САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Солдатов Дмитрий Владимирович

Разработка программно-аппаратной архитектуры многоядерного потокового процессора на ПЛИС

Бакалаврская работа

Допущена к защите.

Зав. кафедрой:

д. ф.-м. н., профессор А. Н. Терехов

Научный руководитель:

ст. преп. Б. Н. Кривошеин

Рецензент:

к. ф.-м. н., доцент Н. Ф. Фоминых

Санкт-Петербург
2013

SAINT-PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Department of Software Engineering

Dmitry Soldatov

Development of the hardware and software architecture for multicore stream processor on FPGA

Bachelor's Thesis

Admitted for defence.
Head of the chair:
professor A.N. Terekhov

Scientific supervisor:
senior teacher B.N. Krivoshein

Reviewer:
senior teacher, Dr. N.F. Fominykh

Saint-Petersburg
2013

Оглавление

Введение

Современные тенденции развития технологий ведут к увеличению объемов обрабатываемой информации, и как следствие повышаются требования к скорости вычислительных систем. Как одно из решений повышения производительности выступает распараллеливание алгоритмов обработки данных. В качестве вычислительных систем для параллельных вычислений обычно используются либо многоядерные процессоры, либо программируемые логические интегральные схемы — ПЛИС.

ПЛИС — это электронные компоненты, предназначенные для дальнейшего создания цифровых устройств. В отличие от обычных интегральных схем, логика работы ПЛИС не определена при изготовлении, она задается в дальнейшем, посредством специальных языков программирования — языков описания аппаратуры, и так же может быть изменена.

FPGA — это разновидность ПЛИС, основанная на том понятии, что любую логическую функцию можно представить в виде таблицы истинности. Т.е. на любое входное воздействие изначально известно и прописано в особой памяти выходное значение функции. Такая особая память называется LUT ячейкой (Look-up table). В силу возможности реализации на FPGA алгоритмов параллельного вычисления, эти устройства широко используются в цифровой обработке сигналов, в криптографии, в компьютерном зрении и других областях, где требуются быстрая обработка больших объемов данных и возможность обработки данных в режиме реального времени.

Как уже было сказано, логика работы ПЛИС описывается посредством языков описания аппаратуры, в данной работе используется язык VHDL. VHDL был разработан в 1983 году, по заказу Министерства Обороны США, в настоящее время широко используется при проектировании электронных устройств для описания цифровых и аналогово-цифровых систем. Так же VHDL может быть использован как язык параллельного программирования общего назначения.

Осенью 2011 года стартовал студенческий проект, задачей которого стояла разработка многоядерного потокового процессора (МПП), ориентированного на потоковую обработку данных для встроенных применений — это важно, поскольку реализуемое устройство работает без использования посторонних вычислительных устройств, таких, например, как персональный компьютер. Архитектура параллельного вычислителя создавалась на базе управляющего процессора и массива простых вычислительных ядер на базе FPGA. В ходе проекта был реализован модуль вычислительного ядра и несколько таких ядер были объединены в простую статическую сеть.

Следующий этап работы заключался в подготовке всей системы к внедрению динамической коммутации между вычислительными ядрами и возможностью легкого увеличения их количества. Создание и развитие архитектуры многопоточного процессора и является целью моей работы.

1. Постановка задачи

Изначально в многоядерном поточном вычислителе связи между вычислительными ядрами задавались статически, это значит, что направление потоков данных внутри вычислительной сети невозможно было изменить. Такая архитектура ограничивает возможность реализации большого количества алгоритмов, которые можно адаптировать под МПП. Один из способов повысить гибкость МПП — это добавить возможность на этапе программирования задавать связи между вычислительными ядрами.

Целью данной работы является создание архитектуры устройства и расширение возможностей реализации различных алгоритмов для нее. Таким образом, в мои задачи входит:

- Реализовать архитектуру вычислительного ядра
- Обеспечить МПП возможностью задавать связи между вычислительными ядрами
- Оценить эффективность архитектуры МПП

2. Предметная область

2.1. VHDL

[6, p. 314] VHDL обычно используется для написания модели, описывающую логическую схему. Не всякая модель может быть синтезируема в рабочее устройство. Несинтезируемые модели используются для проверки устройства в режиме симуляции, имитируя потоки входных данных. Такие модели обычно называются тестбенч.

Самое главное отличие VHDL от прочих языков программирования в том, что он описывает параллельные процессы. Например, если код на C выполняется по очереди, команда за командой, и чтобы обработать разные куски кода одновременно, нужно работать с таймерами и прерываниями, то на VHDL разные блоки программы выполняются параллельно друг другу, но в тоже время в VHDL есть часть команд, которые выполняются последовательно. Поэтому структура программы в корне отличается от привычной.

Основные структуры данных VHDL:

- константы;
- переменные;
- сигналы.

Константы и переменные имеют практически тот же смысл, что и в других языках программирования.

Сигналы похожи на переменные, но физически имеют смысл проводников на печатной плате. Это значит, что сигнал всегда имеет некоторое значение.

Конечным смыслом является то, что модель VHDL транслируется в "шлюзы и провода", которые могут быть загружены на ПЛИС, такие как CPLD или FPGA, что в итоге реализует аппаратное устройство.

2.1.1. Используемые библиотеки VHDL

std_logic_1164

Эта библиотека определяет базовый тип `std_logic` и несколько функций по работе с ним. `std_logic` представляет один бит данных. Но в отличие от типа `bit`, который принимает только два значение 1 и 0, `std_logic` может принимать 9 состояний, что больше соответствует физическим свойствам сигнала и расширяет возможности оперирования сигналами и переменными данного типа. Здесь описываются 4 наиболее важных состояния:

- 'U': неинициализированный (uninitialized). Сигнал еще не задан.
- 'X': неизвестный (unknown). Невозможно определить значения данного сигнала. Такое случается, когда в один порт приходит сразу несколько сигналов с разными значениями.
- '0': логический ноль.
- '1': логическая единица.

В библиотеке std_logic_1164 определен еще один важный тип — `std_logic_vector`. Он используется для задания массивов типа `std_logic`. Часто сигнал такого типа называется шиной, например, запись

```
s1: std_logic_vector(3 downto 0);
```

говорит о том, что определена шина s1 шириной 4 бита.

numeric_std[1]

В случаях, когда над сигналами нужно проводить арифметические операции, `std_logic_vector` удобно представлять в виде десятичного числа. За конвертацию типов и предоставление арифметических функций отвечает библиотека `numeric_std`.

2.2. ISE

[3] ISE – программный инструмент фирмы Xilinx для создания и анализа моделей устройств, написанных на языках описания аппаратуры. Для анализа правильности логики разрабатываемого устройства используется встроенный продукт ISim, в котором эмулируется работа устройства. Подобный анализ является важным этапом, упрощающим разработку устройства и уменьшающим трудозатратную фазу конечного тестирования непосредственно на ПЛИС.

Xilinx CORE Generator упрощает разработку устройства, обеспечивая возможность получить готовый модуль с помощью параметризации шаблона конкретного модуля для ПЛИС. CORE Generator предоставляет каталог шаблонов модулей, таких как память и FIFO-буферы, цифровые сигнальные процессоры (DSP) и т.д.

2.3. Virtex 6

FPGA семейства Virtex6 — это высокопроизводительные платформы фирмы Xilinx для создания различных электронных устройств.

2.3.1. DSP48E1

DSP48E1 — это шаблон модуля, предоставленный инструментом Xilinx CORE Generator, реализующий цифровой сигнальный процессор (ЦСП, DSP), подходящий для определенных FPGA, в том числе и Virtex 6.

DSP как электронный компонент — это микропроцессор, специализированный на обработку цифровых сигналов. DSP имеет Гарвардскую архитектуру и характеризуется быстрым выполнением сложных математических конструкций, например, в этом процессоре умножение с накоплением выполняется за один такт (умножение с накоплением распространенная операция в области обработки сигналов, например, используется при реализации алгоритма перемножения матриц).

DSP лежит в основе вычислительного ядра, поэтому важно иметь представление о его работе. В упрощенной интерпретации, DSP имеет 4 входных порта данных — A, B, C, D; 3 порта управляющих инструкций — INMODE, ALUMODE, OPMODE; и выходной порт результата P. INMODE отвечает за пре-сумматорный блок, то есть при определенных инструкциях на выходе блока могут получится результаты типа `D+A, -A, D` и т.д.

Инструкция OPMODE программирует блоки `X, Y, Z`. Посредством OPMODE определяется участие результата умножителя или предыдущего результата полученного на выходе DSP.

Инструкция ALUMODE определяет конечную линейную комбинацию блоков `X, Y, Z: Z + X + Y; Z - (X + Y)` и т.д.

Для более детального описания DSP48E1 следует обратиться к официальной документации [2].

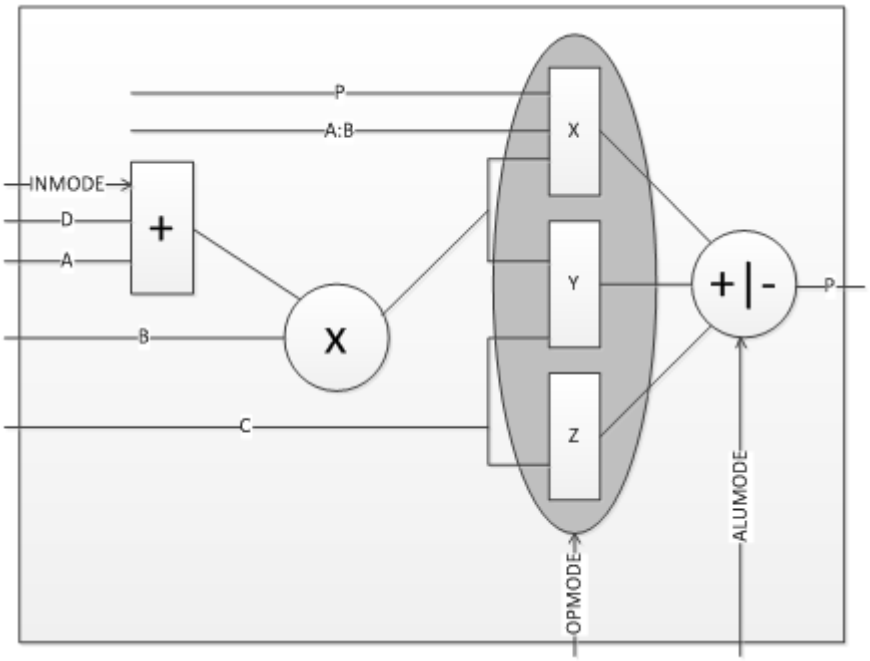

Рис. 1: Модуль DSP

2.3.2. Distributed memory

В проекте так же используется шаблон модуля Distributed Single-Port RAM[4], как внутренняя память вычислительного ядра. В данном проекте используется память шириной 48 бит и глубиной в 64 записи. Ширина памяти обуславливается шириной портов данных DSP, о чем будет рассказано в главе "Архитектура". Интерфейс памяти описывается следующими портами:

Принцип работы памяти: если требуется записать данные в память, по порту a шлется адрес, по порту d - данные, сигнал порта we равен 1. При считывании данных из памяти порт d не принимает участия, сигнал порта we равен 0.

Name	Direction	Bit width	Description
a	In	6	Адресная шина памяти
d	In	48	Шина входных данных
we	In	1	Сигнал по которому определяется, пишутся ли данные в память или происходит считывание.
spo	Out	48	Шина выходных данных

2.4. Студенческий проект

Помимо разработки архитектуры многоядерного поточного процессора, важной частью проекта является адаптация алгоритмов под сам процессор. Изучение алгоритмов, поддающиеся распараллеливанию на архитектуре МПП, реализация и оптимизация времени исполнения алгоритмов для МПП, реализация алгоритмов на Си для Intel Core i7, тестирование, анализ результатов и оценка эффективности работы алгоритмов описана в работе Тодорука Е.А. ”Разработка и оптимизация времени исполнения алгоритмов обработки данных для многоядерного потокового процессора на ПЛИС”[5].

3. Архитектура

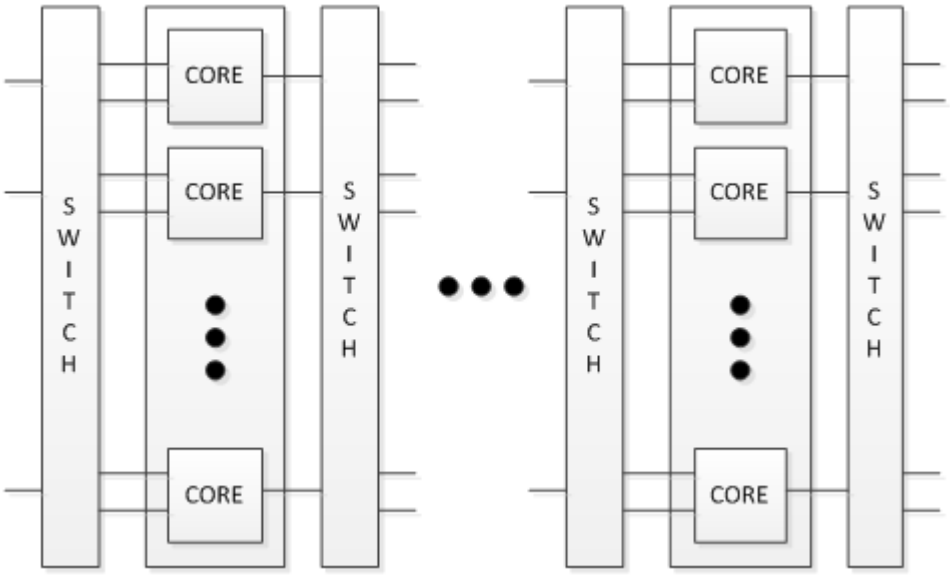

Рис. 2: Многоядерный потоковый вычислитель

Архитектура МПП имеет вид квадратной матрицы 16 на 16, элементами которой являются вычислительные ядра (core). Вычислительные ядра по 16 элементов объединены в модули, называемые слоями (layer). Слои объединены последовательно между собой посредством коммутаторов (switch). Рассмотрим подробно архитектуру каждого модуля.

3.1. Слой (layer)

Данный модуль не содержит вычислительной логики, является оберткой для группы вычислительных ядер. Каждый слой МПП обладает адресом, необходимым при программировании МПП и параметром, отвечающим за количество ядер в слое. Таким образом, можно легко изменить количество вычислительных ядер, поменяв значение этого параметра в файле конфигураций.

3.2. Вычислительное ядро (core)

Вычислительное ядро — это микропроцессор, предназначенный для выполнения арифметических операций над числами. Ядро обладает внутренней памятью, что позволяет хранить в нем целый набор инструкций. Таким образом, на каждом такте ядро может производить различные операции над различными входными данными, не получая инструкции из внешней памяти, что существенно сокращает количество обращений к ней.

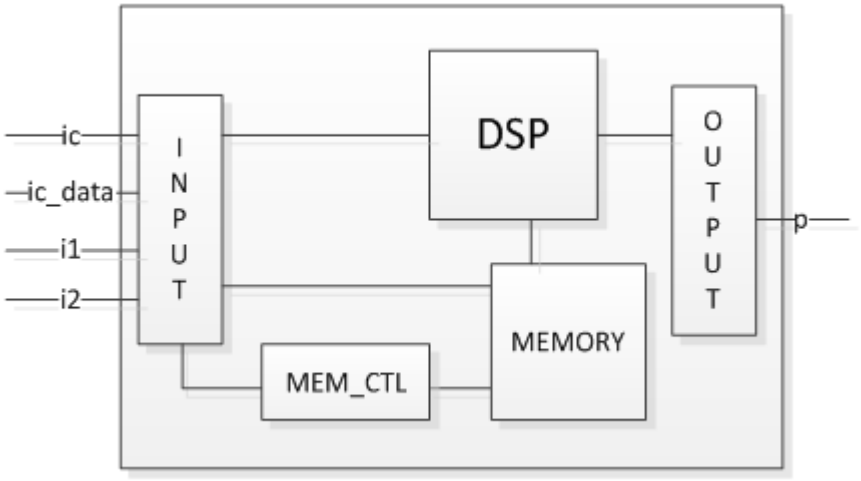

Рис. 3: Вычислительное ядро

На рисунке изображена текущая архитектура вычислительного ядра. Как видно, здесь есть модуль памяти (memory) и контроллер памяти (mem_ctl) о которых было упомянуто ранее. С внешней средой вычислительное ядро взаимодействует посредством модулей input и output. За вычислительные действия отвечает модуль DSP.

Работу вычислительного ядра можно описать в виде конечного автомата. В каждый момент времени ядро находится в одном из состояний, которое определяется посредством входных сигналов. Состояния мож-

но разбить на две группы: состояние программирования ядра — это wr1, wr2, wr3, wm. И состояние работы ядра — bp, processing, paused.

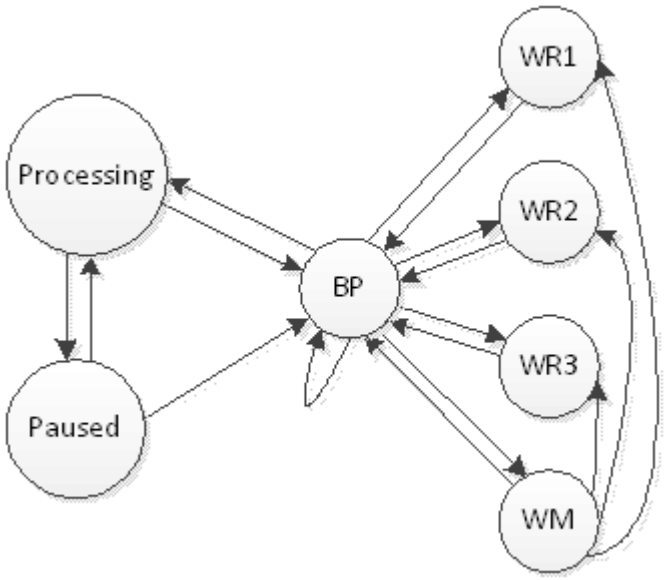

Рис. 4: Автомат состояний вычислительного ядра

- BP (bypass) — состояние по умолчанию и начальное состояние вычислительного ядра. Находясь в этом состоянии, ядро не производит никаких вычислений, передавая полученные данные следующему ядру без изменений.
- WR1, WR2, WR3, WM (write register, write memory) — состояние записи данных в регистры и в память.
- Processing — состояние вычисления. Находясь в этом состоянии, посредством инструкции из внутренней памяти, ядро на каждом такте определяет вычислительную операцию и набор данных, над которыми эта операция будет производиться, и выдает результат.
- Paused — работа ядра приостановлена.

3.2.1. Модуль input

За определение состояния ядра отвечает модуль input. Ввод вычислительного ядра централизован, то есть один модуль отвечает за принятие данных из внешней среды, а уж через этот модуль данные передаются внутрь вычислительного ядра.

Таблица 1: Порты модуля input

Name	Direction	Bit width	Description
clk	In	1	Сигнал тактовой частоты
i1, i2	In	48	Порты внешних данных
p_in	In	48	Внутренние данные, полученные с выхода этого же ядра
ic	In	8	Управляющая команда ядра
ic_data	In	48	Данные, сопровождающие управляющую команду ic
r1, r2, r3	Register	48	Внутренние регистры ядра
ctrl_r1, ctrl_r2, ctrl_r3	In	2	Контроллеры внутренних регистров R1, R2, R3
ctrl_a, ctrl_b, ctrl_c, ctrl_d	In	3	Контроллеры портов A_OUT, B_OUT, C_OUT, D_OUT
run	Out	1	Сигнал запуска работы ядра
wme	Out	1	Сигнал начала записи во внутреннюю память
mem_data	Out	48	Данные внутренней микрокоманды ядра
a_out, b_out, c_out, d_out	Out	30, 18, 48, 25	Данные для модуля DSP

Внутренние регистры

В модуле Input находятся регистры для хранения дынных 3 x 48 бит[1], поскольку непосредственно отсюда данные передаются на вход в DSP. Внутренние регистры могут быть нужны для констант, необходимых во время вычислений.

Записать данные в регистры можно посредством портов ic и ic_data, о чем подробно рассказано в пункте "Программирование ядер". Так же, данные в регистр могут быть записаны на каждом такте посредством инструкции из внутренней памяти. Ниже предоставлена таблица команд управления регистрами (аналогично для всех регистров).

Код команды	Источник данных
01	P_IN
10	I1
11	I2

Контроль портов DSP

В этом же модуле обеспечивается контроль портов, отвечающих за входные данные модуля DSP (выходные порты A_OUT, B_OUT, C_OUT, D_OUT модуля Input соединены соответственно со входными портами модуля DSP A,B,C,D).

Какими способами мы можем передать данные в эти порты?

- Внешние порты (I1, I2), через которые идет поток данных из внешней памяти

[1]макс. размерность входного порта DSP

- Порт P_IN - внутренний сигнал для зацикливания результата вычисления ядра ему же на вход
- Внутренние регистры (R1, R2, R3)

Контроль над выбором источника данных задается посредством мультиплексоров внутри модуля Input, которые в свою очередь, на каждом такте определяют свой выбор с помощью инструкций из внутренней памяти.

Программирование ядер

Для того чтобы производить какие-то вычисления, прежде всего, нужно запрограммировать каждое ядро вычислительной сети. Это делается посредством портов ic и ic_data.

Какие команды существуют, можно видеть ниже.

Код IC-команды	Наименование IC-команды	Описание IC-команды
000	bp	Перейти в состояние bypass.
001	wr1	Запись в регистр R1
010	wr2	Запись в регистр R2
011	wr3	Запись в регистр R3
100	wm	Запись инструкций в память
101	processing	Начать обход внутренней памяти
110	paused	Остановить обход внутренней памяти

Вместе с каждой командой ic в вычислительное ядро поступают данные по порту ic_data. Например, если число 2 необходимо записать в регистр под номером 3, то сигналы, подаваемые на вход модулю будут:

```
ic <= "011";
ic_data <= "10";
```

В случае, когда ядро переходит в состояние вычисления - данные берутся из портов i1, i2, порт ic_data игнорируется.

3.2.2. Модуль memory controller

При реализации контроллера памяти, было решено не передавать через него данные, посылаемые памяти. Данные, а также флаг записи идут в память напрямую из модуля ввода. Единственная задача контроллера выдавать адреса ячеек памяти. В начальном состоянии контроллер посылает адрес 0x00. Как только он получает сигнал записи в память или сигнал старта программы, контроллер начинает инкрементировать адрес до тех пор, пока не пропадет сигнал записи или не появится сигнал конца программы.

Порты модуля memory controller

Name	Direction	Bit width	Description
clk	In	1	Сигнал тактовой частоты
run	In	1	Флаг старта выполнения программы. При появлении этого сигнала память начнет выдавать команды для ядра
spo_in	In	1	Флаг конца программы. Этот сигнал говорит контроллеру памяти о том, что программа была выполнена. При его появлении контроллер начнет выдавать адреса с начала (с адреса 0x00), то есть программа в памяти зацикливается
wme	In	1	Флаг записи программы в память, 1 бит. При появлении этого сигнала контроллер будет выдавать адреса для записи в память начиная с адреса 0x00 и инкрементировать адрес на каждом такте
addr_out	Out	6	Данные, сопровождающие команду ic

3.2.3. Модуль memory

Модуль memory сгенерирован с помощью инструмента Xilinx CORE Generator, о принципе работы данного модуля подробно рассказано в

главе ”Предметная область”.

Как уже было сказано, вычислительное ядро, находясь в состоянии вычисления (processing), на каждом такте берет инструкцию из внутренней памяти, по которой определяется, какие данные будут переданы на вход DSP, какие операции будут производиться над выбранными данными и какие данные будут записаны в регистры модуля Input. В этом разделе будет рассказано о структуре инструкций, которые хранятся во внутренней памяти вычислительного ядра.

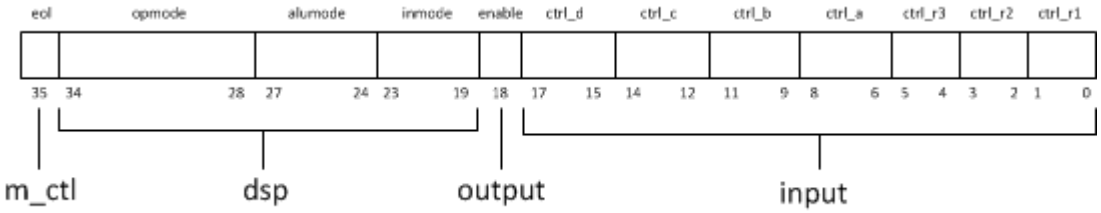

Рис. 5: Инструкция внутренней памяти

- eol — флаг конца программы. Сигнал идет на контроллер памяти, оповещая его, что программа выполнена. При получении этого сигнала, контроллер памяти начинает заново генерировать адреса с 0x00.

- opmode, alumode, inmode — параметры для модуля DSP. Этими командами задаются вычислительные операции.

- enable — включение вывода для модуля output. Этот сигнал подается, когда вычислительное ядро готово выдавать результаты вычислений.

- ctrl_d, ctrl_c, ctrl_b, ctrl_a, ctrl_r1, ctrl_r2, ctrl_r3 — параметры для

модуля input. Используются для конфигурирования мультиплексоров.

3.2.4. Модуль DSP

Модуль DSP реализован с помощью шаблона dsp48e1 и отвечает за арифметические операции над входными данными вычислительного ядра. Подробно про этот модуль рассказано в главе "Предметная область".

3.2.5. Модуль output

Предназначен для вывода результатов из вычислительного ядра. Из этого же модуля результаты передаются на вход этому же вычислительному ядру (порт P_IN в модуле Input), это может быть полезным, когда над результатом нужно произвести повторно такие же вычисления.

3.3. Коммутатор ядер (switch)

Каждый коммутатор вычислительной сети обладает уникальным адресом, соответствующим адресу впереди стоящего слоя. В задачи коммутатора входит задание связей между ядрами в смежных слоях, передача вычисляемых значений с одного слоя на другой и распределение командных инструкций вычислительных ядер на этапе программирования.

Порты коммутатора

Name	Direction	Bit width	Description
clk	In	1	Сигнал тактовой частоты
layers_mask	In	16	Маска слоев
cores_mask	In	16	Маска ядер
pkg_type	In	2	Сигнал, определяющий тип программирующего пакета
pkg_in	In	24	Содержание программирующего пакета
data_ins	In	48x16	Входные порты данных
data_outs	Out	48x32	Выходные порты данных
ic_outs	Out	8x16	Порты управляющих команд ядру
ic_data_outs	Out	48x16	Порты данных, сопровождающих управляющие команды ic_outs

На этапе программирования МПП коммутаторам посылаются программирующие пакеты. Коммутаторы в свою очередь, в зависимости от типа пакета, разбирают его по разному. Существует два типа пакетов:

- Пакет коммутатору (pkg_type = ”01”) содержит номера входного и выходного порта коммутатора, которые нужно связать.
- Пакет ядру (pkg_type = ”10”) содержит управляющую команду ядра (ic) и сопутствующие данные (ic_data), которые передаются на выходные порты ic_outs и id_data_outs соответственно. **Важно:** на этапе программирования ядра, основными рабочими портами

являются `ic` и `ic_data`. На этапе работы – `i1, i2`, по которым идет поток входных данных, и выходной порт `p`. Именно по причине того, что на этапе работы в вычислительном ядре участвует два входных порта и один выходной, в коммутаторе входных портов в два раза меньше, чем выходных.

Адрес пакета задается битовыми масками по слоям и по ядрам, что позволяет запрограммировать несколько вычислительных ядер за один такт. Например, характерным первым шагом программирования МПП является инициализация всех ядер состоянием bypass, что можно сделать, послав соответствующий пакет с масками layers_mask = (others => ”1”) и cores_mask = (others => ”1”) - то есть всем ядрам во всех слоях. По такому же принципу можно задавать одинаковые связи входных и выходных портов в разных коммутаторах (в этом случае используется порт layers_mask).

Связи между ядрами задаются специальной матрицей, которая программируется посредством команд коммутатору. Конечный результат связи входных и выходных портов коммутатора можно определить как многозначную функцию, то есть одному входному порту могут соответствовать несколько выходных портов.

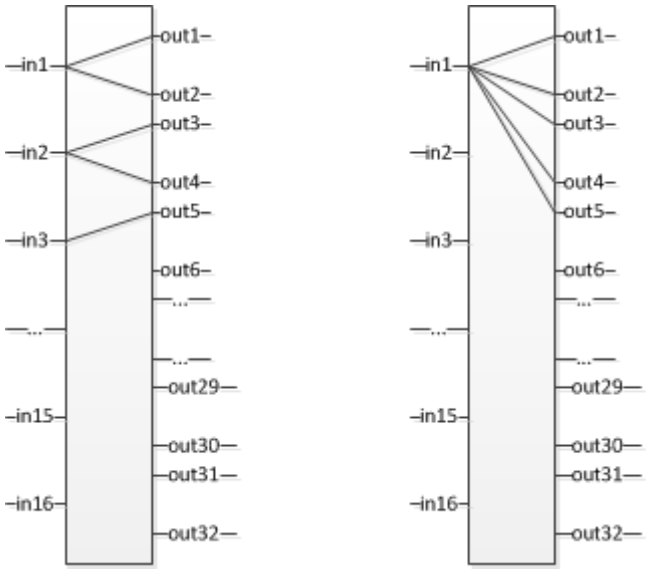

Рис. 6: Примеры связей внутри коммутатора

4. Тестирование

Тестирование МПП проводилось в инструменте ISim с помощью тестовых модулей. Тестовый модуль — это модуль программы, который генерирует сигналы, подающиеся на вход тестируемому модулю. Поскольку каждый модуль имеет четкую, ограниченую логику работы, предоставляется вполне возможным сделать полное тестовое покрытие функциональности МПП в целом. При логическом синтезе известна частота выполнения операций для полученной структуры, поэтому можно уже на этапе моделирования оценить его производительность для различных алгоритмов, задав в тестовом модуле соответствующий алгоритм генерации исходных данных. В результате синтеза тактовая частота МПП получилась 467MHz. В таблице представлены результаты работы устройства, полученные в работе Тодорука Е.А. "Разработка и оптимизация времени исполнения алгоритмов обработки данных для многоядерного потокового процессора на ПЛИС"[5].

Алгоритм	Поток (МБ)	Ширина потока (Байт)	Время(с)	
			Intel Core i7	МПП
Фильтр Лапласа	98	25	4,7	0,009
Свёртка	137	100	1,715	0,0022
БПФ	65	64	1,526	0,0092

5. Выводы

В ходе тестирования алгоритмов на МПП было отмечено, что удобно было бы расширить внутреннюю память вычислительных ядер для хранения констант алгоритмов и промежуточных результатов вычислений. В реализованной архитектуре каждое ядро имеет три регистра для подобных процедур, что для некоторых алгоритмов недостаточно.

При логическом синтезе проекта использовалась конфигурация Virtex-6 с уровнем скорости -1. Согласно спецификации, в таком случае частота DSP48E1 равна 450MHz. Исследования показали, что добавляемая логика других модулей не ухудшает производительности МПП. Более того, при увеличении количества вычислительных ядер достигается линейный рост производительности, поскольку ядра в слое между собой не связаны. Увеличение количества реально используемых слоев, т.е. тех, в которых хотя бы одно вычислительное ядро иногда принимает значение не bypass, приводит к задержке поступления результатов в начале работы устройства, обусловленной так называемой "глубиной конвейера" — ожиданием, пока результаты пройдут все слои МПП. Задержка имеет постоянное значение (добавление слоя приводит к увеличению задержки на 2нс), поэтому при обработке больших объемов данных она незначительна. Таким образом масштаб системы ограничен только ресурсами используемой платы FPGA и тактовой частотой системы, в которую встраивается МПП.

Подводя итог, можно сказать, что использование данной архитектуры целесообразно для встроенных применений и исполнения алгоритмов потоковой обработки данных.

Заключение

Главной задачей данной работы была реализация архитектуры вычислительного ядра и модуля динамической коммутации ядер. Она успешна выполнена.

- Протестирована логика работы каждого модуля и устройства в целом
- Получена синтезируемая архитектура многоядерного поточного вычислителя
- Получены оценки эффективности архитектуры МПП для потоковых алгоритмов обработки данных

Список литературы

[1] VHDL Packages: numeric_std, std_logic_arith. — URL: http://www.ece.msstate.edu/~reese/EE8993/lectures/numeric_standard.pdf.

[2] Xilinx. Virtex-6 FPGA DSP48E1 Slice User Guide. — 2011. — URL: http://www.xilinx.com/support/documentation/user_guides/ug369.pdf.

[3] Xilinx. ISE In-Depth Tutorial. — 2012. — URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_3/ise_tutorial_ug695.pdf.

[4] Xilinx. LogiCORE IP Distributed Memory Generator v7.2 Product Guide. — 2012. — URL: http://www.xilinx.com/support/documentation/ip_documentation/dist_mem_gen/v7_2/pg063-dist-mem-gen.pdf.

[5] Е.А. Тодорук. Разработка и оптимизация времени исполнения алгоритмов обработки данных для многоядерного потокового процессора на ПЛИС. — 2013.

[6] Р.И. Грушвицкий. Проектирование систем на микросхемах программируемой логики. — 2002.