Санкт-Петербургский Государственный Университет

Математико-Механический факультет

Кафедра системного программирования

курсовая работа студента 361 группы

Транслятор микрокода для многоядерного потокового вычислителя

Автор:
Александр Улитин

Научный руководитель:
Б. Н. Кривошеин

24 мая 2012

Содержание

1 Введение

Объемы информационных потоков постоянно увеличиваются, соответственно растут и объемы числовой обработки этих потоков. В повседневной жизни часто возникает необходимость обрабатывать большие потоки информации на небольших устройствах. Например, обработка изображений с видеокамеры, анализ показаний с датчиков радиоуправляемых вертолетов. Эти задачи для хорошо распараллеливаются. Их хорошо можно решать с помощью видеокарты персонального компьютера используя технологию CUDA, но для этого требуется чтобы у нас был компьютер в нашем устройстве. В нашей курсовой работе мы начали создавать архитектуру параллельного вычислителя.

Мы хотим создать архитектуру параллельного вычислителя на базе управляющего процессора и массива простых вычислительных ядер на базе FPGA. На текущем этапе проекта мы уже сформировали общую схему работы, разработали простой вычислительный модуль(DSP-UP) на базе цифрового процессора сигналов DSP48E1, разработали схему соединения простых вычислительных модулей, разработали транслятор микрокоманд для простого вычислительного модуля.

Моей частью этой работы была разработка системы микрокоманд для простого вычислительного модуля, и разработка транслятора для программирования микропамяти.

2 Описание работы `DSP-UP`

В нашей системе DSP-UP является простым вычислительным модулем, которые, в конечном продукте, будут выполнять основную вычислительную работу. Была задача сделать как можно более простой вычислитель, чтобы была возможность разместить их как можно больше на кристалле. Поэтому в качестве «процессора» было выбрано использовать простой цифровой процессор сигналов DSP48E1 (обертка над ним DSP-SHORT, см. рис. на с. 5). Описание модулей вы можете прочитать в курсовых работах [7, 8, 9].

Память нашего вычислительного устройства можно представить как матрицу размером $n \times m$, где m - размер одной команды, n - максимальное количество команд, записанное в микропамять. Каждая команда состоит из нескольких записей, значения которых управляет устройством. Схема одной команды приведена на рис.1. Каждый бит из этой памяти отвечает за работу какого-либо устройства. Например, биты из crtl_d(17-15) управляют мультиплексором, который установлен перед входом D у DSP48E1. Соответственно, установка этого поля влияет на то, какая линия будет использоваться. Например, если установить значение поля в 000, то будет использоваться нулевой вход (все входные биты у порта D будут нули), а если установить в 010, то будет использоваться значение из регистра R2.

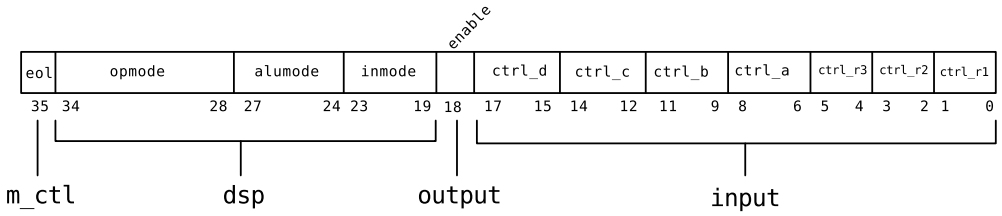

Рис. 1: организация микропамяти

Описание полей

- `eol` – битовое поле, определяющее знак для m_ctl о том, что программа закончилась
- `ctrl_`i, $i \in \{a, b, c, d\}$ – управление входами DSP48E1[2]. В зависимости от значения этого поля на вход DSP будет подаваться одно из следующих значений: значение регистра R1, R2 или R3; выходное значение P у DSP48E; входы I1 или I2.
- `opmode, inmode, alumode` – команды управления вычислительным устройством DSP48E.

- `crtl_r`i, $i \in \{1, 2, 3\}$ – управление регистрами. В зависимости от значения этого поля в регистры могут быть записаны данные из I1,I2 или P. По умолчанию, в регистр ничего не записывается.

- `enable` – битовое поле, которое определяет когда необходимо передать результат вычислений следующему модулю.

Конвейер

В проекте DSP48E1 настроена на конвейерное вычисление. Для повышения производительности было выбрано использовать конвейер с двумя стадиями работы.

Микропрограммисту следует учитывать факт наличия конвейера.

3 Транслятор

УГМ

Для ускорения разработки было решено не разрабатывать транслятор микропамяти самостоятельно, а использовать «Универсальный генератор микрокоманд» (УГМ) Андрея Полиэктова[5]. Благодаря УГМ можно не тратить время на написание однотипных трансляций, а сразу перейти к написанию конфигурации.

УГМ транслирует код с языка конфигурации УГМ в бинарный код, который подходит для занесения в память. Все используемые трансляции однотипны. Текстовый файл, задающий микропрограмму, записываются команды, причем так, что у каждой команды есть атрибуты. Пример:

```
1 %include "settings.mcs"
2 Begin
3 {multab;abcd I1 I1 I1 I1;}
4 {nop;}
5 {nop; output;}
```

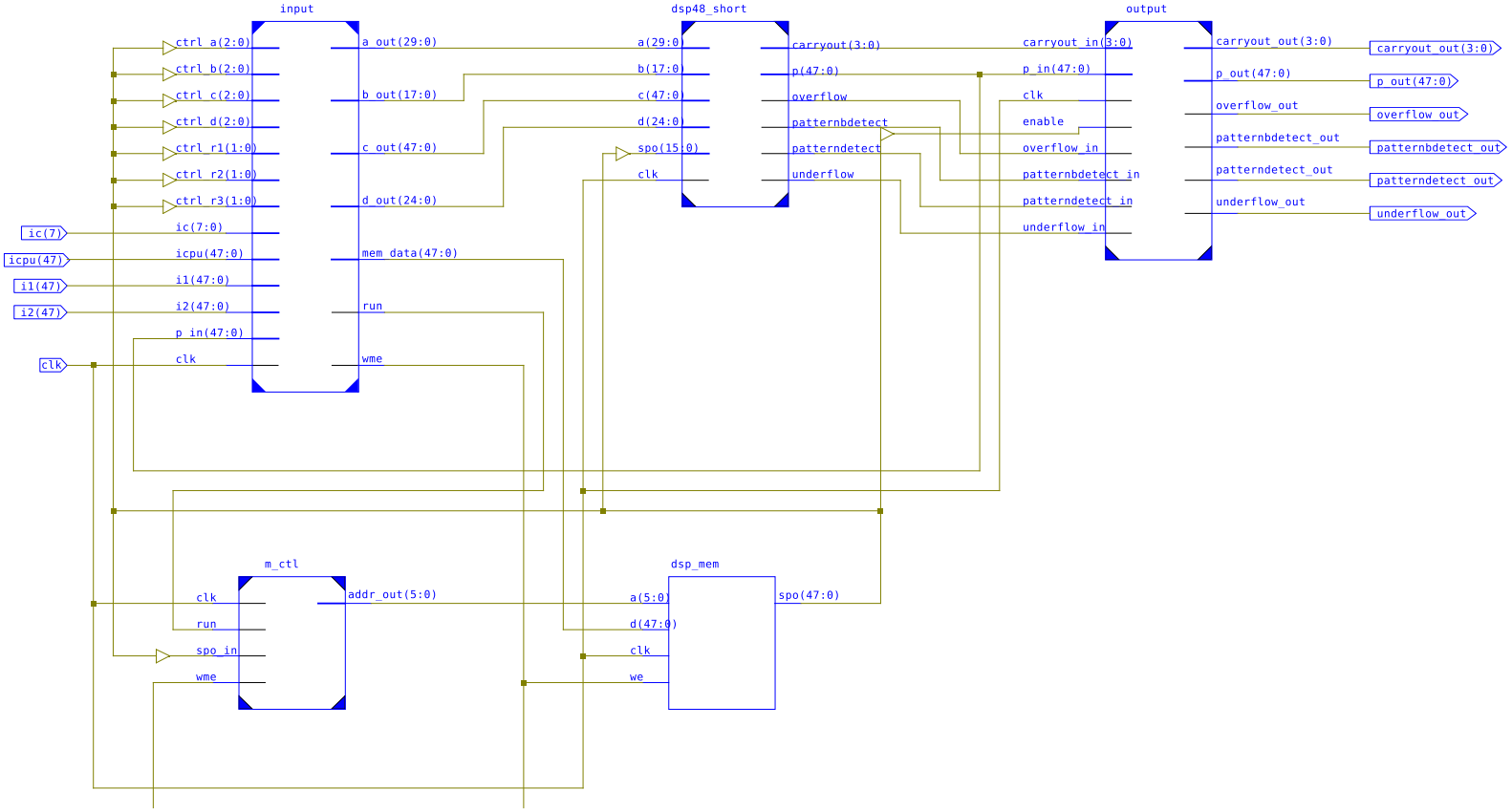

Рис. 2: Общая схема DSP-UP

```
6 {reset;}
7 end.
```

Формальная грамматика, описывающая формат задания программы с учетом написанной конфигурации:

```
1 <Input file > ::= <Include>Begin<Program>End
2 <Include> ::= %include "settings.mcs" \n
3 <program> ::= <Prorgam elem><Program> | <Prorgam elem>
4 <Prorgam elem> ::= <Label> | <number> | <Microcommand>
5 <Microcommand> ::= {<Attributes>}
6 <Attributes> ::= <Attribute>;<Attributes>|
7 <Attribute> ::= <commands with argument>
```

Наша конфигурация

Команды для программирования нашей микропамяти можно условно разделить микрокоманды на три категории:

1. Управление

команда	Выполнимая формула
`multab`	$A[0-25] \cdot B$
`add`	$A : B + C$
`sub`	$A : B - C$
`xorabc`	$A : B \,\texttt{xor}\, C$
`orabc`	$A : B \,\texttt{or}\, C$
`andabc`	$A : B \,\texttt{and}\, C$
...	

2. Выборка данных

команда	описание
`abcd <A> <B> <C> <D>`	позволяет установить входные данные у блока `DSP-SHORT`
`a <A>`	устанавливает вход `A`
`b <B>`	устанавливает вход `B`
`c <C>`	устанавливает вход `C`
`d <D>`	устанавливает вход `D`

3. Команды записи в регистр

команда	описание
WR<i> <F>	записать в регистр i информацию со входа F

4. Разное

команда	описание
nop	ничего не делать[1]
reset	сигнал `m_ctrl` о том, что программа закончила работу
output	передача результата следующему модулю

Бинарные числа УГМ поддерживает установку полей числами записанными в десятичном и шестнадцатеричном формате. Было бы удобно, чтобы можно было задавать числа и в бинарном виде, т. к. в конечном итоге все равно значения устанавливаются отдельным битам. Поэтому был реализован препроцессор, который осуществляет эти простые трансформации

`set "INMODE" to 0b01110;` → `set "INMODE" to 14`

[1]Если было запущено конвеерное вычисление, то оно завершатся. команда nop реализуется тем, что она сохраняет значения входов с предыдущей команды

Выход транслятора

В результате работы УГМ получается файл с адресами, которые преобразуются постпроцессором в формат MIF, пригодный для инициализации памяти в симуляторе.

```
:0x00000x000050776D80      0000...110101000000
:0x00010x000050776D80  →   0000...000000000000
:0x00020x000050776D80      0000...000000000000
```

4 Тестирование

Тестирование кода проводилось с помощью симулятора ISim. Для тестирование мы написали испытательный модуль для симулятора (testbench), котором память инициализируется из сгенерированного MIF файла. Благодаря этому, во время испытания, можно не тратить время для программирование микропамяти через протокол IC/ICPU[7], а сразу проводить испытания программы.

5 Заключение

В ходе выполнения данной курсовой работы были достигнуты следующие результаты:

- Получен транслятор микрокода для простого вычислительного ядра.
- Получен опыт использования технологий
  - VHDL
  - Xilinx ISE
  - ISim

Дальнейшая работа

Наш проект не завершен. Полученный транслятор слишком низкоуровневый и не подходит для написания прикладных приложений. В дальнейшем, планируется использование этого транслятора для написания более высокоуровнего языка.

Список литературы

[1] Xilinx. *Distributed Memory v7.1 Product Specification*, ds230 edition, 2005.

[2] Xilinx. *Virtex-6 FPGA DSP48E1 Slice User Guide*, ug369 edition, 2011.

[3] П. Н. Бибило. *Основы языка VHDL.* Солон-Р, 2002.

[4] Р. И. Грушвицкий Е. П. Угрюмов, А. Х. Мурсаев. *Проектирование систем на микросхемах программируемой логики.* БХВ -Петербург, 2002.

[5] А. Ю. Полиэктов. Универсальный генератор микрокоманд. диплом, СПбГУ Математико-Механический факультет, 1998.

[6] А. К. Поляков. *Языки VHDL и VERILOG в проектировании цифровой аппаратуры.* СОЛОН-пресс, 2003.

[7] Д. В. Солдатов. Реализация модулей ввода/вывода ПВП в связке с ядром dsp48e в рамках проекта МПВ. курсовая работа, СПбГУ Математико-Механический факультет, 2012.

[8] Е. А. Тодорук. Разработка модуля памяти для многоядерного потокового вычислителя. курсовая работа, СПбГУ Математико-Механический факультет, 2012.

[9] Д. Ю. Забранский. Разработка сети простых вычислительных процессоров и фильтра в рамках студенческого проекта МПВ. курсовая работа, СПбГУ Математико-Механический факультет, 2012.