Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Компьютерные системы и сети (ИУ6)»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
по курсу «Архитектура ЭВМ»

«Методология разработки и верификации ускорителей вычислений на платформе Xilinx Alveo»

Студент: ИУ7-53Б _____ М. Д. Маслова
(группа) (подпись, дата) (И. О. Фамилия)

Преподаватель: _____ Е. Н. Дубровин
(подпись, дата) (И. О. Фамилия)

Москва, 2021

Содержание

Введение

Целью данной работы является изучение архитектуры гетерогенных вычислительных систем и технологии разработки ускорителей вычислений на базе ПЛИС фирмы Xilinx.

В ходе лабораторной работы предлагается изучить основные сведения о платформе Xilinx Alveo U200, разработать RTL (Register Transfer Language, язык регистровых передач)) описание ускорителя вычислений по индивидуальному варианту, выполнить генерацию ядра ускорителя, выполнить синтез и сборку бинарного модуля ускорителя, разработать и отладить тестирующее программное обеспечение на серверной хост-платформе, провести тесты работы ускорителя вычислений.

1 Основные теоретические сведения

В данном разделе будут описаны технология разработки ускорителей вычислений на модулях Xilinx Alveo, а также архитектура разрабатываемого усорителя.

1.1 Технология разработки ускорителей вычислений на модулях Xilinx Alveo

Ускорителями вычислений принято называть специальные аппаратные устройства, способные выполнять ограниченный ряд задач с большей параллельностью и за меньшее время в сравнении с универсальными микропроцессорными ЭВМ. Как правило, ускоритель представляет собой структуру, включающую большое количество примитивных микропроцессорных устройств, объединенных шинами связей.

Создание ускорителей вычислений является трудоемким процессом, так как охватывает не только аппаратную разработку самого устройства, но и предполагает оптимизацию архитектуры ЭВМ для обеспечения наибольшей пропускной способности каналов передачи операндов и результатов, а также минимизации задержек и вычислительных затрат при ожидании работы ускорителей. Можно условно разделить ускорители на два класса: ускорители на основе СБИС и на основе ПЛИС.

В данной лабораторной работе мы изучим технологию создания ускорителей вычислений на основе ПЛИС (ускоритель **Xilinx Alveo U200** на основе ПЛИС xcu200-fsgd2104-2-e архитектуры Xilinx UltraScale).

Для работы с ускорительной платой разработано специальное окружение **XRT** (Xilinx Runtime), включающее компоненты пользовательского пространства и драйвера ядра.

В оборудовании, используемом для проведения лабораторной работы, использована так называемая XDMA сборка XRT, которая предполагает следующий сценарий взаимодействия ускорителя и пользовательского ПО:

1. Пользовательское ПО сканирует и инициализирует доступные ускорительные платы, совместимые с XRT, определяет доступные ресурсы, создает программное окружение пользовательского аппаратного ядра ускорителя

(далее используется термин kernel).

2. Ресурсы локальной памяти ускорительной платы отображаются в пространство памяти хост системы.

3. Инициализируются каналы DMA для прямого доступа к памяти ускорителя.

4. Данные, подлежащие обработке, копируются из ОЗУ в локальную память ускорителя посредством DMA.

5. Ядру ускорителя (или нескольким ядрам) посредством записи управляющих регистров, передаются параметры вычислений. Пользователь может увеличивать количество параметров по своему усмотрения. Типичным случаем является передача указателей на начало буферов исходных операндов и буфера результата, а также количество обрабатываемых значений.

6. Хост-система выдает сигнал Start ядрам ускорителей, после чего начинается обработка внутри платы Xilinx Alveo.

7. По завершении обработки kernel устанавливает флаг DONE, что вызывает прерывание по шине PCIe.

8. Драйвер обрабатывает прерывание и сообщает пользовательскому ПО о завершении обработки.

9. Пользовательское ПО инициализирует DMA передачу результатов из локальной памяти ускорителя в ОЗУ хост-системы.

1.2 Описание архитектуры разрабатываемого ускорителя

В ходе лабораторной работы будет использован базовый шаблон так называемого RTL проекта VINC, который может быть создан в IDE Xilinx Vitis и САПР Xilinx Vivado. Шаблон VINC выполняет попарное сложение чисел исходного массива и сохраняет результаты во втором массиве. Проект VINC включает:

– Проект ПО хоста, выполняющий инициализацию аппаратного ядра и его тестирование через OpenCL вызовы.

– Синтезируемый RTL проект ядра ускорителя на языках Verilog и SystemVerilog.

– Функциональный тест ускорителя VINC на языке SystemVerilog.

Проект VINC представляет собой аппаратное устройство, связанное шиной AXI4 MM (Memory mapped) с DDR[i] памятью, и получающее настроечные параметры по интерфейсу AXI4 Lite от программного обеспечения хоста (рисунок 1.1). В рамках всей системы используется единое 64-х разрядное адресное пространство, в котором формируются адреса на всех AXI4 шинах.

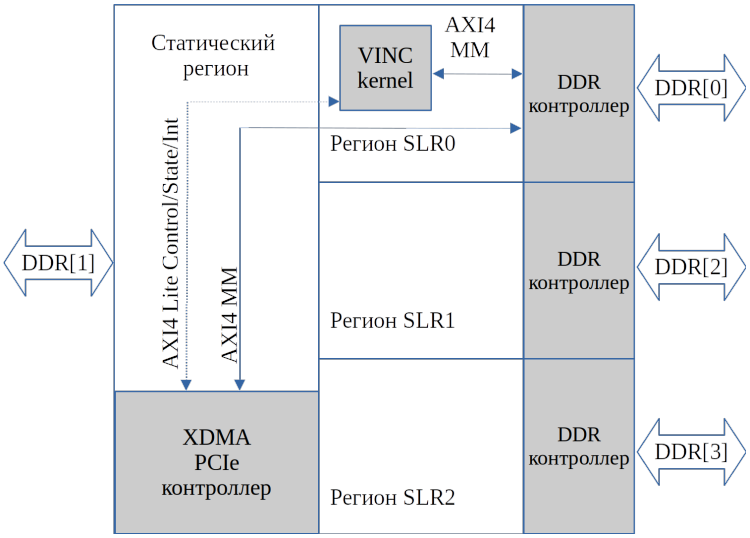

Рисунок 1.1 – Функциональная схема разрабатываемой аппаратной системы

В каждой карте U200 имеется возможность подключить ускоритель к любому DDR[i] контроллеру в том регионе, где будет размещен проект. Всего для пользователя доступны 3 динамических региона: SLR0,1,2, для которых выделены каналы локальной памяти DDR[0], DDR[2], DDR[3] соответственно. Вся подключенная память DDR[0..3] доступна со стороны статического региона, в котором размещена аппаратная часть XRT.

Выбор одного из регионов для размещения проектов осуществляется на этапе так называемой линковки конфигурационного файла при помощи компилятора v++(фактически: компоновки, размещение и трассировки нескольких проектов в единый конфигурационный файл).

2 Практическая часть

В данном разделе представлен ход выполнения лабораторной работы.

2.1 Исходный проект VINC

В данном подразделе описывается работа с исходным проектом VINC, код инкремента данных в котором представлен на рисунке 2.1.

```
77    // Adder function
78    always @(posedge s_axis_aclk) begin
79      for (i = 0; i < LP_NUM_LOOPS; i = i + 1) begin
80        d2_tdata[i*C_ADDER_BIT_WIDTH+:C_ADDER_BIT_WIDTH] <= d1_tdata[C_ADDER_BIT_WIDTH*i+:C_ADDER_BIT_WIDTH] + d1_constant;
81      end
82    end
```

Рисунок 2.1 – Код инкремента данных

2.1.1 Моделирование

На рисунке 2.2 рпредставлена транзация чтения данных вектора на шине AXI4 MM из DDR памяти, на рисунке 2.3 – транзация записи данных, на рисунке 2.4 – инкремент данных в модуле.

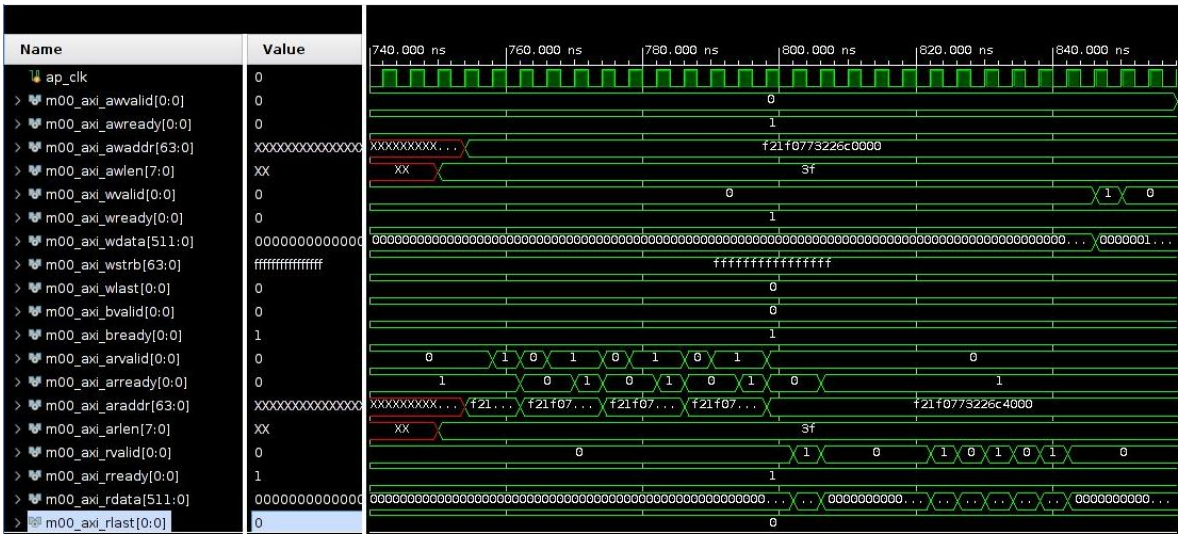

Рисунок 2.2 – Транзакция чтения данных вектора на шине AXI4 MM из DDR памяти

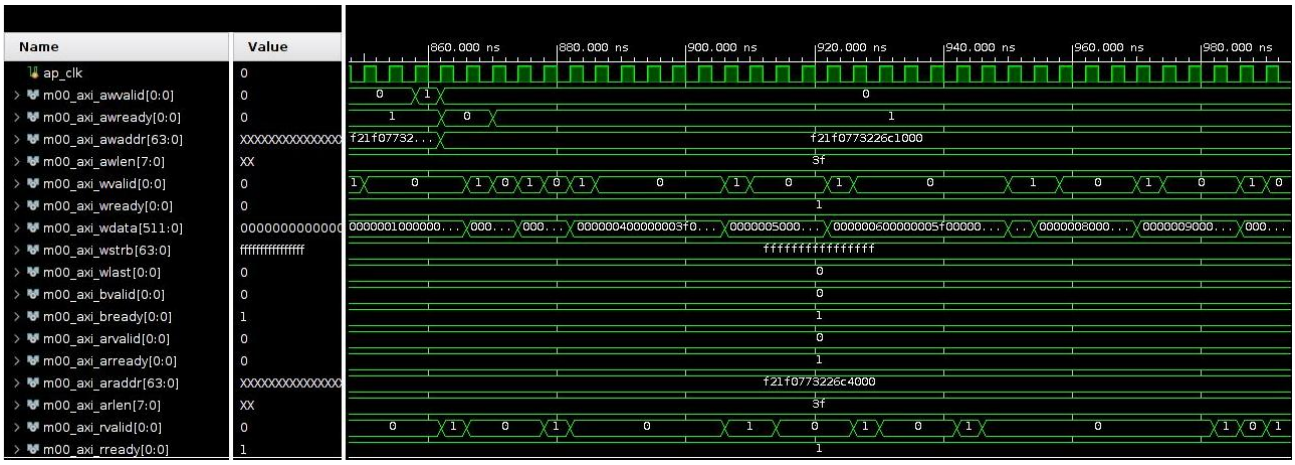

Рисунок 2.3 – Транзакция записи данных на шине AXI4 MM

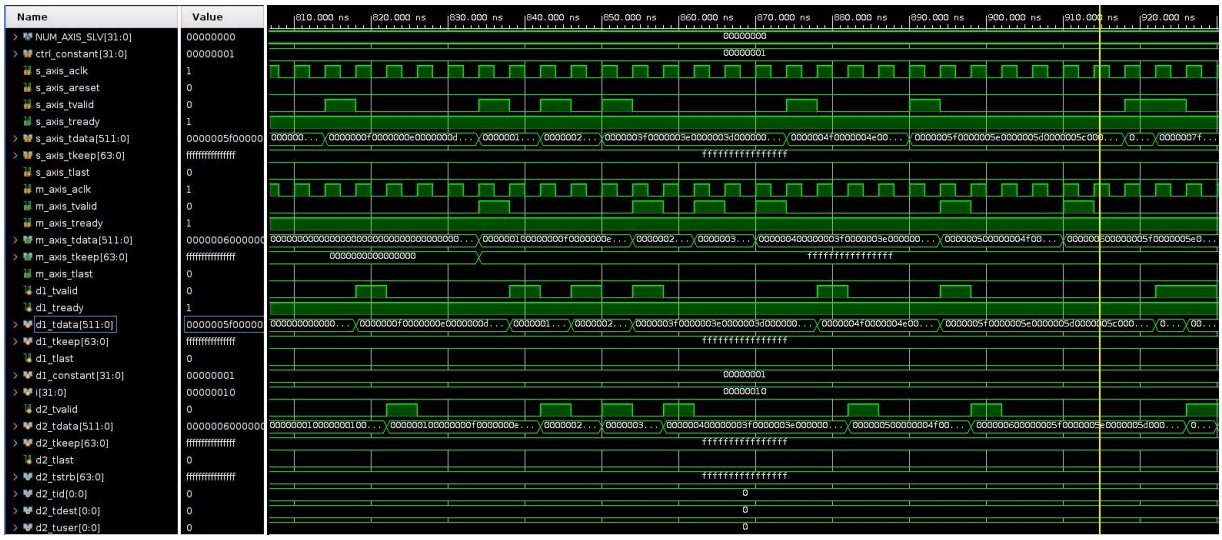

Рисунок 2.4 – Инкремент данных в модуле

2.2 Проект VINC по варианту

В данном подразделе описывается работа с проектом VINC по индивидуальному варианту, код инкремента данных в котором представлен на рисунке 2.5.

```
77      // Adder function
78      always @(posedge s_axis_aclk) begin
79        for (i = 0; i < LP_NUM_LOOPS; i = i + 1) begin
80          d2_tdata[i*C_ADDER_BIT_WIDTH+:C_ADDER_BIT_WIDTH] <= d1_tdata[C_ADDER_BIT_WIDTH*i+:C_ADDER_BIT_WIDTH] & 'hf0f0f0f0f0 + 10;
81        end
82      end
```

Рисунок 2.5 – Код инкремента данных

2.2.1 Моделирование

На рисунке 2.6 рпредставлена транзация чтения данных вектора на шине AXI4 MM из DDR памяти, на рисунке 2.7 – транзация записи данных, на рисунке 2.8 – инкремент данных в модуле.

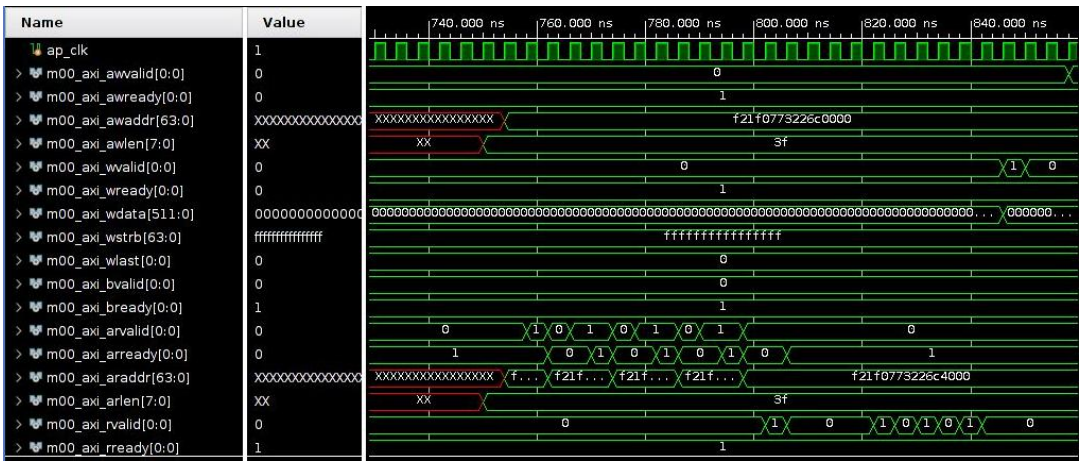

Рисунок 2.6 – Транзакция чтения данных вектора на шине AXI4 MM из DDR памяти

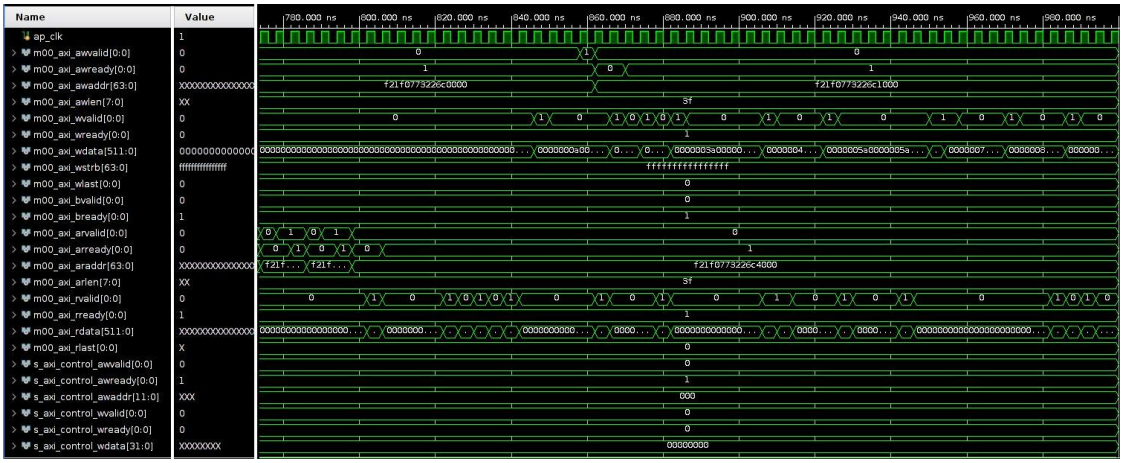

Рисунок 2.7 – Транзакция зaпписи данных на шине AXI4 MM

2.2.2 Сборка проекта

Для сборки проекта компилятором `v++` используется конфигурационный файл **.cfg*, который содержит основную информацию для работы компилятора:

– количество и условные имена экземпляров ядер;

– тактовая частота работы ядра;

Рисунок 2.8 – Инкремент данных в модуле

– для каждого ядра: выбор региона SLR, памяти DDR, высокопроизводительной памяти PLRAM;

– параметры синтеза и оптимизации проекта.

На листинге 2.1 представлен конфигурационный файл использующий в данной работе в соответствии с индивидуальным вариантом.

Листинг 2.1 – Конфигурационный файл

```
[connectivity]
nk=rtl_kernel_wizard_0:1:vinc0
slr=vinc0:SLR2
sp=vinc0.m00_axi:DDR[3]

[vivado]
prop=run.impl_1.STEPS.OPT_DESIGN.ARGS.DIRECTIVE=Explore
prop=run.impl_1.STEPS.PLACE_DESIGN.ARGS.DIRECTIVE=Explore
prop=run.impl_1.STEPS.PHYS_OPT_DESIGN.IS_ENABLED=true
prop=run.impl_1.STEPS.PHYS_OPT_DESIGN.ARGS.DIRECTIVE=AggressiveExplore
prop=run.impl_1.STEPS.ROUTE_DESIGN.ARGS.DIRECTIVE=Explore
```

В результате компиляции генерируется файл `*.xclbin`, который может быть передан в ускорительную карту, также генерируется лог файл `v++*.log` и файл описания ресурсов `*.xclbin.info`, содержимое которых приведено в приложении А.

2.2.3 Тестирование

Для тестирования используется программа, исходный код которой представлен в файле `host_example.cpp`. Цикл проверки результатов работы ускорителя по индивидуальному варианту представлен на листинге 2.2.

Листинг 2.2 – Модифицированный модуль `host_example.cpp`

```
1    for (cl_uint i = 0; i < number_of_words; i++) {
2        if ((h_data[i] & 0xf0f0f0f0f0 + 10) != h_axi00_ptr0_output[i]) {
3            printf("ERROR in rtl_kernel_wizard_0::m00_axi - array index %d (host
                 addr 0x%03x) - input=%d (0x%x), output=%d (0x%x)\n",
4                    i, i*4, h_data[i], h_data[i], h_axi00_ptr0_output[i],
                        h_axi00_ptr0_output[i]);
5            check_status = 1;
6        }
```

Результаты тестирования представлены на рисунке 2.9.

```
iu7072@dl580:~/workspace/Alveo_lab1_kernels/vivado_rtl_kernel/rtl_kernel_wizard_0_ex/exports$ xgdb --args rtl_kernel_wizard_
0_host_example.exe /iu_home/iu7072/workspace/Alveo_lab1_kernels/src/vitis_rtl_kernel/rtl_kernel_wizard_0/vinc.xclbin
GNU gdb (GDB) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from rtl_kernel_wizard_0_host_example.exe...
(gdb) run
Starting program: /iu_home/iu7072/workspace/Alveo_lab1_kernels/vivado_rtl_kernel/rtl_kernel_wizard_0_ex/exports/rtl_kernel_w
izard_0_host_example.exe /iu_home/iu7072/workspace/Alveo_lab1_kernels/src/vitis_rtl_kernel/rtl_kernel_wizard_0/vinc.xclbin
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff5b2f700 (LWP 4483)]
INFO: Found 1 platforms
INFO: Selected platform 0 from Xilinx
INFO: Found 1 devices
CL_DEVICE_NAME xilinx_u200_xdma_201830_2
Selected xilinx_u200_xdma_201830_2 as the target device
INFO: loading xclbin /iu_home/iu7072/workspace/Alveo_lab1_kernels/src/vitis_rtl_kernel/rtl_kernel_wizard_0/vinc.xclbin
[New Thread 0x7ffff4f2d700 (LWP 4858)]
[New Thread 0x7fffeffff700 (LWP 4859)]
[New Thread 0x7fffef7fe700 (LWP 4860)]
[New Thread 0x7fffeeffd700 (LWP 4861)]
[New Thread 0x7fffee7fc700 (LWP 4862)]
[New Thread 0x7fffedffb700 (LWP 4863)]
INFO: Test completed successfully.
[Thread 0x7ffff4f2d700 (LWP 4858) exited]
[Thread 0x7ffff5b2f700 (LWP 4483) exited]
[Thread 0x7fffedffb700 (LWP 4863) exited]
[Thread 0x7fffee7fc700 (LWP 4862) exited]
[Thread 0x7fffeeffd700 (LWP 4861) exited]
[Thread 0x7fffef7fe700 (LWP 4860) exited]
[Thread 0x7fffeffff700 (LWP 4859) exited]
[Inferior 1 (process 4472) exited normally]
```

Рисунок 2.9 – Результаты тестирования

Таким образом, все тесты были пройдены, и программа на ускорителе работает верно.

3 Контрольные вопросы

В данном разделе представлены ответы на контрольные вопросы.

– **Преимущества и недостатки XDMA и QDMA платформ**

Сборка QDMA, доступная на картах ускорителей Alveo, предоставляет разработчикам прямое потоковое соединение с низкой задержкой между хостом и ядрами. Оболочка QDMA включает высокопроизводительный DMA, который использует несколько очередей, оптимизированных как для передачи данных с высокой пропускной способностью, так и для передачи данных с большим количеством пакетов. Только QDMA позволяет передавать поток данных непосредственно в логику FPGA параллельно с их обработкой.

Оболочка XDMA требует, чтобы данные сначала были полностью перемещены из памяти хоста в память FPGA, прежде чем логика FPGA сможет начать обработку данных, что влияет на задержку на запуска задачи.

Потоковая передача напрямую в работающие ускорительные ядра позволяет быстро и без излишней буферизации передавать операнды и результаты вычислений на хост по потоковому интерфейсу AXI4 Stream. Решение QDMA подходит для приложений, в которых вычисления строятся на передачи сравнительно небольших пакетов, но при этом требуется высокая производительность и минимальная задержка отклика.

– **Последовательность действий, необходимых для инициализации ускорителя со стороны хост-системы**

1. Сканирование и инициализация доступных ускорительных план, совместымых с XRT.
2. Определение доступных ресурсов.
3. Создание программного окружения пользовательского аппаратного ядра ускорителя.
4. Инициализация локальной памяти ускорителя посредством DMA.
5. Передача параметров вычислений ядру ускорителя.
6. Подача сигнала Start для начала обработку внутри платы.
7. Инициализация DMA передачи результатов из локальной памяти ускорителя в ОЗУ хост-системы по окончанию обработки.

– **Какова процедура запуска задания на исполнения в ускорительном ядре VINC**

1. Копирование данных из .xclbin и данных, подлежащих обработке, в локальную память ускорителя посредством DMA.
2. Создание исполняемого файла в памяти ускорителя.
3. Получение параметров вычислений.
4. Начало обработки по сигналу Start от хост-системы.
5. Обработка.
6. Установка флага DONE для сообщения хочт-системе о завершении обработки.

– **Процесс линковки на основании содержимого файла v++_*.log**

1. Анализ профиля устройства.
2. Анализ конфигурационного файла.
3. Поиск необходимых интерфейсов.
4. Создание графа связности системы.
5. Связывание синтезированных ядер с платформой (FPGA linking synthesized kernels to platform).
6. Оптимизация логики ПЛИС (FPGA logic optimization).
7. Размещение логического блока в динамическом регионе (FPGA logic placement).
8. Маршрутизация ПЛИС (FPGA routing).
9. Генерация файла *.xclbin.

Заключение

В ходе лабораторной работы были изучены архитектуры гетерогенных вычислительных систем и технологии разработки ускорителей вычислений на базе ПЛИС фирмы Xilinx.

Были выполнены следующие задачи:

– изучены основные сведения о платформе Xilinx Alveo U200;

– разработано RTL описание ускорителя вычислений по индивидуальному варианту;

– выполнена генерация ядра ускорителя;

– выполнены синтез и сборка бинарного модуля ускорителя;

– разработано и отлажено тестирующее программное обеспечение на серверной хост-платформе;

– проведены тесты работы ускорителя вычислений.

Таким образом, все поставленные задачи были выполнены, а цель достигнута.