



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	3
1 Основные теоретические сведения	4
1.1 Методология ускорения вычислений на основе ПЛИС	4
1.2 Оптимизация времени обработки и пропускной способности	5
1.2.1 Конвейерная обработка циклов	6
1.2.2 Разворачивание циклов	6
1.2.3 Поточковая обработка	6
2 Функции ядра на основе индивидуального задания	8
3 Режим Emulation-SW	10
4 Режим Emulation-HW	11
5 Режим Hardware	14
5.1 Обоснование результатов тестов и выводы	18
6 Контрольные вопросы	19
Заключение	21

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 высокоуровневого синтеза;
- выполнить его отладку.

1 Основные теоретические сведения

1.1 Методология ускорения вычислений на основе ПЛИС

Микропроцессоры и графические процессоры имеют predetermined 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.

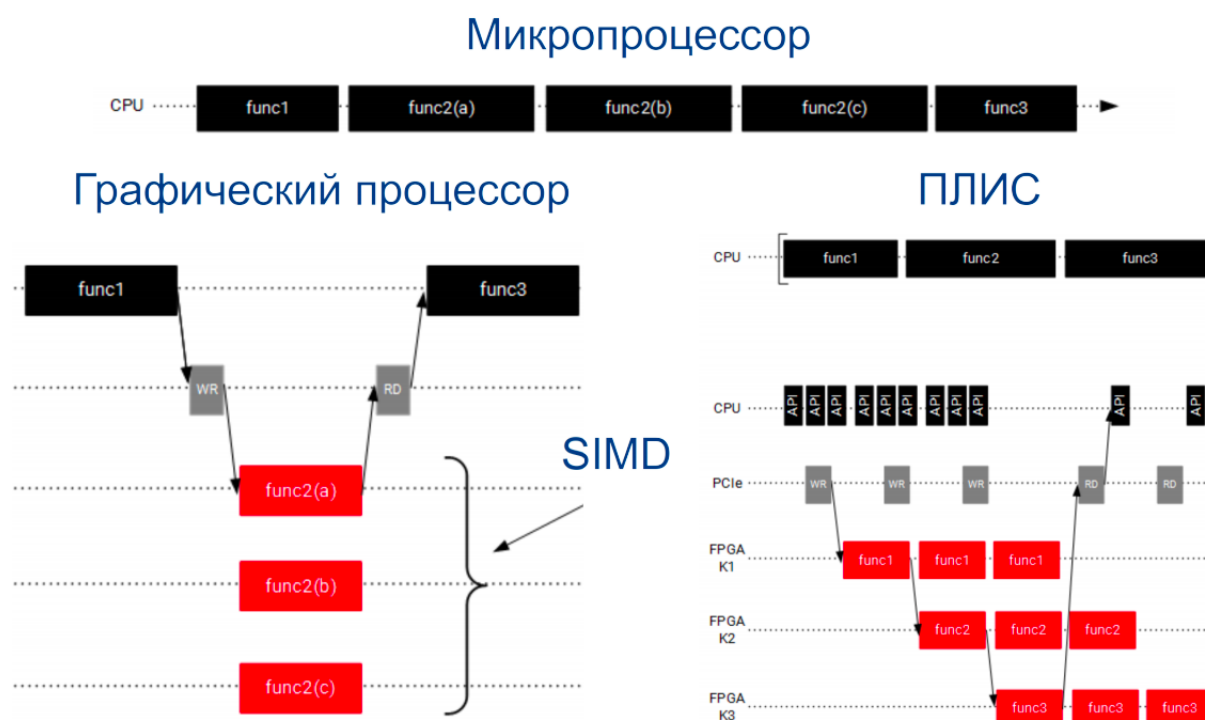


Рисунок 1.1 – Принципы организации вычислений на различных платформах

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.

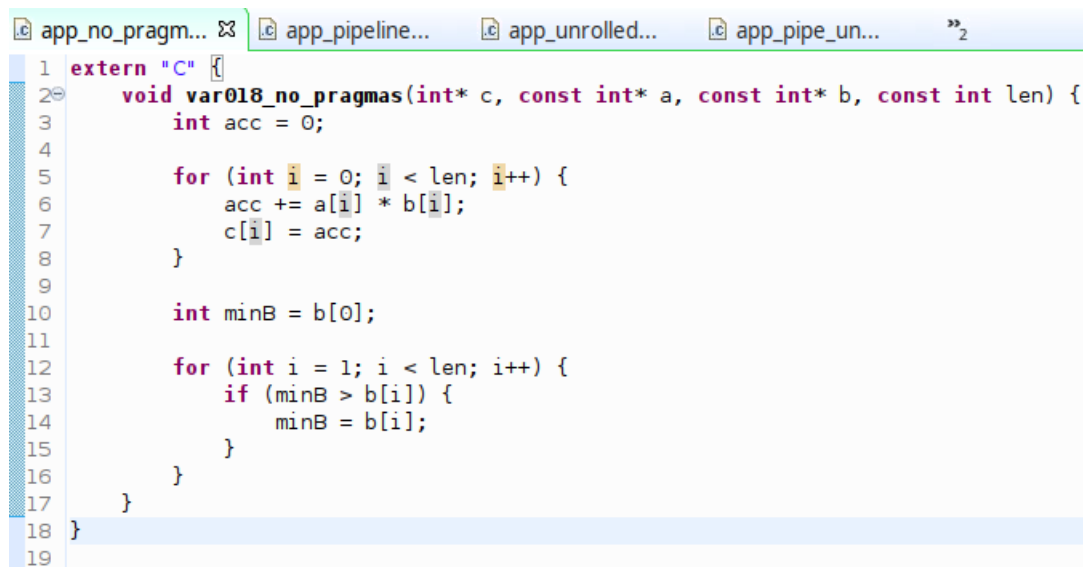
1.2.3 Потокковая обработка

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.

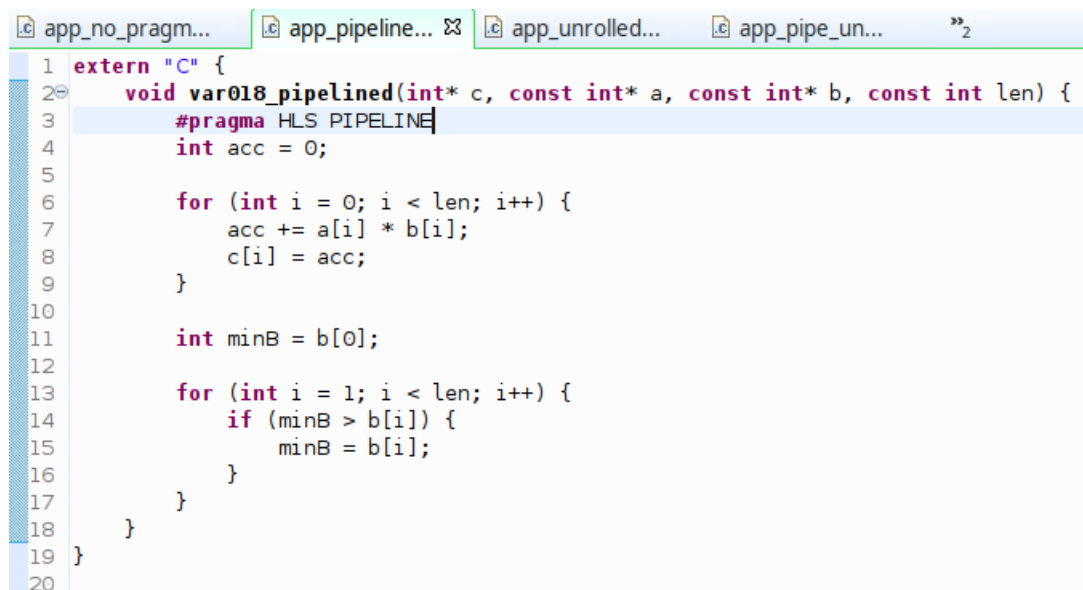
2 Функции ядра на основе индивидуального задания

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



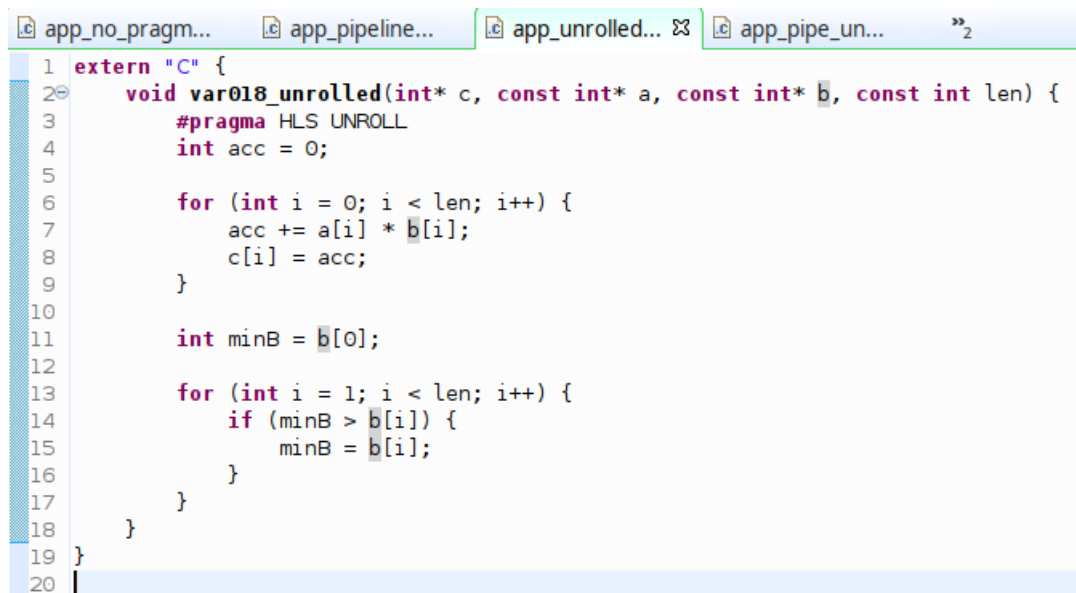
```
1 extern "C" {
2     void var018_no_pragmas(int* c, const int* a, const int* b, const int len) {
3         int acc = 0;
4
5         for (int i = 0; i < len; i++) {
6             acc += a[i] * b[i];
7             c[i] = acc;
8         }
9
10        int minB = b[0];
11
12        for (int i = 1; i 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



```
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

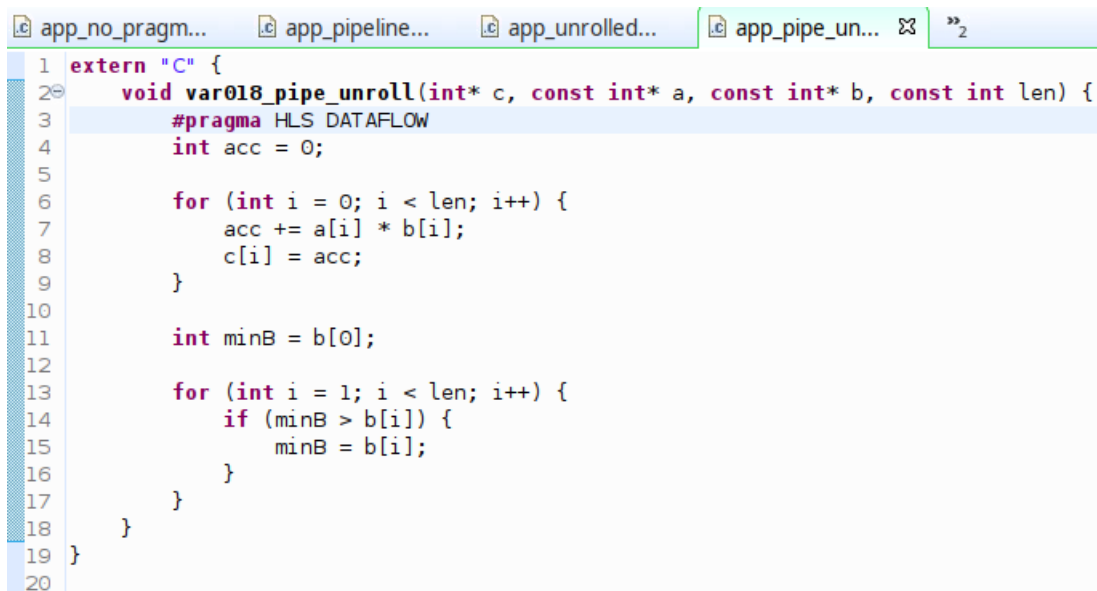


```

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



```

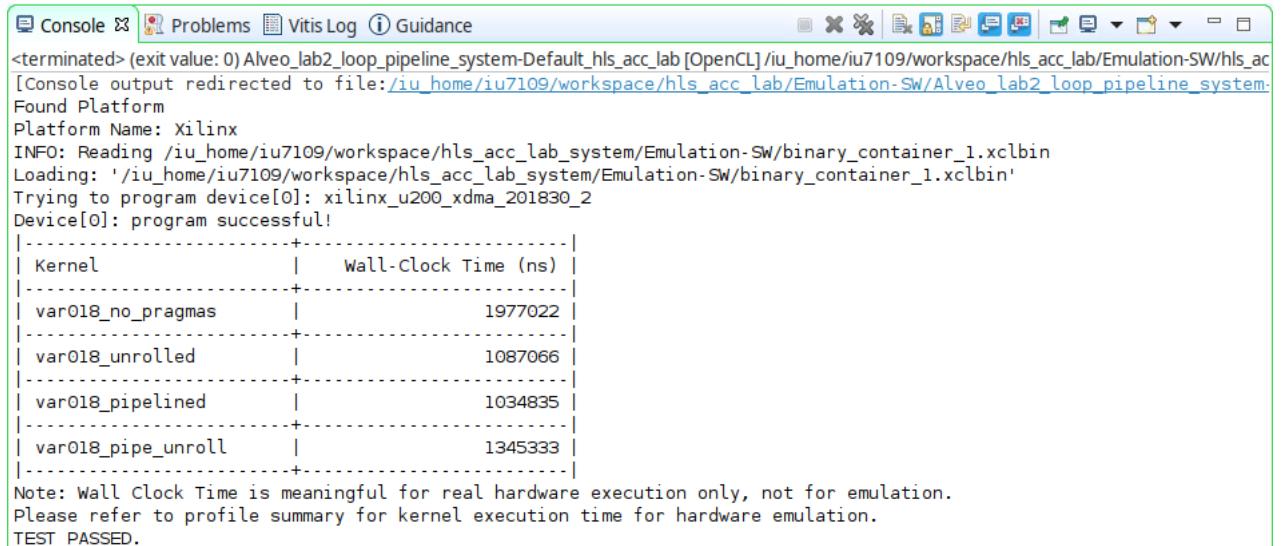
1 extern "C" {
2     void var018_pipe_unroll(int* c, const int* a, const int* b, const int len) {
3         #pragma HLS DATAFLOW
4         int acc = 0;
5
6         for (int i = 0; i < len; i++) {
7             acc += a[i] * b[i];
8             c[i] = acc;
9         }
10
11        int minB = b[0];
12
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 работы приложения в режиме Emulation-SW.



```
<terminated> (exit value: 0) Alveo_lab2_loop_pipeline_system-Default_hls_acc_lab [OpenCL] /iu_home/iu7109/workspace/hls_acc_lab/Emulation-SW/hls_ac
[Console output redirected to file:/iu_home/iu7109/workspace/hls_acc_lab/Emulation-SW/Alveo_lab2_loop_pipeline_system-
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7109/workspace/hls_acc_lab_system/Emulation-SW/binary_container_1.xclbin
Loading: '/iu_home/iu7109/workspace/hls_acc_lab_system/Emulation-SW/binary_container_1.xclbin'
Trying to program device[0]: xilinx_u200_xdma_201830_2
Device[0]: program successful!
|-----|
| Kernel | Wall-Clock Time (ns) |
|-----|
| var018_no_pragmas | 1977022 |
|-----|
| var018_unrolled | 1087066 |
|-----|
| var018_pipelined | 1034835 |
|-----|
| var018_pipe_unroll | 1345333 |
|-----|
Note: Wall Clock Time is meaningful for real hardware execution only, not for emulation.
Please refer to profile summary for kernel execution time for hardware emulation.
TEST PASSED.
```

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.

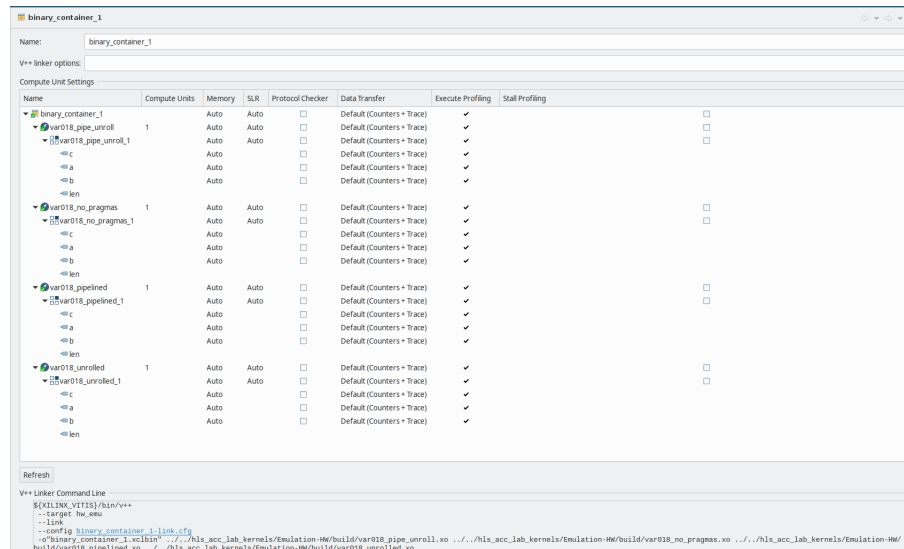


Рисунок 4.1 – Копия экрана Assistant View для сборки Emulation-HW

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

```

SystemDebugger_hls_acc_lab_system_hls_acc_lab [OpenCL] hls_acc_lab/Emulation-HW/hls_acc_lab (12/25/21, 4:25 AM)
Loading: '/iu_home/iu7109/workspace/hls_acc_lab_system/Emulation-HW/binary_container_1.xclbin'
Trying to program device[0]: xilinx_u200_xdma_201830_2
INFO: [HW-EMU 01] Hardware emulation runs simulation underneath. Using a large data set will result in long simulati
INFO::[ Vitis-EM 22 ] [Time elapsed: 3 minute(s) 11 seconds, Emulation time: 0.0881827 ms]
Data transfer between kernel(s) and global memory(s)
var018_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 0.000 KB          WR = 0.000 KB
var018_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 0.000 KB          WR = 0.000 KB
var018_pipelined_1:m_axi_gmem-DDR[1]          RD = 0.000 KB          WR = 0.000 KB
var018_unrolled_1:m_axi_gmem-DDR[1]           RD = 0.000 KB          WR = 0.000 KB

Device[0]: program successful!
|-----+-----|
| Kernel                  | Wall-Clock Time (ns) |
|-----+-----|
| var018_no_pragmas       | 23014067251          |
|-----+-----|
| var018_unrolled         | 23012120868          |
|-----+-----|
INFO::[ Vitis-EM 22 ] [Time elapsed: 8 minute(s) 2 seconds, Emulation time: 0.245644 ms]
Data transfer between kernel(s) and global memory(s)
var018_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR = 4.000 KB
var018_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 0.000 KB          WR = 0.000 KB
var018_pipelined_1:m_axi_gmem-DDR[1]          RD = 2.023 KB          WR = 1.000 KB
var018_unrolled_1:m_axi_gmem-DDR[1]           RD = 8.000 KB          WR = 4.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 13 minute(s) 4 seconds, Emulation time: 0.412531 ms]
Data transfer between kernel(s) and global memory(s)
var018_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR = 4.000 KB
var018_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 0.000 KB          WR = 0.000 KB
var018_pipelined_1:m_axi_gmem-DDR[1]          RD = 7.102 KB          WR = 3.500 KB
var018_unrolled_1:m_axi_gmem-DDR[1]           RD = 8.000 KB          WR = 4.000 KB

| var018_pipelined        | 479218715067         |
|-----+-----|
| var018_pipe_unroll      | 34019643644          |
|-----+-----|
Note: Wall Clock Time is meaningful for real hardware execution only, not for emulation.
Please refer to profile summary for kernel execution time for hardware emulation.
TEST PASSED.
INFO::[ Vitis-EM 22 ] [Time elapsed: 18 minute(s) 10 seconds, Emulation time: 0.59378 ms]
Data transfer between kernel(s) and global memory(s)
var018_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR = 4.000 KB
var018_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 8.000 KB          WR = 4.000 KB
var018_pipelined_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR = 4.000 KB
var018_unrolled_1:m_axi_gmem-DDR[1]           RD = 8.000 KB          WR = 4.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 18 minute(s) 10 seconds, Emulation time: 0.593787 ms]
Data transfer between kernel(s) and global memory(s)
var018_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR = 4.000 KB
var018_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 8.000 KB          WR = 4.000 KB
var018_pipelined_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR = 4.000 KB
var018_unrolled_1:m_axi_gmem-DDR[1]           RD = 8.000 KB          WR = 4.000 KB

```

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.

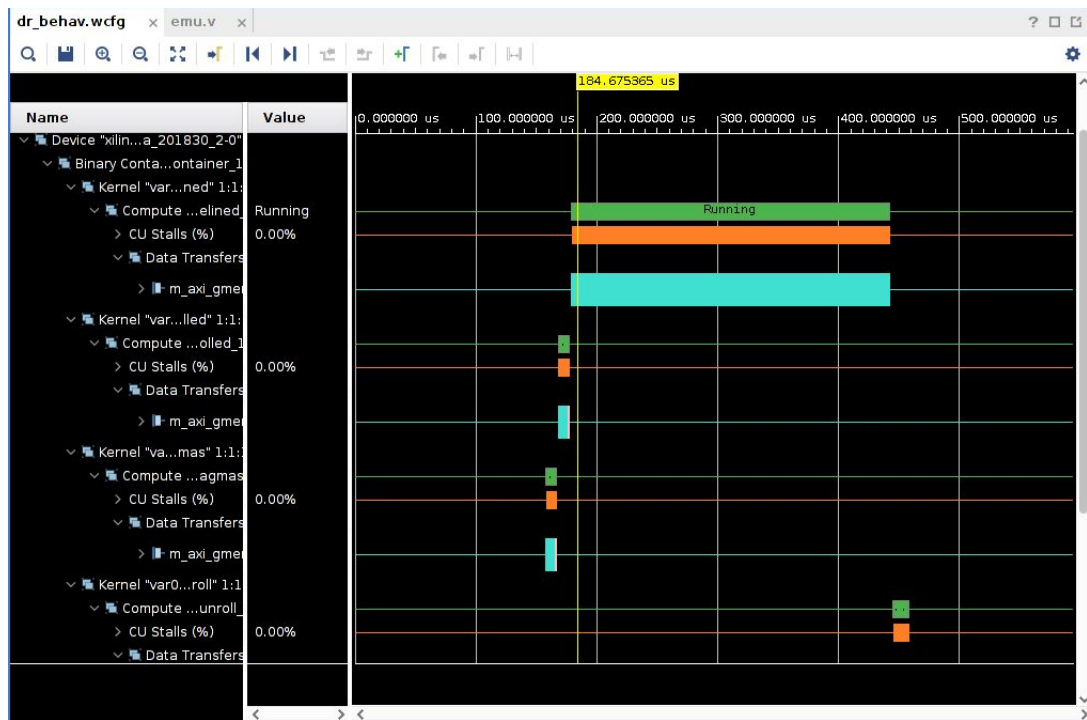


Рисунок 4.3 – Окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW - 1

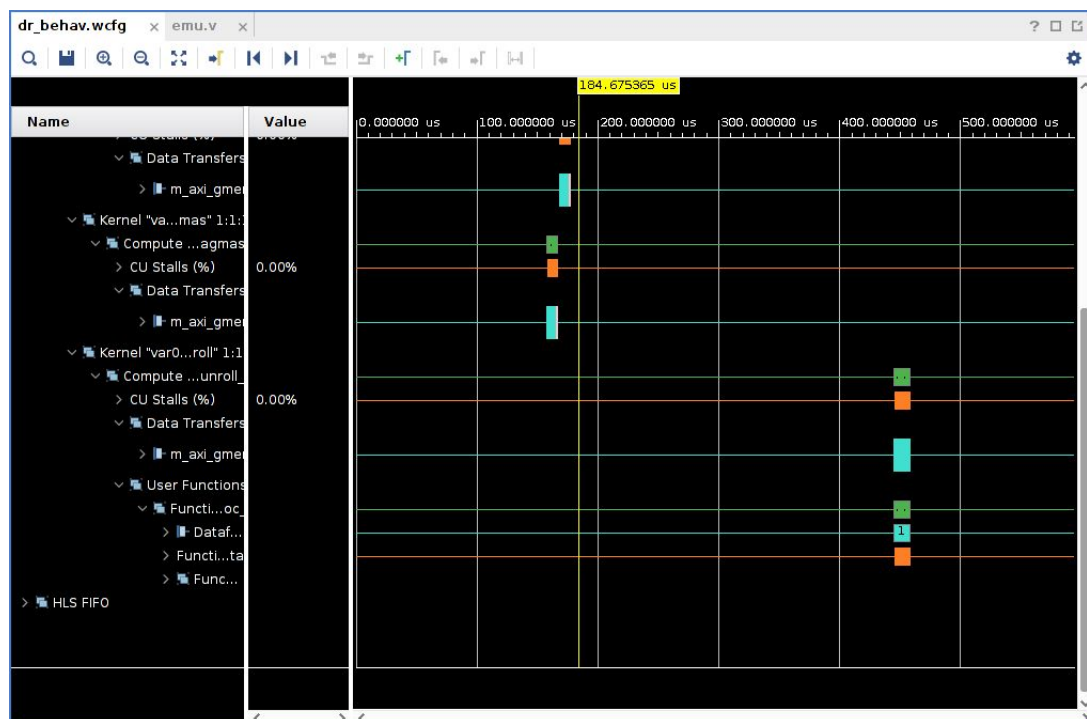
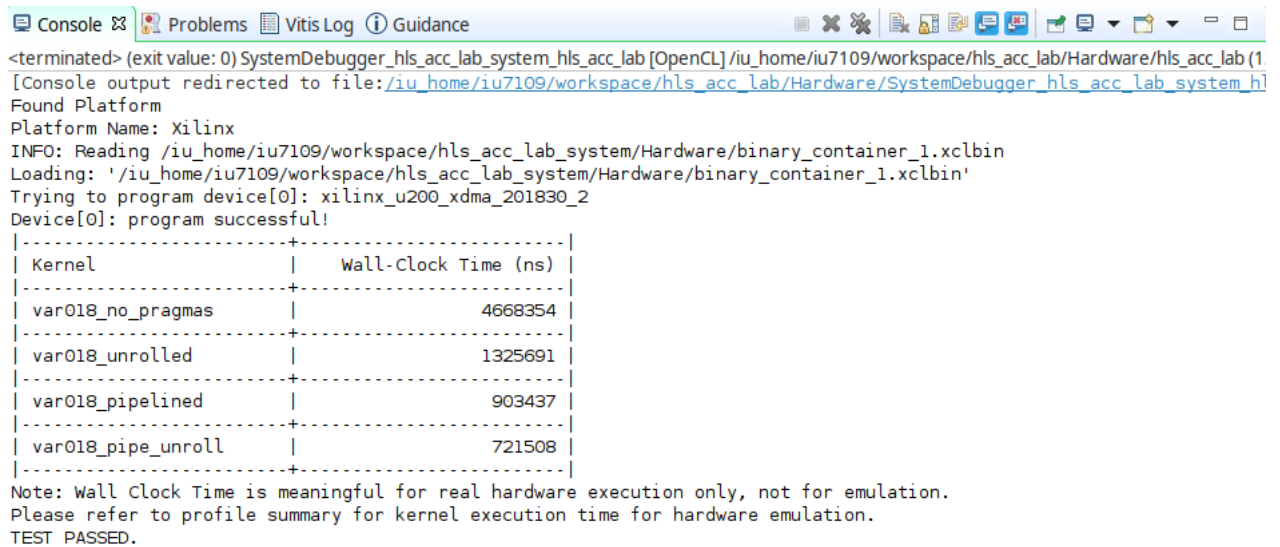


Рисунок 4.4 – Окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW - 2

5 Режим Hardware

На рисунке 5.1 показаны результаты работы приложения в режиме Hardware.



The screenshot shows the Vitis IDE console with the following text:

```
<terminated> (exit value: 0) SystemDebugger_hls_acc_lab_system_hls_acc_lab [OpenCL] /iu_home/iu7109/workspace/hls_acc_lab/Hardware/hls_acc_lab (1.
[Console output redirected to file:/iu_home/iu7109/workspace/hls_acc_lab/Hardware/SystemDebugger_hls_acc_lab_system_h
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7109/workspace/hls_acc_lab_system/Hardware/binary_container_1.xclbin
Loading: '/iu_home/iu7109/workspace/hls_acc_lab_system/Hardware/binary_container_1.xclbin'
Trying to program device[0]: xilinx_u200_xdma_201830_2
Device[0]: program successful!
|-----+-----|
| Kernel | Wall-Clock Time (ns) |
|-----+-----|
| var018_no_pragmas | 4668354 |
|-----+-----|
| var018_unrolled | 1325691 |
|-----+-----|
| var018_pipelined | 903437 |
|-----+-----|
| var018_pipe_unroll | 721508 |
|-----+-----|
Note: Wall Clock Time is meaningful for real hardware execution only, not for emulation.
Please refer to profile summary for kernel execution time for hardware emulation.
TEST PASSED.
```

Kernel	Wall-Clock Time (ns)
var018_no_pragmas	4668354
var018_unrolled	1325691
var018_pipelined	903437
var018_pipe_unroll	721508

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».

binary_container_1 (Hardware Emulation)

Summary

×

binary_container_1

/u_home/lu7109/workspace/hls_acc_lab_system_hw_link/Emu

STATUS

Completed

Log

GUIDANCE

7 warnings

System Guidance

CLOCK FREQUENCIES

KERNEL_CLK

500 Mhz

DATA_CLK

300 Mhz

VERSION

Vitis V++ Compiler Release 2020.2, SW Build (by xbuild) on 2020-11-18-05:13:29

STARTED

December 25, 2021 03:39

COMPLETED

December 25, 2021 04:04

PLATFORM

xilinx_u200_xdma_201830_2
Platform Diagram

KERNELS

System Diagram

var018_pipe_unroll
1 compute unit
Compile Summary

var018_no_pragmas
1 compute unit
Compile Summary

var018_pipelined
1 compute unit
Compile Summary

var018_unrolled
1 compute unit
Compile Summary

Рисунок 5.2 – Копия экрана вкладки «Summary»

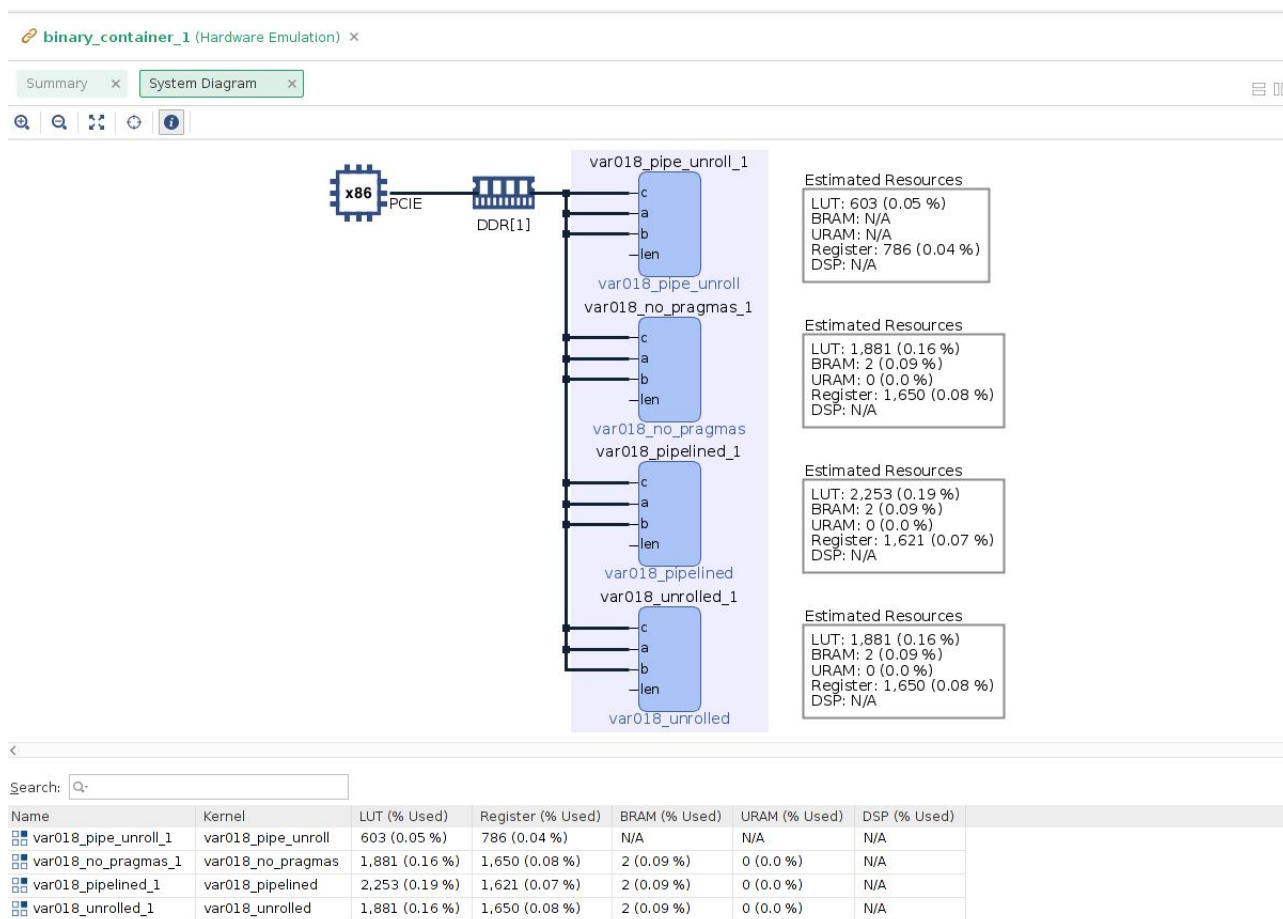


Рисунок 5.3 – Копия экрана вкладки «System Diagram»

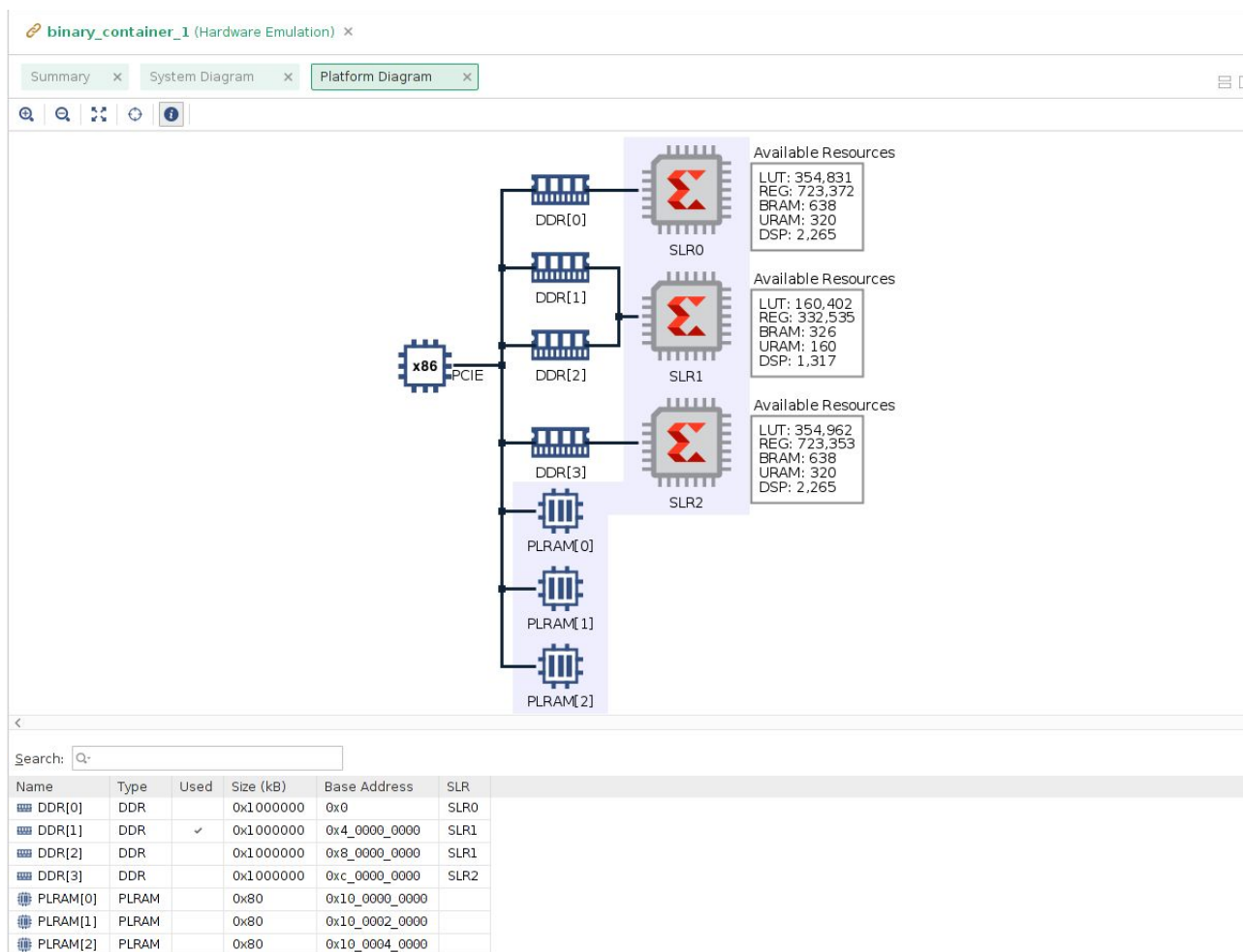


Рисунок 5.4 – Копия экрана вкладки «Platform Diagram»

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

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var018_no_pragmas	II Violation				0		no	2	~0	0	0	1650	~0	1881	~0	0.00
VTIS_LOOP_5_1	II Violation			77	2		yes									

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

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var018_pipe_unroll							dataflow	2	~0	0	0	1588	~0	1683	~0	0.00
Loop_VITIS_LOOP_6_1_proc	II Violation						no	0	0	0	0	786	~0	603	~0	0.00
VITIS_LOOP_6_1	II Violation				145	2	yes									

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

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var018_pipelined					0		no	2	~0	0	0	1621	~0	2253	~0	0.00
Loop_VITIS_LOOP_6_1_proc	II Violation						no									
VITIS_LOOP_6_1	II Violation				77		no									

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

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var018_unrolled					0		no	2	~0	0	0	1650	~0	1881	~0	0.00
Loop_VITIS_LOOP_6_1_proc	II Violation						no									
VITIS_LOOP_6_1	II Violation				77	2	yes									

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 рисунке 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.