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 «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе № 5 по дисциплине "Архитектура ЭВМ"

Тема Разработка ускорителей вычислений средствами САПР высокоуровневого синтеза Xilinx Vitis HLS

Студент Хамзина Р. Р.

Группа ИУ7-53Б

Оценка (баллы)

Преподаватель Дубровин Е. Н.

Москва — 2021 г.

Содержание

Введение

Целью данной лабораторной работы является изучение методики и технологии синтеза аппаратных устройств ускорения вычислений по описаниям на языках высокого уровня.

Для достижения поставленной цели необходимо решить следующие задачи:

- рассмотреть маршрут проектирования устройств, представленных в виде синтаксических конструкций ЯВУ C/C++;
- изучить принципы работы IDE Xilinx Vitis HLS и методику анализа и отладки устройств;
- разработать ускоритель вычислений по индивидуальному заданию;
- разработать код для тестирования ускорителя;
- реализовать ускоритель с помощью средств высокоуровненного синтеза;
- выполнить его отладку.

1 Основные теоретические сведения

1.1 Методология ускорения вычислений на основе ПЛИС

Микропроцессоры и графические процессоры имеют предопределенную архитектуру с фиксированным количеством ядер, набором инструкций, и жесткой архитектурой памяти, и обладают высокими тактовыми частотами и хорошо сбалансированной конвейерной структурой. Графические процессоры масштабируют производительность за счет большого количества ядер и использования параллелизма SIMD/SIMT, что представлено на рисунке 1.1. В отличие от них, программируемые устройства представляют собой полностью настраиваемую архитектуру, которую разработчик может использовать для размещения вычислительных блоков с требуемой функциональностью. В таком случае, высокий уровень производительности достигается за счет создания длинных конвейеров обработки данных, а не за счет увеличения количества вычислительных единиц.

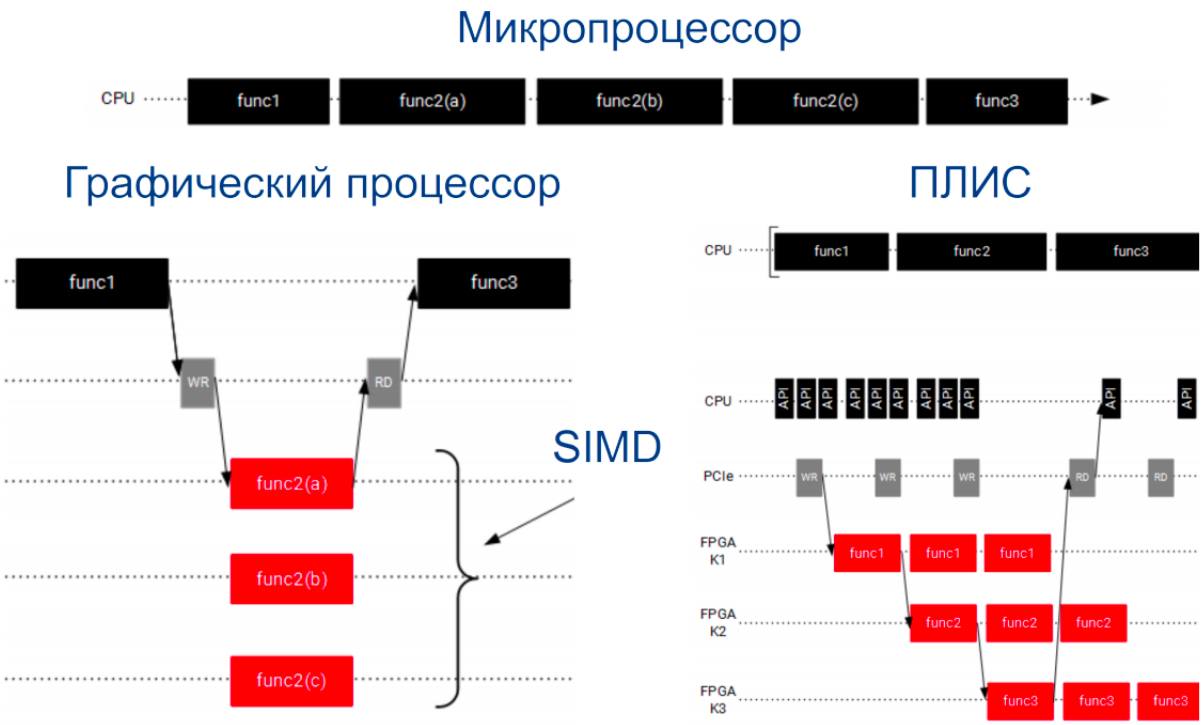


Рисунок 1.1 – Принципы организации вычислений на различных платформах

Методологию создания ускорителей на ПЛИС с применением средств синтеза высокого уровня (High Level Synthesis, HLS) можно представить в виде трех этапов:

- создание архитектуры приложения;
- разработка ядра аппаратного ускорителя на языках C/C++;
- анализ производительности и выявление способов ее повышения.

1.2 Оптимизация времени обработки и пропускной способности

Существует несколько подходов к оптимизации.

1.2.1 Конвейерная обработка циклов

Полагая, что одна итерация цикла занимает более одного такта (фаза выборки данных, фаза вычисления, фаза записи результата), может быть организован конвейер выполнения. Для этого необходимо поместить в тело цикла прагму <HLS PIPELINE>.

Без конвейерной обработки каждая последующая итерация цикла начинается через некоторое число тактов. При конвейерной обработке цикл может начинать последующие итерации цикла менее чем за некоторое число тактов, например, в каждом втором такте или в каждом такте.

Стоит отметить, что конвейерная обработка цикла приводит к разворачиванию любых циклов, вложенных внутрь конвейерного цикла. Если внутри цикла существуют зависимости по данным, может оказаться невозможным достичь запуска новой итерации в каждом такте.

1.2.2 Разворачивание циклов

Разворачивание циклов является общепризнанным механизмом снижения времени выполнения циклов. Этот механизм может быть описан самим разработчиком вручную, если он просто повторит вычисления и сократит количество итераций цикла. С помощью прагмы <HLS UNROLL> компилятор v++ позволяет запустить механизм автоматического разворачивания цикла частично или полностью.

1.2.3 Потоковая обработка

Реализация механизма потоковой обработки (#pragma HLS DATAFLOW) также опирается на представление вычислительных действий в виде многостадийного конвейера. Однако, в то время как директива конвейеризации (#pragma HLS PIPELINE) используется для реализации конвейера выполнения операций внутри функций или циклов, потоковая обработка данных позволяет сформировать конвейер из более крупных вычислительных

блоков: нескольких функций или нескольких последовательных циклических конструкций. Таким образом, директива HLS DATAFLOW позволяет сформировать вычислительный конвейер на уровне задач. Для этих целей компилятор HLS Vitis выполнит анализ зависимостей по данным между задачами и реализует структуру обрабатывающих блоков и FIFO очередей между ними.

2 Функции ядра на основе индивидуального задания

В листингах 2.1-2.4 представлены функций ядра: не оптимизированный цикл, конвейерная организация цикла, частично равернутый цикл и конвейерный и частично развернутый цикл.

app_no_pragm... | app_pipeline... | app_unrolled... | app_pipe_un...

```
1  extern "C" {
2      void var018_no_pragmas(int* c, const int* a, const int* b, const int len) {
3          int acc = 0;
4
5          for (int i = 0; i < len; i++) {
6              acc += a[i] * b[i];
7              c[i] = acc;
8          }
9
10         int minB = b[0];
11
12         for (int i = 1; i < len; i++) {
13             if (minB > b[i]) {
14                 minB = b[i];
15             }
16         }
17     }
18 }
19
```

Рисунок 2.1 – Не оптимизированный цикл на основе индивидуального задания

app_no_pragm... | app_pipeline... | app_unrolled... | app_pipe_un...

```
1  extern "C" {
2      void var018_pipelined(int* c, const int* a, const int* b, const int len) {
3          #pragma HLS PIPELINE
4          int acc = 0;
5
6          for (int i = 0; i < len; i++) {
7              acc += a[i] * b[i];
8              c[i] = acc;
9          }
10
11         int minB = b[0];
12
13         for (int i = 1; i < len; i++) {
14             if (minB > b[i]) {
15                 minB = b[i];
16             }
17         }
18     }
19 }
20
```

Рисунок 2.2 – Конвейерная организация цикла на основе индивидуального задания

app_no_pragm... app_pipeline... app_unrolled... app_pipe_un...

```
1  extern "C" {
2      void var018_unrolled(int* c, const int* a, const int* b, const int len) {
3          #pragma HLS UNROLL
4          int acc = 0;
5
6          for (int i = 0; i < len; i++) {
7              acc += a[i] * b[i];
8              c[i] = acc;
9          }
10
11         int minB = b[0];
12
13         for (int i = 1; i < len; i++) {
14             if (minB > b[i]) {
15                 minB = b[i];
16             }
17         }
18     }
19 }
20
```

Рисунок 2.3 – Частично развернутый цикл на основе индивидуального задания

app_no_pragm... app_pipeline... app_unrolled... app_pipe_un...

```
1  extern "C" {
2      void var018_pipe_unroll(int* c, const int* a, const int* b, const int len) {
3          #pragma HLS DATAFLOW
4          int acc = 0;
5
6          for (int i = 0; i < len; i++) {
7              acc += a[i] * b[i];
8              c[i] = acc;
9          }
10
11         int minB = b[0];
12
13         for (int i = 1; i < len; i++) {
14             if (minB > b[i]) {
15                 minB = b[i];
16             }
17         }
18     }
19 }
20
```

Рисунок 2.4 – Конвейерный и частично развернутый цикл на основе индивидуального задания

3 Режим Emulation-SW

На рисунке 3.1 показаны результаты рабты приложения в режиме Emulation-SW.

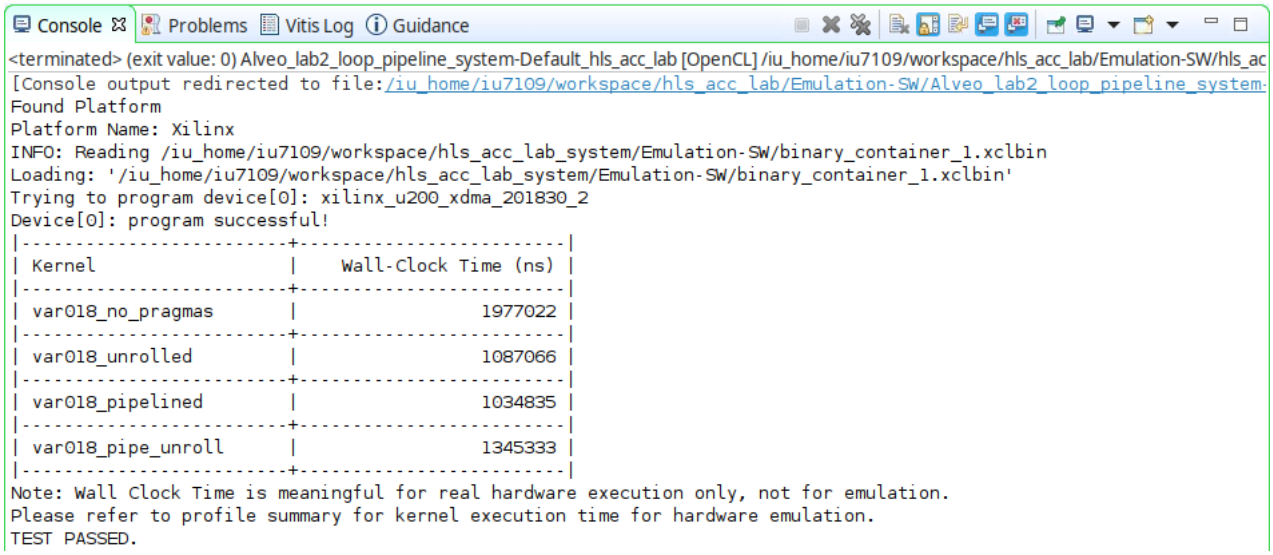


Рисунок 3.1 – Результаты работы приложения в режиме Emulation-SW

4 Режим Emulation-HW

На рисунке 4.1 представлена копия экрана Assistant View для сборки Emulation-HW.

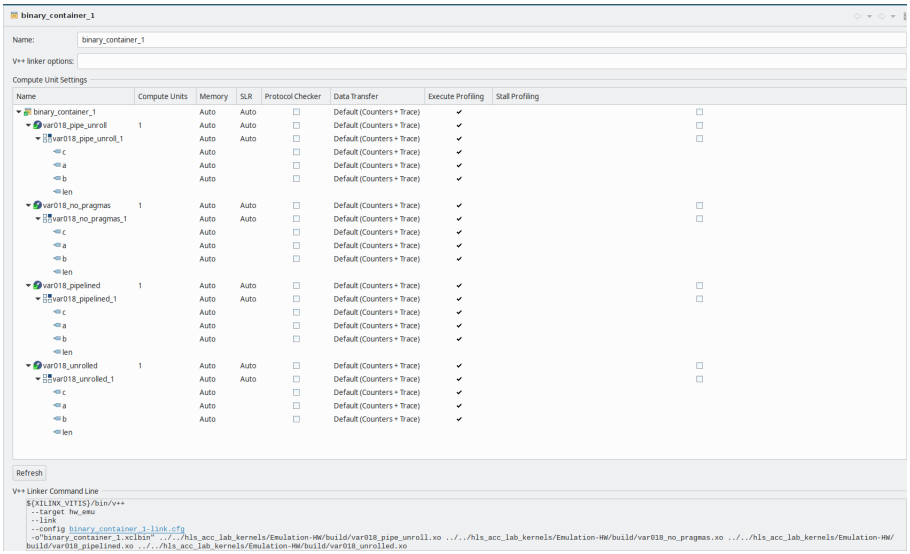


Рисунок 4.1 – Копия экрана Assistant View для сборки Emulation-HW

На рисунке 4.2 показаны результаты работы приложения в режиме Emulation-HW.

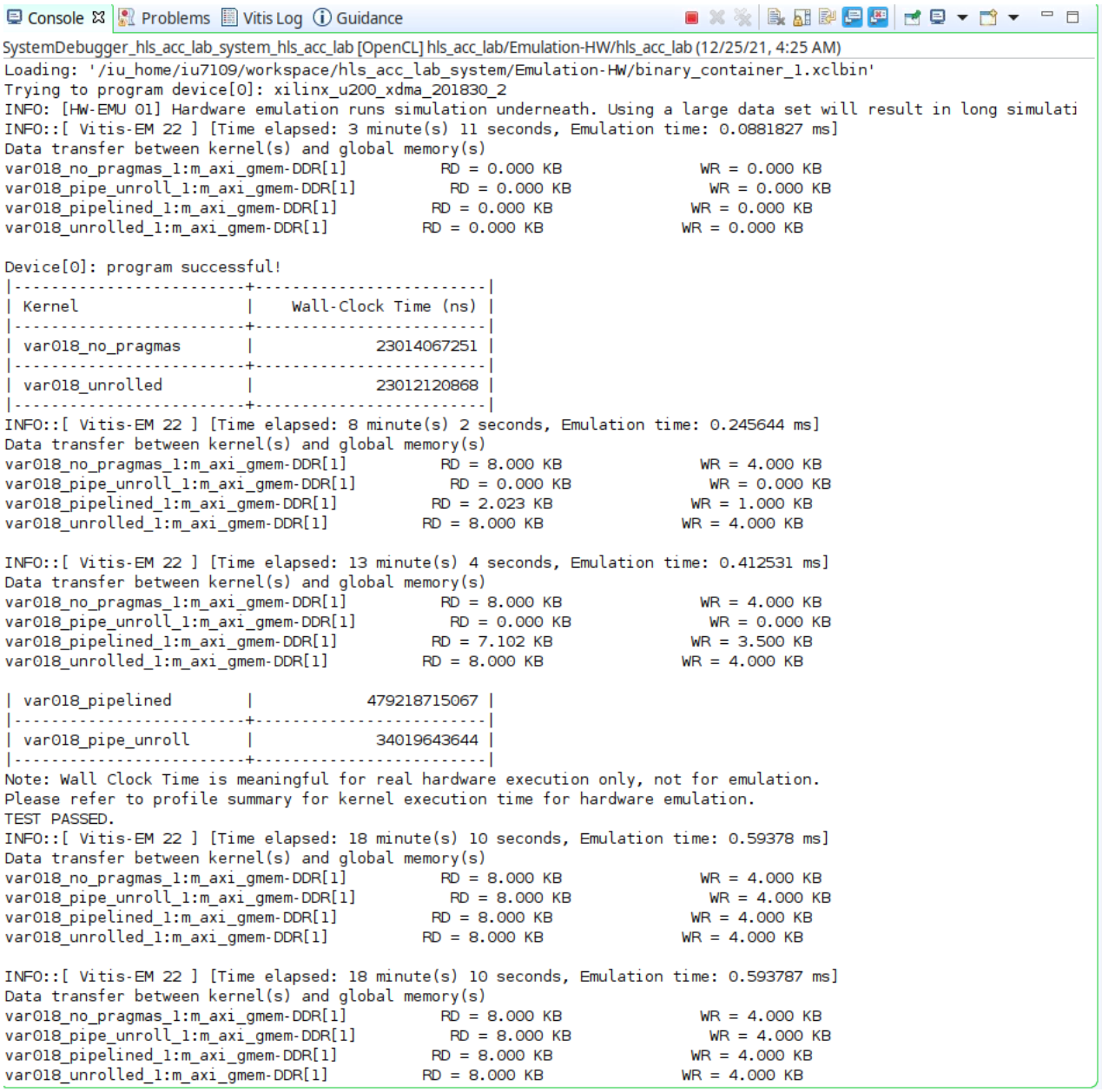


Рисунок 4.2 – Результаты работы приложения в режиме Emulation-HW

Окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW представлено на рисунках 4.3-4.4.

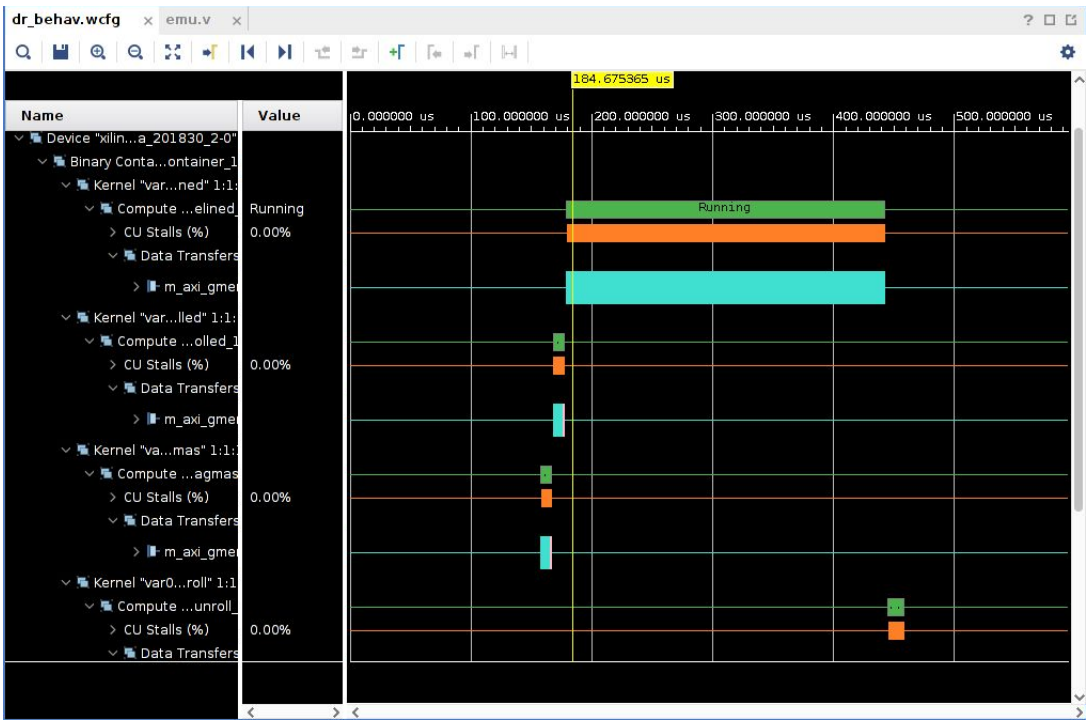


Рисунок 4.3 – Окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW - 1

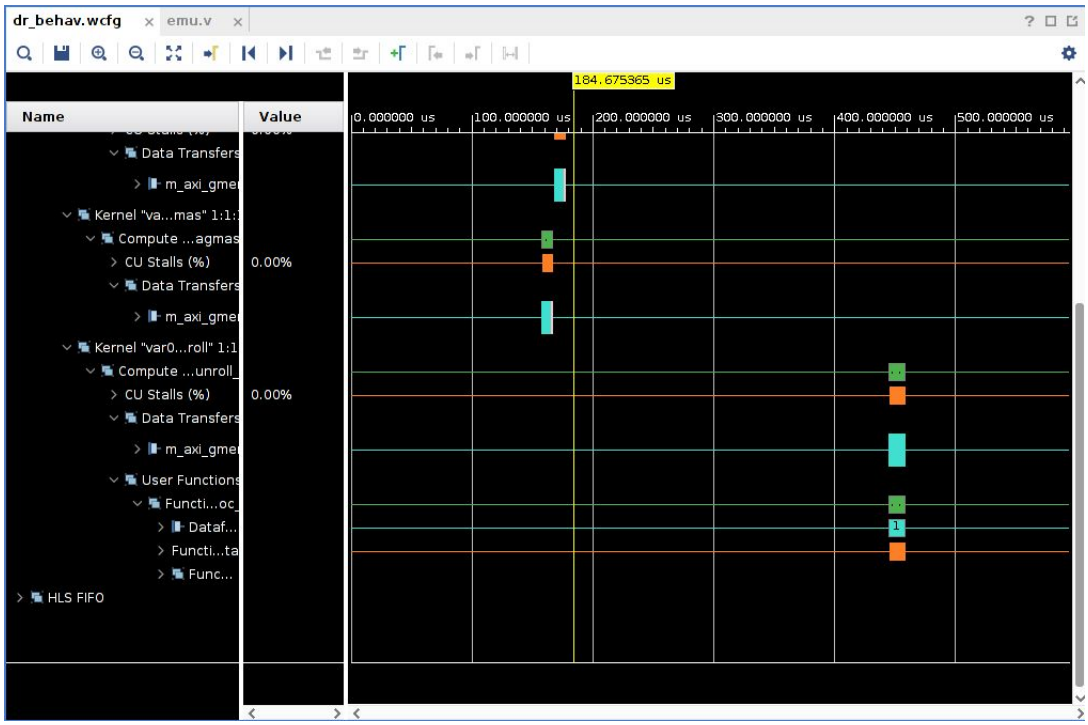


Рисунок 4.4 – Окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW - 2

5 Режим Hardware

На рисунке 5.1 показаны результаты работы приложения в режиме Hardware.

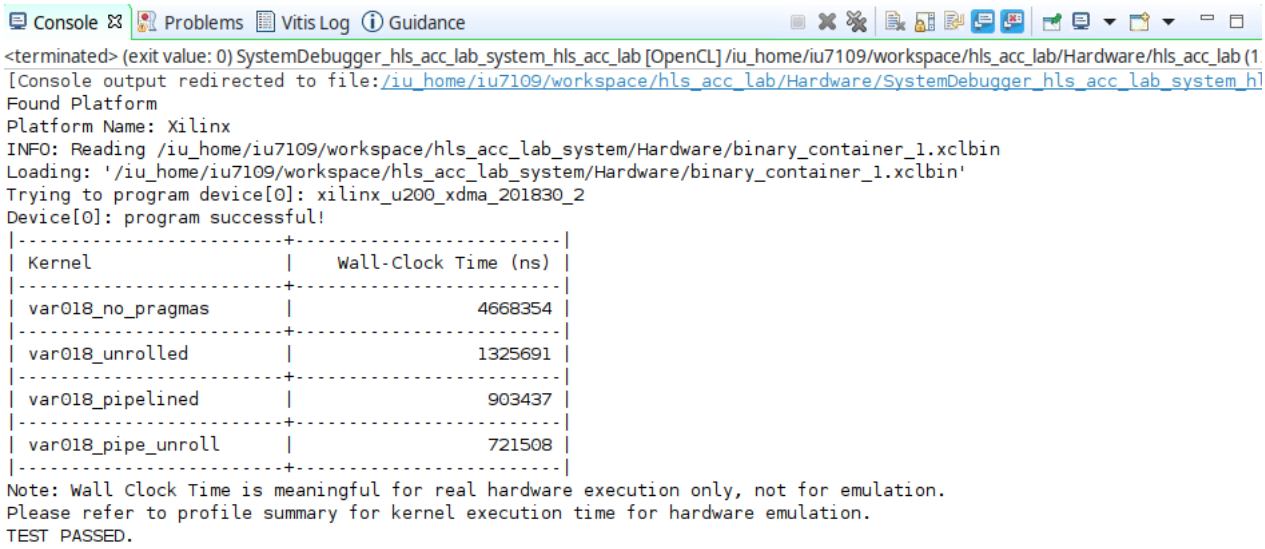


Рисунок 5.1 – Результаты работы приложения в режиме Hardware

На рисунках 5.2-5.4 представлены копии экранов для вкладок «Summary», «System Diagram», «Platform Diagram».

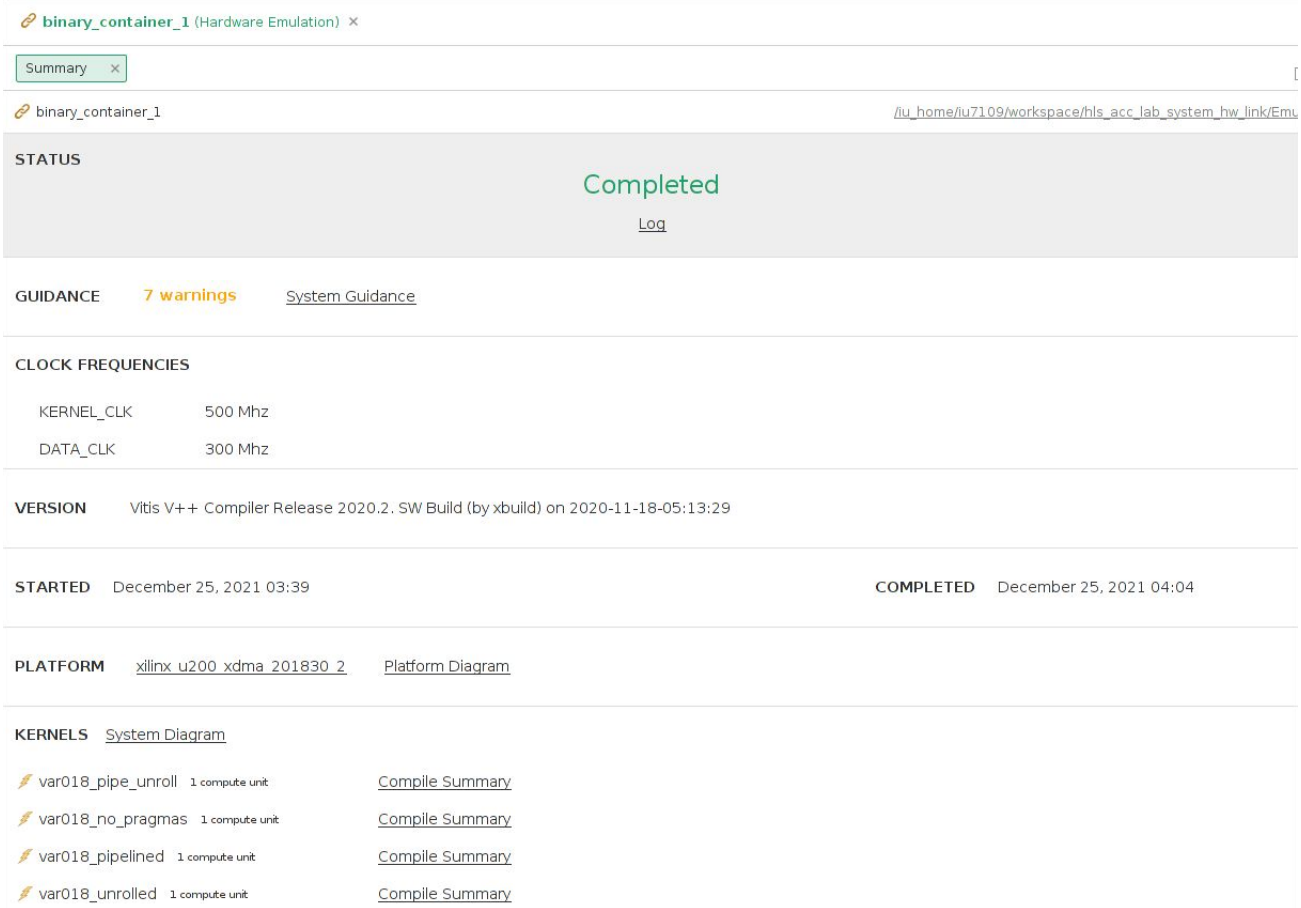
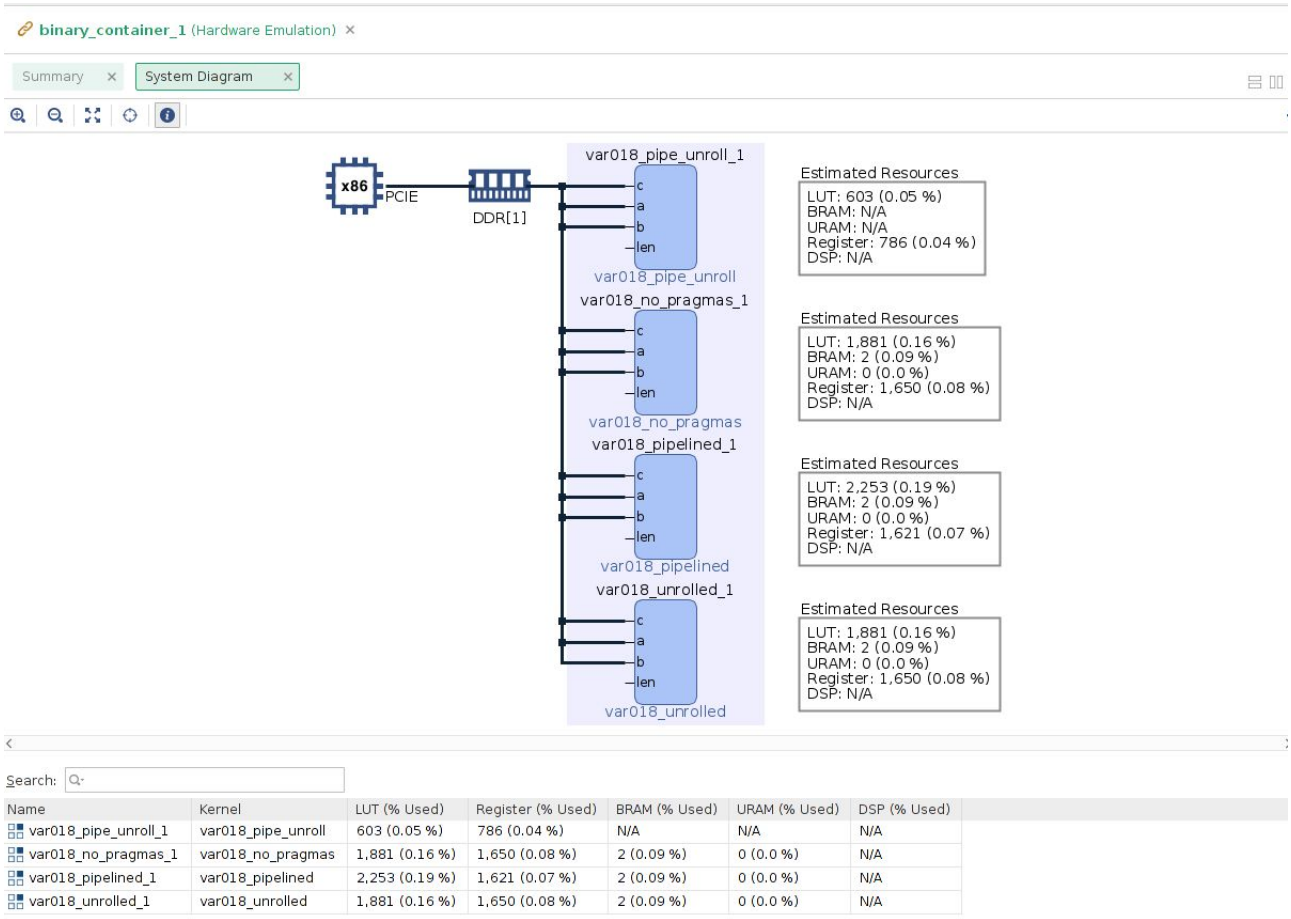


Рисунок 5.2 – Копия экрана вкладки «Summary»



Name	Kernel	LUT (% Used)	Register (% Used)	BRAM (% Used)	URAM (% Used)	DSP (% Used)
var018_pipe_unroll_1	var018_pipe_unroll	603 (0.05 %)	786 (0.04 %)	N/A	N/A	N/A
var018_no_pragmas_1	var018_no_pragmas	1,881 (0.16 %)	1,650 (0.08 %)	2 (0.09 %)	0 (0.0 %)	N/A
var018_pipelined_1	var018_pipelined	2,253 (0.19 %)	1,621 (0.07 %)	2 (0.09 %)	0 (0.0 %)	N/A
var018_unrolled_1	var018_unrolled	1,881 (0.16 %)	1,650 (0.08 %)	2 (0.09 %)	0 (0.0 %)	N/A

Рисунок 5.3 – Копия экрана вкладки «System Diagram»

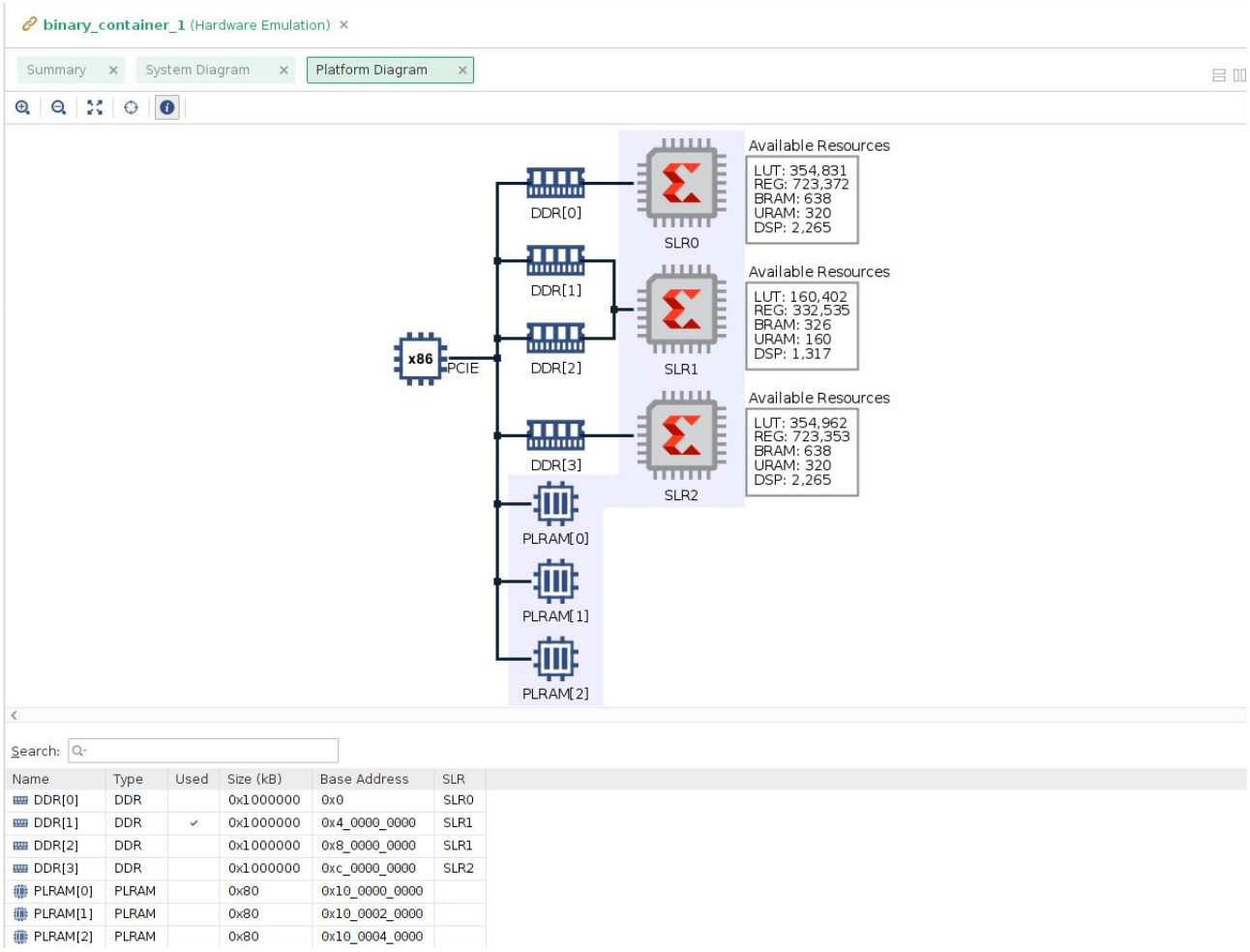


Рисунок 5.4 – Копия экрана вкладки «Platform Diagram»

На рисунках 5.5-5.8 показаны копии экранов вкладок «HLS Synthesis» для каждого ядра сборки Hardware.

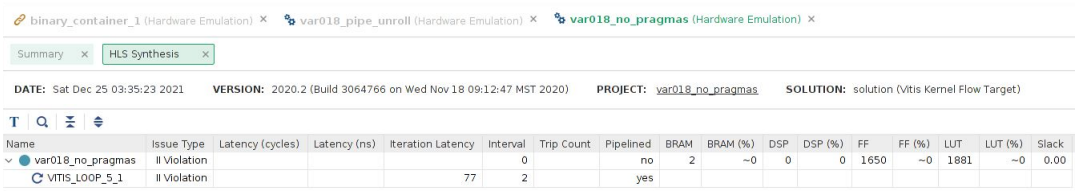


Рисунок 5.5 – Копия экрана вкладки «HLS Synthesis» для не оптимизированного цикла

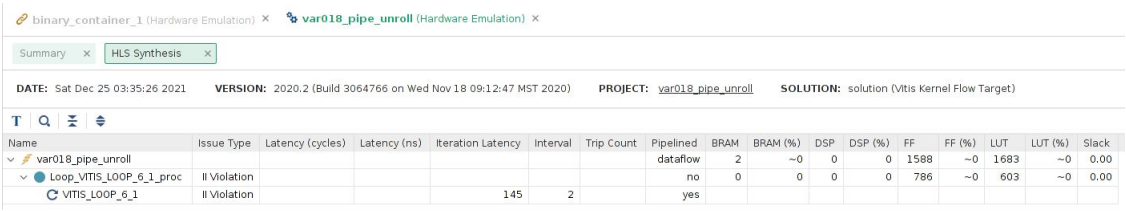


Рисунок 5.6 – Копия экрана вкладки «HLS Synthesis» для конвейерного и частично развернутого цикла

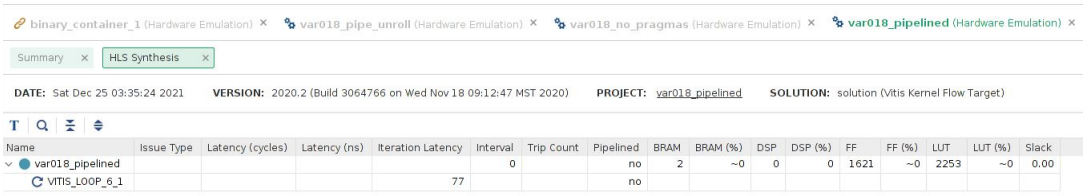


Рисунок 5.7 – Копия экрана вкладки «HLS Synthesis» для конвейерной организации цикла

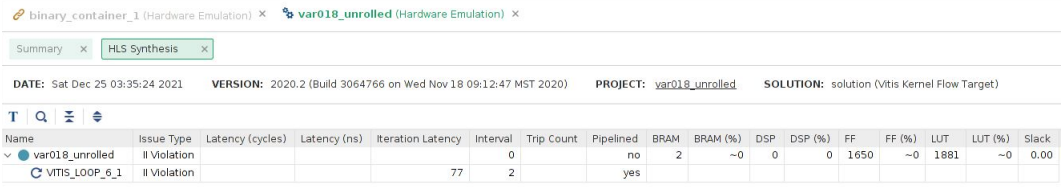


Рисунок 5.8 – Копия экрана вкладки «HLS Synthesis» для частично развернутого цикла

5.1 Обоснование результатов тестов и выводы

Из результатов работы приложения в режиме Hardware, показанных на риснуке 5.1, можно сделать вывод о том, что наибольшее время выполнения у не оптимизированного цикла. Наименьшее время выполнения было достигнуто у конвейерного и частично развернутого цикла, так как механизм опирается на представление вычислительных действий в виде многостадийного конвейера.

6 Контрольные вопросы

1. Назовите преимущества и недостатки аппаратных ускорителей на ПЛИС по сравнению с CPU и графическими ускорителями?

Преимущества аппаратных ускорителей на ПЛИС:

- низкая стоимость в сравнении с аппаратными ускорителями;
- большая частота эмуляции;
- компактность.

Недостатки аппаратных ускорителей на ПЛИС:

- необходимость перекомпиляции проекта и переконфигурации ПЛИС при любом исправлении содержимого проекта;
- наличие специализированного программного обеспечения для разделения модели микросхемы на части для загрузки в отдельные ПЛИС.

2. Назовите основные способы оптимизации циклических конструкций ЯВУ, реализуемых в виде аппаратных ускорителей?

Способы оптимизаций циклов:

- конвейерная обработка циклов;
- разворачивание циклов;
- потоковая обработка.

3. Назовите этапы работы программной части ускорителя в хост-системе?

Этапы работы программной части ускорителя в хост-системе:

- инициализация среды OpenCL;
- приложение создает три буфера, необходимых для обмена данными с ядром: два буфера для передачи исходных данных и один для вывода результата;

- запуск задачи на исполнение;
- после завершения работы всех команд выходной буфер содержит результаты работы ядра.

4. В чем заключается процесс отладки для вариантов сборки Emulation-SW, Emulation-HW и Hardware?

Для сборки Emulation-SW код ядра компилируется для работы на ЦПУ хост-системы. Этот вариант сборки служит для верификации совместного исполнения кода хост-системы и кода ядра, для выявления синтаксических ошибок, выполнения отладки на уровне исходного кода ядра, проверки поведения системы.

Для сборки Emulation-HW код ядра компилируется в аппаратную модель (RTL), которая запускается в специальном симуляторе на ЦПУ. Этот вариант сборки и запуска занимает больше времени, но обеспечивает подробное и точное представление активности ядра. Данный вариант сборки полезен для тестирования функциональности ускорителя и получения начальных оценок производительности.

Для сборки Hardware код ядра компилируется в аппаратную модель (RTL), а затем реализуется на FPGA. В результате формируется двоичный файл xclbin, который будет работать на реальной FPGA.

5. Какие инструменты и средства анализа результатов синтеза возможно использовать в Vitis HLS для оптимизации ускорителей?

Компилятор Xilinx Vitis v++ позволяет генерировать из описаний на языке C/C++ синтезируемые низкоуровневые RTL проекты, которые затем отображаются на структуру ПЛИС.

Заключение

В данной лабораторной работе были рассмотрены и изучены технологии синтеза аппаратных устройств ускорения вычислений по языкам высокого уровня. Цель, поставленная перед началом работы, была достигнута. В ходе лабораторной работы были решены все поставленные задачи.