САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
Математико-механический факультет
Кафедра Системного Программирования

Репин Дмитрий Юрьевич

Библиотека Memory Window для языка HaSCoL

Курсовая работа

Научный руководитель:
Медведев Олег

Санкт-Петербург
2014

Оглавление

Введение

Есть очень много задач(и появляется еще больше) которые надо решать с использованием FPGA или “гибких кристаллов”. Но до недавнего времени делать это было достаточно неудобно, для их программирования были доступны только языки низкого уровня: verilog, vhdl. Программировать на них не лучше чем программировать на ассемблере - долго и муторно. И поэтому в последнее время начали появляться альтернативы, например Hascol.

Язык Hascol предназначен для программирования “гибких кристаллов” он более высокоуровневый чем существующие аналоги, но у него есть один серьезный недостаток: на нем почти ничего не написано. Одной из целей моей курсовой работы как раз и будет увеличение кода, написанного на Hascol’e. Естественно что писать что-то, что потом никто не будет использовать не имеет смысла. Чтобы понять что же позже будет использоваться стоит посмотреть что вообще программируют на гибких кристаллах. Часто это работа с графикой: масштабирование изображения (например на различных мониторах), графические фильтры, распознавание образов.

Рабочая среда

Сначала стоит сказать о той среде, окружении, том мире, в котором существует мой проект. В этой части я коснусь как физического плана - программируемых вентильных матриц, так и логического - языка программирования высокого уровня HaSCoL.

FPGA

Программируемая пользователем вентильная матрица (ППВМ, англ. Field-Programmable Gate Array, FPGA) — полупроводниковое устройство, которое пользователь может программировать даже после изготовления; отсюда название: «программируемая пользователем». Далее в тексте я буду использовать сокращение FPGA, потому что, на мой взгляд, русское сокращение выглядит ужасно и плохо читается. FPGA программируются с помощью исходного кода на языке проектирования (типа VHDL), на котором можно описать эту логику работы микросхемы. Также, стоит отметить, что FPGA является одной из архитектурных разновидностей программируемых логических интегральных схем (ПЛИС).

FPGA могут быть перепрограммированы практически в любой момент в процессе их использования. Они состоят из конфигурируемых логических блоков, подобных переключателям с множеством входов и одним выходом (логические вентили или gates). В цифровых схемах такие переключатели реализуют базовые двоичные операции AND, NAND, OR, NOR и XOR. Тогда как в большинстве современных микропроцессоров функции логических блоков фиксированы и не могут модифицироваться. В этом и состоит принципиальное отличие FPGA: и функции блоков, и конфигурация соединений между ними могут меняться с помощью специальных сигналов, посылаемых схеме. В некоторых специализированных интегральных схемах (ASIC) используются логические матрицы, аналогичные FPGA по структуре, однако они конфигурируются один раз в процессе производства. Однако, такая

гибкость, очевидно, не бесплатна и требует существенного увеличения количества транзисторов микросхемы.

Применение FPGA:

- Малосерийные дорогие сетевые маршрутизаторы – сетевой поток слишком быстрый, чтобы обрабатывать на CPU, но серия слишком мала, чтобы создавать специальный чип
- Пример – «some cards in Cisco routers», по данным cisco
- Действительно, имея 100-1000 параллельно работающих модулей памяти, распараллеливать обработку пакетов из разных потоков кажется неплохой идеей
- Pci-express карточка от fiberblaze для фильтрации сетевого потока 4x10ГБит/с по типам протоколов, ключевым словам и т. д.
- Аналогичная от nallatech («pcie-180»)
- Статья от Xilinx research про DSL для описания сетевых фильтров, которые потом работают до 400ГБит/с на high-end virtex7
- Financial High Frequency Trading (low latency application)
- COPACOBANA - *the Cost-Optimized Parallel COde Breaker*

HaSCoL

HaSCoL (Hardware-SoftwareCodesign Language) - высокоуровневый язык описания аппаратуры.

Информационная система на HaSCoL представляется в виде набора каналов для передачи сообщений и обработчиков, которые могут ожидать сообщения из канала (каналов) и посылать результаты работы в каналы. Время в языке дискретно и измеряется в тактах. Доставка сообщений не занимает времени. Присутствует поддержка блокирующей доставки сообщений — сообщение считается переданным только в том такте, в котором получатель сигнализирует о готовности принять сообщение. До этого момента посылающий обработчик частично блокируется.

Каждый обработчик является конвейером. Причем, если в нем более одной стадии, то данные, после обработки в 1-ой стадии, попадают во 2-ую только на следующем такте. При этом блокировка распространяется автоматически, так что приостановка одной стадии конвейера на ожидание готовности получателя сообщения влечет приостановку предыдущих стадий в случае, когда это необходимо, чтобы избежать потери данных. Также, автоматически генерируются конвейерные регистры для распространения параметров сообщений и локальных переменных. Это можно продемонстрировать на следующем примере, реализующем трехстадийный конвейер для вычисления квадратного многочлена:

```
poly(a, b, c, x : integer(32)) { –- конвейер получает данные из канала poly
    x2 = x * x | bx = b * x; -- символ ';' обозначает конец такта
    ax2 = a * x2 | bxc = bx + c; -- '|' — параллельное исполнение операторов
     send result(ax2 + bxc) –- «send» означает блокирующую посылку сообщения
}
```

В этом конвейере заводятся локальные переменные x2, bx, ax2, bxc, позволяющие донести результаты промежуточных вычислений до следующих стадий конвейера. Для данного примера автоматически генерируются 6 регистров для передачи параметров и переменных по стадиям. В оптимальном режиме работы обработчик принимает новый многочлен и точку на каждом такте и выдает на каждом такте значение многочлена, принятого два такта назад. Если канал «result» становится неготов принимать сообщения, то в конвейере накапливаются промежуточные результаты для двух последних точек и канал «poly» также становится неготов принимать сообщения.

Несколько обработчиков могут слушать один и тот же канал или посылать сообщения в один и тот же канал. Язык поддерживает возможность указания того, как следует разрешать возникающие при этом конфликты.

Также, язык поддерживает возможность декомпозиции устройства не несколько блоков — аналогично механизму «entity/architecture» в VHDL. Эта же возможность позволяет легко интегрировать существующие блоки, реализованные на VHDL, в устройство на HaSCoL.

Аналоги HaSCol делятся на два типа: самостоятельные языки и расширения для стандартных языков, таких как C/C++, Java, Python и т.д. В первой группе, в которой находится и сам HaSCoL из известных можно назвать только BlueSpec, если есть какие-то еще, то они по большей части неизвестны. Во второй группе примеров достаточно много (что достаточно логично - написать расширения для известного языка проще чем написать самостоятельный язык). Проблема же второй группы в том, что, имея такой инструмент, все равно приходится изучать, в первую очередь, особенности устройства и расширения, при этом знать сам язык, для которого и создано расширение, можно на очень низком уровне. При этом, мы практически не получаем преимуществ от использования известного языка, но получаем ограничения им накладываемые. В специальном же языке эти ограничения можно обойти.

Memory Window

Предыстория

Алгоритмы обработки изображений и видео очень удобно реализовывать на FPGA - в этих алгоритмах, обычно, легко параллелизуемые вычисления, которые, к тому же, имеют небольшие требования к памяти, но, с другой стороны, часто требовательны к скорости вычислений (вплоть до режима реального времени). Есть у этих алгоритмов и еще одна общая особенность: каждый из них можно разделить на две части: в первой происходит работа с памятью: идет обход изображения, а во второй происходят

вычисления. Очевидно, что обход изображения очень легко абстрагировать от конкретного алгоритма. Таким образом, если реализовать библиотеку, предоставляющую инструменты для легкой и быстрой реализации такого обхода, можно значительно ускорить разработку целого класса алгоритмов. В реализации такой библиотеки, или Ip core, если говорить в терминах аппаратных реализаций, и состояла моя задача.

Результат работы

Результатом моего проекта стала библиотека Memory Window, реализованная на языке HaSCoL. В ней реализована сущность - окно в памяти, которое представляет из себя матрицу (необязательно квадратную), в каждый момент времени содержащую в себе какую то часть изображения.

Интерфейс и детали реализации библиотеки на текущий момент (май 2014 года):

Окно в памяти реализовано с помощью макросов HaSCoL, и его создание происходит с помощью вызова макроса memoryWindow(N, M, elementSize, Mmax), где N - количество строк в окне, M - количество столбцов, elementSize - размер одного элемента изображения, Mmax - количество столбцов элементов в целом изображении. Этот макрос создает процесс memoryWindow, который имеет следующие каналы:

- входящие
    a. in config

входной канал начальной настройки. Устанавливается адрес в памяти, где лежат данные и общий размер этих данных

    b. in inpDataLine

входной канал для получения данных из памяти

- исходящие
    a. dataOutp

выходной канал, по которому передается рабочее окно

Принцип работы окна в памяти: процесс считывает данные, лежащие по указанному адресу и складывает их в некий буфер,находящийся в RAM памяти FPGA. Как только в буфере набирается достаточно данных чтобы сформировать матрицу требуемого размера (т.е. когда приходит M-й элемент N-й строки) - матрица сформировывается и посылается в выходной канал. После этого каждый такт посылается матрица, сдвинутая от предыдущей на один столбец элементов до тех пор, пока не будет достигнут конец строки. После этого в выводе матриц будет небольшая пауза, в связи с тем, что необходимо получить достаточно данных следующей строки, чтобы ее сформировать. На текущий момент единственный видимый вариант убрать эту задержку - это “переместить” эту задержку в начало, с помощью некоторых алгоритмических трюков. Но, к сожалению такой подход работает только при заранее известном (еще на момент проектирования библиотеки) размере картинки, что лишает смысла саму идею универсального кода. Важно отметить, что после начала работы данные передаются каждый такт, за исключением того момента, когда начинается новая строка, т.е. вычисления могут производится практически без простоя вычислительных блоков, зависящих от этих данных. Очевидно, что задержки при переводе строки линейно зависят от размера матрицы (а конкретно от количества столбцов в ней).

Обзор аналогов

Так как окно в памяти - действительно нужная сущность, универсальная для многих алгоритмов, то естественно, что ее уже реализовывали неоднократно. Есть готовые IP блоки написанные на vhdl, их достаточно много, но у них есть общий недостаток: кроме собственно окна в памяти в этом блоке обычно множество других вещей, возможно ненужных в данной ситуации. Найти отдельный блок только для окна в памяти у меня не получилось. Также далеко не для всех из них есть исходные тексты, что тоже может оказаться проблемой.

Еще я нашел реализованное окно в памяти для Vivado HLS. С ним сравнивать особенно интересно, потому что этот инструмент также (как и Хаскол) позиционируется как высокоуровневая замена для vhdl. Vivado HLS - это обертка на языке C++ для vhdl. У этого инструмента есть как свои плюсы, так и минусы, но речь сейчас не об этом. Для него существует реализация класса, который так и называется Memory Window. Если сравнивать этот класс с моей реализацией, то можно увидеть, что его возможности значительно ниже. В нем реализовано окно как массив, реализованы методы для сдвига на единицу в любом направлении и методы для чтения или записи. В нем нет буферизации, он не умеет обращаться к памяти, то есть это, также типовое действие, без которого реализация алгоритмов обработки данных бессмысленна, приходится делать самому, либо искать еще какой-то модуль для этого.

Список литературы

1. Обзор высокоуровневого языка разработки аппаратуры HaSCoL на примере клона процессора Xilinx Microblaze

О.Медведев, НИИ ИТ СПбГУ

url(18.05.2014):*http://oops.math.spbu.ru/dours/HaSCoL/KBSM.pdf*

2. Wikipedia - Программируемая пользователем вентильная матрица

url(18.05.2014): http://ru.wikipedia.org/wiki/FPGA

3. Accelerating multiple alignment on FPGA with a high-level hardware description language

Oleg Medvedev St.Petersburg State University

url(18.05.2014): http://oops.math.spbu.ru/dours/HaSCoL/SECR11.pdf

4. Hardware Description Language Based on Message Passing and Implicit Pipelining

Oleg Medvedev, Dmitri Boulytchev St.Petersburg State University

url(18.05.2014): http://oops.math.spbu.ru/dours/HaSCoL/EWDT09.pdf