`** и переменной **`mx`** начальные значения „нуль“; узел **`guard`** служит для завершения цикла: после вычисления признака завершения он пропускает рассчитанные

данные на внешний порт и переходит в закрытое состояние; узел **cmp** реализует компаратор, а узел **max** осуществляет выбор большего из двух параметров.

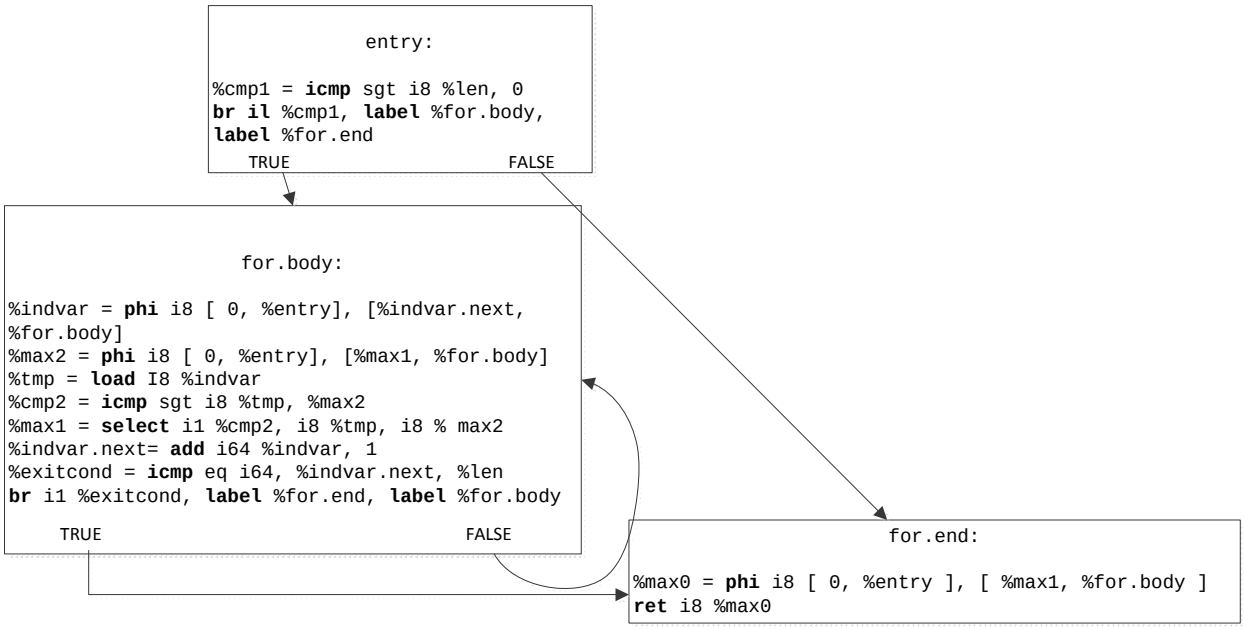


*Рис. 2*

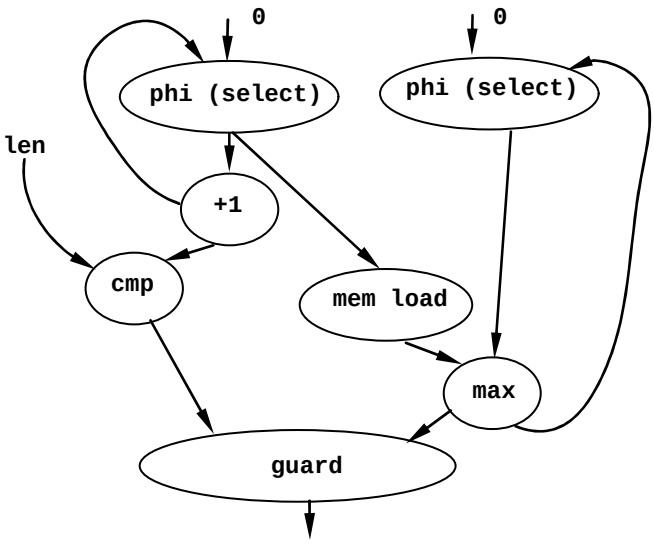


*Рис. 3*

Функциональность узлов **phi** и **guard** обеспечивается наличием внутреннего состояния, т.е. данные узлы содержат внутри себя регистры. Память также является устройством с внутренним состоянием, и чтение через порт памяти занимает как минимум один такт. Все остальные элементы комбинационные.

Во многих схемах задержка прохождения данных через длинную комбинационную цепочку не „укладывается“ в требуемый период тактового сигнала, в этом случае комбинационные схемы разбиваются регистрами на несколько стадий. Такое преобразование называется конвейеризацией. Рассмотрим пример конвейеризации для данной функции.

*Конвейеризация вычислений.* В классической для HLS-систем модели FSMD решается задача планирования вычислений для ограниченного набора ресурсов. В рассматриваемой в данной статье модели ограничение на ресурсы не вводится, поэтому планирование не выполняется, однако решается задача конвейеризации, похожая по содержанию на планирование в

FSMD. Задачей конвейеризации является разбиение вычислений на несколько стадий в зависимости от требований к тактовой частоте и временным параметрам комбинационных схем. Рассмотрим несколько случаев конвейеризации для тела цикла, приведенного на рис. 3.

Предположим, что задержка через любой комбинационный элемент составляет 1 нс, доступ к памяти занимает 1 такт, а целевая частота 200 МГц. Тогда все комбинационные блоки „укладываются“ в один такт и конвейер становится двухстадийным (из-за задержки в памяти и **phi**). Графически такой конвейер изображен на рис. 4, регистры в узлах **phi** показаны в виде прямоугольников.

Если целевую частоту задать равной 600 МГц, то комбинационная цепочка через сумматор **+1** и компаратор **cmp** не будут „укладываться“ в период тактового сигнала. Такую цепочку можно разбить на два такта, тем самым получив трехстадийный конвейер, который выполняет три итерации цикла параллельно. Несмотря на то что комбинационный путь через узел **max** „укладывается“ в 600 МГц, к нему также требуется добавить конвейерный регистр, в противном случае на узел **guard** будут поступать данные, полученные в результате разных итераций цикла, что нарушает семантику исходной программы. Если бы путь через сумматор и компаратор был разбит на пять стадий, то к пути через узел **max** также пришлось бы добавить четыре конвейерных регистра.

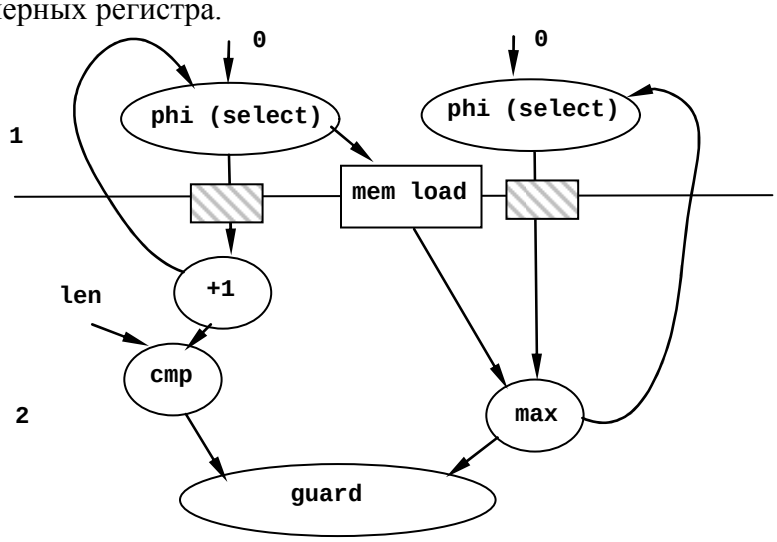


*Рис. 4*

Рассмотрим в заключение еще один пример конвейера для данной функции. Предположим, что доступ к памяти занимает не один такт, как было установлено ранее, а два такта. При этом условии все операции, зависящие по данным от результата чтения из памяти, перемещаются на один такт вперед. Таким образом, узел **max** перемещается со второй стадии на третью.

При реализации рассмотренных принципов конвейеризации можно использовать алгоритм ASAP (As Soon as Possible) — выполнение операций в конвейере при первой возможности. На первом этапе алгоритма осуществляется поиск всех узлов, не зависимых по данным (для приведенного примера это узлы **phi**), затем начинается построение конвейера „сверху-вниз“ по графу потока данных.

**Заключение.** Представленные в статье результаты исследований показывают, как создать простое средство высокоуровневого синтеза на базе готовой компиляторной инфраструктуры. Рассмотрены основные этапы синтеза вплоть до генерации микроархитектуры вычислительного конвейера.

За гранью исследования осталось рассмотрение такой важной темы, как реализация аппаратно-ориентированных оптимизаций. К их числу относятся автоматическое распараллеливание и использование арифметики с нестандартными типами данных. Тем не менее даже

базовые возможности в области высокоуровневого синтеза могут значительно упростить задачу создания и анализа специализированных вычислительных блоков.

## СПИСОК ЛИТЕРАТУРЫ

1. *Gajski D. D., Ramachandran L.* Introduction to high-level synthesis // IEEE Design & Test of Computers. 1994. Vol. 11. P. 44—54.

2. *Попов Р. И.* Применение потоковых вычислительных моделей в проектировании специализированных процессоров // Науч.-техн. вестн. СПбГУ ИТМО. 2011. № 75. С. 77—81.

3. *Lattner C., Adve V.* LLVM: A compilation framework for lifelong program analysis & transformation // Intern. Symp. on Code Generation and Optimization, USA. 2004. March. P. 75.

***Сведения об авторе***

***Роман Игоревич Попов*** — аспирант; Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики, кафедра вычислительной техники; E-mail: ripopov@gmail.com

Рекомендована кафедрой вычислительной техники

Поступила в редакцию 22.05.12 г.