



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»(ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.04 «Программная инженерия»

О Т Ч Е Т по лабораторной работе №5

Название Разработка ускорителей вычислений средствами САПР высокого

синтеза Xilinx Vitis HLS Alveo

Дисциплина Архитектура электронно-вычислительных машин

Студент:

Козлова И. В.

Фамилия, И.О.

Преподаватель:

Попов А. Ю.

подпись, дата

Фамилия, И. О.

Оглавление

Цель работы	2
1 Основные теоретические сведения	3
Основные теоретические сведения	3
1.1 Методология ускорения вычислений на основе ПЛИС	3
Вариант индивидуального задания	5
Файлы функций ядра на основе индивидуального задания	6
Результаты работы приложения в режиме Emulation-SW	8
Копия экрана Assistant View для сборки Emulation-HW	9
Результаты работы приложения в режиме Emulation-HW	10
Окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW	11
Результаты работы приложения в режиме Hardware	12
Копии вкладок для сборки Hardware	13
Обоснование результатов тестов и выводы.	20
Контрольные вопросы	21
Заключение	23

Цель работы

Изучение методики и технологии синтеза аппаратных устройств ускорения вычислений по описаниям на языках высокого уровня.

Для достижения поставленной цели необходимо выполнить следующие задачи:

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

1 Основные теоретические сведения

1.1 Методология ускорения вычислений на основе ПЛИС

Ускорение вычислительных алгоритмов с использованием программируемых логических интегральных схем (ПЛИС) имеет ряд преимуществ по сравнению с их реализацией на универсальных микропроцессорах, или графических процессорах. В то время, как традиционная разработка программного обеспечения связана с программированием на заранее определенном наборе машинных команд, разработка программируемых устройств - это создание специализированной вычислительной структуры для реализации желаемой функциональности.

На рисунке 1.1 представлены принципы организации вычислений на различных платформах.

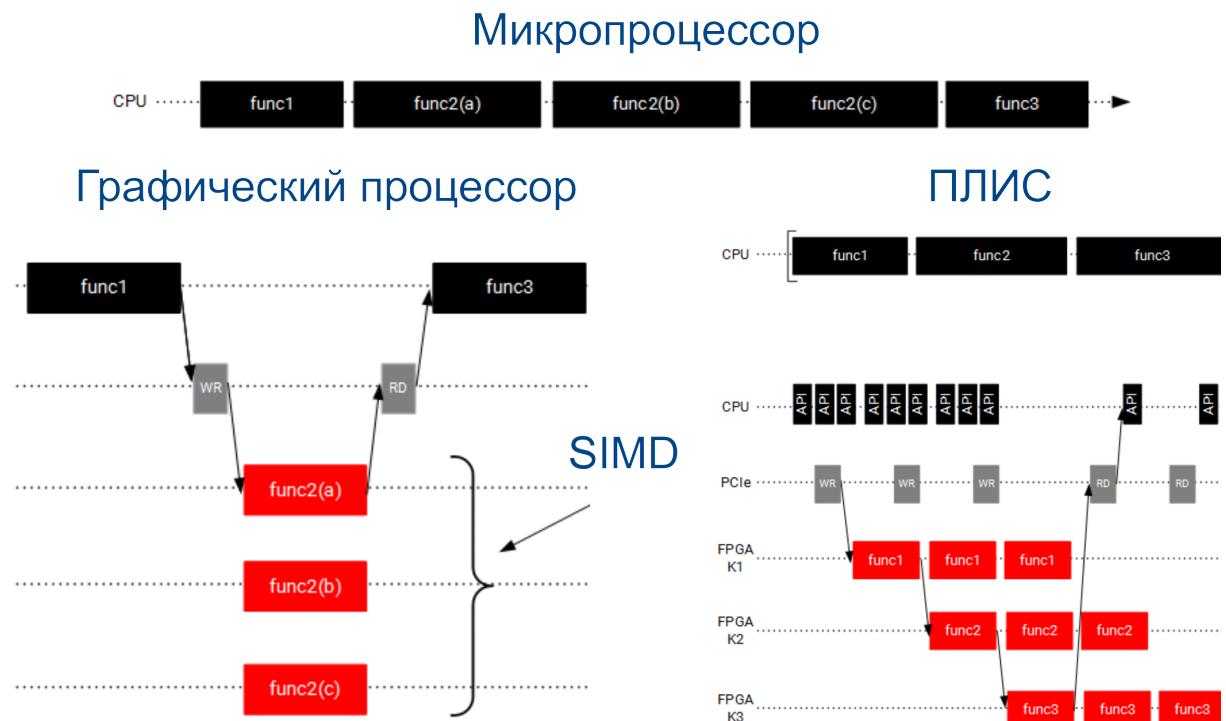


Рисунок 1.1 – Принципы организации вычислений на различных платформах

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

виде трех этапов:

1. Создание архитектуры приложения.
2. Разработка ядра аппаратного ускорителя на языках C/C++.
3. Анализ производительности и выявление способов ее повышения.

Вариант индивидуального задания

В листинге 1.1 представлен индивидуальный вариант (по списку 9).

Листинг 1.1 – Индивидуальный вариант 9

```
1 extern "C" {
2     void var009(int* c, const int* a, const int* b, const int len)
3     {
4         int meanA = 0;
5         int meanB = 0;
6         for (int i = 0; i < len; i++) {
7             meanA += a[i];
8             meanB += b[i];
9         }
10        for (int i = 0; i < len; i+=2) {
11            c[i] = meanA;
12            c[i+1] = meanB;
13        }
14    }
```

Файлы функций ядра на основе индивидуального задания

В листинге 1.2 представлен не оптимизированный цикл на основе индивидуального задания.

Листинг 1.2 – Индивидуальный вариант 9

```
1 extern "C" {
2     void var009(int* c, const int* a, const int* b, const int
3                 len) {
4         int meanA = 0;
5         int meanB = 0;
6         for (int i = 0; i < len; i++) {
7             meanA += a[i];
8             meanB += b[i];
9         }
10        for (int i = 0; i < len; i+=2) {
11            c[i] = meanA;
12            c[i+1] = meanB;
13        }
14    }
```

В листинге 1.3 представлен конвейерная организация цикла на основе индивидуального задания.

Листинг 1.3 – Индивидуальный вариант 9

```
1 extern "C" {
2     void var009_pipelined(int* c, const int* a, const int* b,
3                            const int len) {
4         #pragma HLS PIPELINE
5         int meanA = 0; int meanB = 0;
6         for (int i = 0; i < len; i++) {
7             meanA += a[i]; meanB += b[i];
8         }
9         for (int i = 0; i < len; i+=2) {
10            c[i] = meanA; c[i+1] = meanB;
11        }
12    }
```

В листинге 1.4 представлен частично развернутый цикл на основе индивидуального задания.

Листинг 1.4 – Индивидуальный вариант 9

```
1 extern "C" {
2     void var009_unrolled(int* c, const int* a, const int* b, const
3         int len) {
4         #pragma HLS UNROLL
5         int meanA = 0; int meanB = 0;
6         for (int i = 0; i < len; i++) {
7             meanA += a[i]; meanB += b[i];
8         }
9         for (int i = 0; i < len; i+=2) {
10            c[i] = meanA;
11            c[i+1] = meanB;
12        }
13 }
```

В листинге 1.5 представлен конвейерный и частично развернутый цикл на основе индивидуального задания.

Листинг 1.5 – Индивидуальный вариант 9

```
1 extern "C" {
2     void var009_pipe_unroll(int* c, const int* a, const int* b,
3         const int len) {
4         #pragma HLS DATAFLOW
5         int meanA = 0;
6         int meanB = 0;
7         for (int i = 0; i < len; i++) {
8             meanA += a[i]; meanB += b[i];
9         }
10        for (int i = 0; i < len; i+=2) {
11            c[i] = meanA; c[i+1] = meanB;
12        }
13 }
```

Результаты работы приложения в режиме Emulation-SW

На рисунке 1.2 представлены результаты работы приложения в режиме Emulation-SW.

```
[Console output redirected to file:/iu_home/iu7039/workspace/p2/his_acc_lab/Emulation-SW/
Alveo_lab2_loop_pipeline_system-Default_his_acc_lab.launch.log]
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7039/workspace/p2/his_acc_lab_system/Emulation-SW/binary_container_1.xclbin
Loading: '/iu_home/iu7039/workspace/p2/his_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) |
+-----+
| var009_no_pragmas | 2349771 |
+-----+
| var009_urolloed | 3779410 |
+-----+
| var009_pipelined | 998985 |
+-----+
| var009_pipe_unroll | 2988039 |
+-----+
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 1.2 – Результаты работы приложения в режиме Emulation-SW

Копия экрана Assistant View для сборки Emulation-HW

На рисунке 1.3 представлена копия экрана Assistant View для сборки Emulation-HW.

binary_container_1														
Name:	binary_container_1													
V++ linker options:														
Compute Unit Settings														
Name	Compute Units	Memory	SLR	Protocol Checker	Data Transfer	Execute Profiling	Stall Profiling							
binary_container_1	Auto	Auto	<input type="checkbox"/>		Default (Counters + Trace)	✓	<input type="checkbox"/>							
var09_no_pragmas	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
var09_no_pragmas_1		Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
c		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
a		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
b		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
len														
var09_pipe_unroll	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
var09_pipe_unroll_1		Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
c		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
a		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
b		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
len														
var09_pipelined	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
var09_pipelined_1		Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
c		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
a		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
b		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
len														
var09_unrolled	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
var09_unrolled_1		Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
c		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
a		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
b		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>							
len														

Рисунок 1.3 – Копия экрана Assistant View для сборки Emulation-HW

Результаты работы приложения в режиме Emulation-HW

На рисунке 1.4 представлены результаты работы приложения в режиме Emulation-HW.

```
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7039/workspace/p2/his_acc_lab_system/Emulation-HW/binary_container_1.xclbin
Loading: '/iu_home/iu7039/workspace/p2/his_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 simulation times. It is recommended that a small dataset is used for faster execution. The flow uses
approximate models for DDR memory and interconnect and hence the performance data generated is
approximate.
INFO::[ Vitis-EM 22 ] [Time elapsed: 3 minute(s) 17 seconds, Emulation time: 0.0928636 ms]
Data transfer between kernel(s) and global memory(s)
var009_no_pragmas_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var009_pipe_unroll_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var009_pipelined_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var009_unrolled_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB

Device[0]: program successful!
+-----+-----+
| Kernel | Wall-Clock Time (ns) |
+-----+-----+
| var009_no_pragmas | 26007526577 |
+-----+-----+
| var009_unrolled | 26012330694 |
+-----+-----+
INFO::[ Vitis-EM 22 ] [Time elapsed: 8 minute(s) 18 seconds, Emulation time: 0.248408 ms]
Data transfer between kernel(s) and global memory(s)
var009_no_pragmas_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB
var009_pipe_unroll_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var009_pipelined_1:m_axi_gmem-DDR[1] RD = 4.281 KB WR = 0.000 KB
var009_unrolled_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 13 minute(s) 19 seconds, Emulation time: 0.403139 ms]
Data transfer between kernel(s) and global memory(s)
var009_no_pragmas_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB
var009_pipe_unroll_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var009_pipelined_1:m_axi_gmem-DDR[1] RD = 14.078 KB WR = 0.000 KB
var009_unrolled_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB

+-----+-----+
| var009_pipelined | 498152404808 |
+-----+-----+
| var009_pipe_unroll | 38013403948 |
+-----+-----+
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) 23 seconds, Emulation time: 0.55815 ms]
Data transfer between kernel(s) and global memory(s)
var009_no_pragmas_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB
var009_pipe_unroll_1:m_axi_gmem-DDR[1] RD = 8.000 KB WR = 4.000 KB
var009_pipelined_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB
var009_unrolled_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 19 minute(s) 20 seconds, Emulation time: 0.588043 ms]
Data transfer between kernel(s) and global memory(s)
var009_no_pragmas_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB
var009_pipe_unroll_1:m_axi_gmem-DDR[1] RD = 8.000 KB WR = 4.000 KB
var009_pipelined_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB
var009_unrolled_1:m_axi_gmem-DDR[1] RD = 16.000 KB WR = 4.000 KB

INFO: [HW-EMU 06-0] Waiting for the simulator process to exit
```

Рисунок 1.4 – Результаты работы приложения в режиме Emulation-HW

Окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW

На рисунке 1.5 представлено окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW.

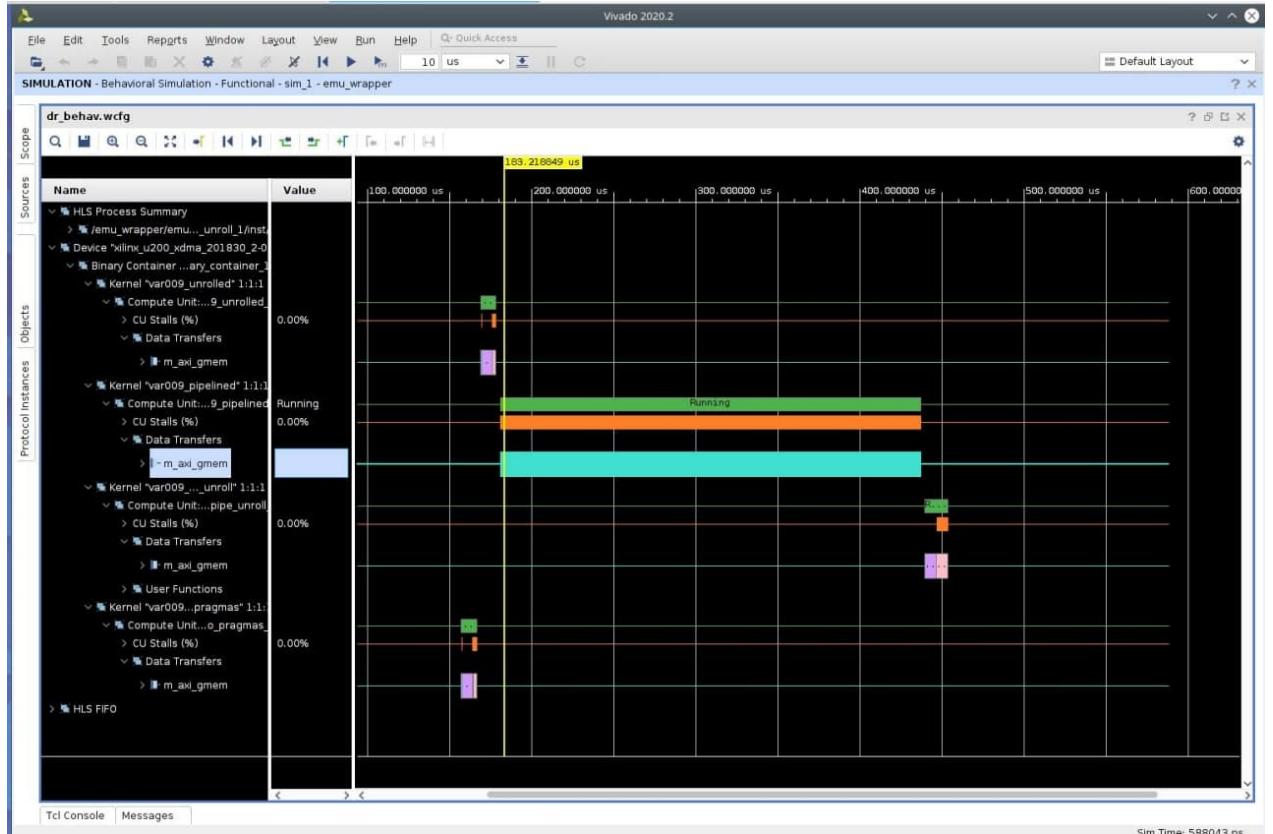


Рисунок 1.5 – Окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW

Результаты работы приложения в режиме Hardware

На рисунке 1.6 представлены результаты работы приложения в режиме Hardware.

```
[New Thread 0x7ffff5b2f700 (LWP 3478)]
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7039/workspace/p2/his_acc_lab_system_hw_link/Hardware/
binary_container_1.xclbin
Loading: '/iu_home/iu7039/workspace/p2/his_acc_lab_system_hw_link/Hardware/
binary_container_1.xclbin'
Trying to program device[0]: xilinx_u200_xdma_201830_2
[New Thread 0x7ffff4f2d700 (LWP 5362)]
Device[0]: program successful!
+-----+
| Kernel | Wall-Clock Time (ns) |
+-----+
[New Thread 0x7fffeb16f700 (LWP 5373)]
[New Thread 0x7ffffea96e700 (LWP 5377)]
[New Thread 0x7ffffea16d700 (LWP 5379)]
[New Thread 0x7ffffe996c700 (LWP 5380)]
[New Thread 0x7ffffe916b700 (LWP 5382)]
| var009_no_pragmas | 3925104 |
| var009_uropped | 184468 |
| var009_pipelined | 545361 |
| var009_pipe_unroll | 714374 |
+-----+
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.
[Thread 0x7ffffe916b700 (LWP 5382) exited]
[Thread 0x7ffffea16d700 (LWP 5379) exited]
[Thread 0x7ffffea96e700 (LWP 5377) exited]
[Thread 0x7ffffe996c700 (LWP 5380) exited]
[Thread 0x7fffeb16f700 (LWP 5373) exited]
[Thread 0x7ffff4f2d700 (LWP 5362) exited]
[Thread 0x7ffff5b2f700 (LWP 3478) exited]
```

Рисунок 1.6 – Результаты работы приложения в режиме Hardware

Копии вкладок для сборки Hardware

На рисунке 1.7 представлена копия экрана вкладки «Summary».

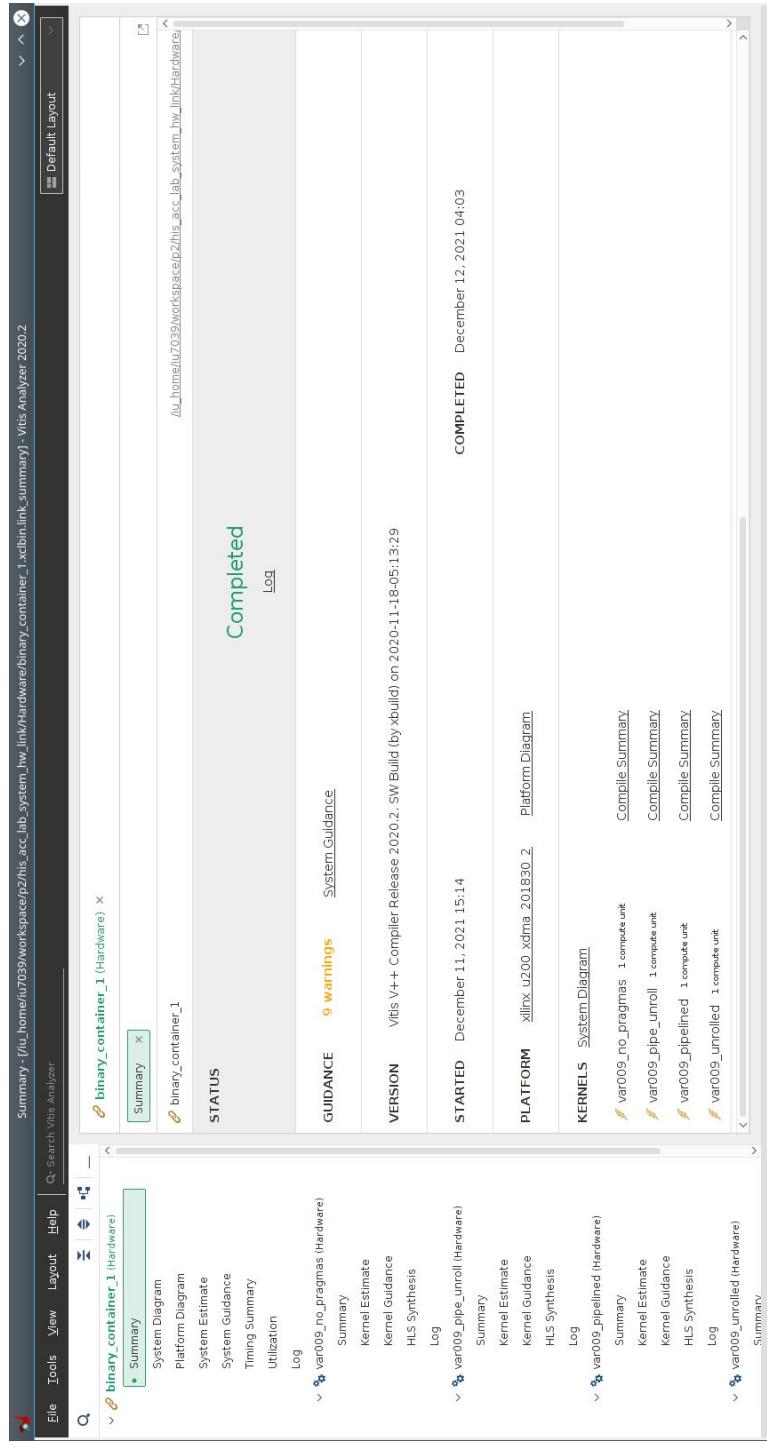


Рисунок 1.7 – Копия экрана вкладки «Summary»

На рисунке 1.8 представлена копия экрана вкладки «System Diagram».

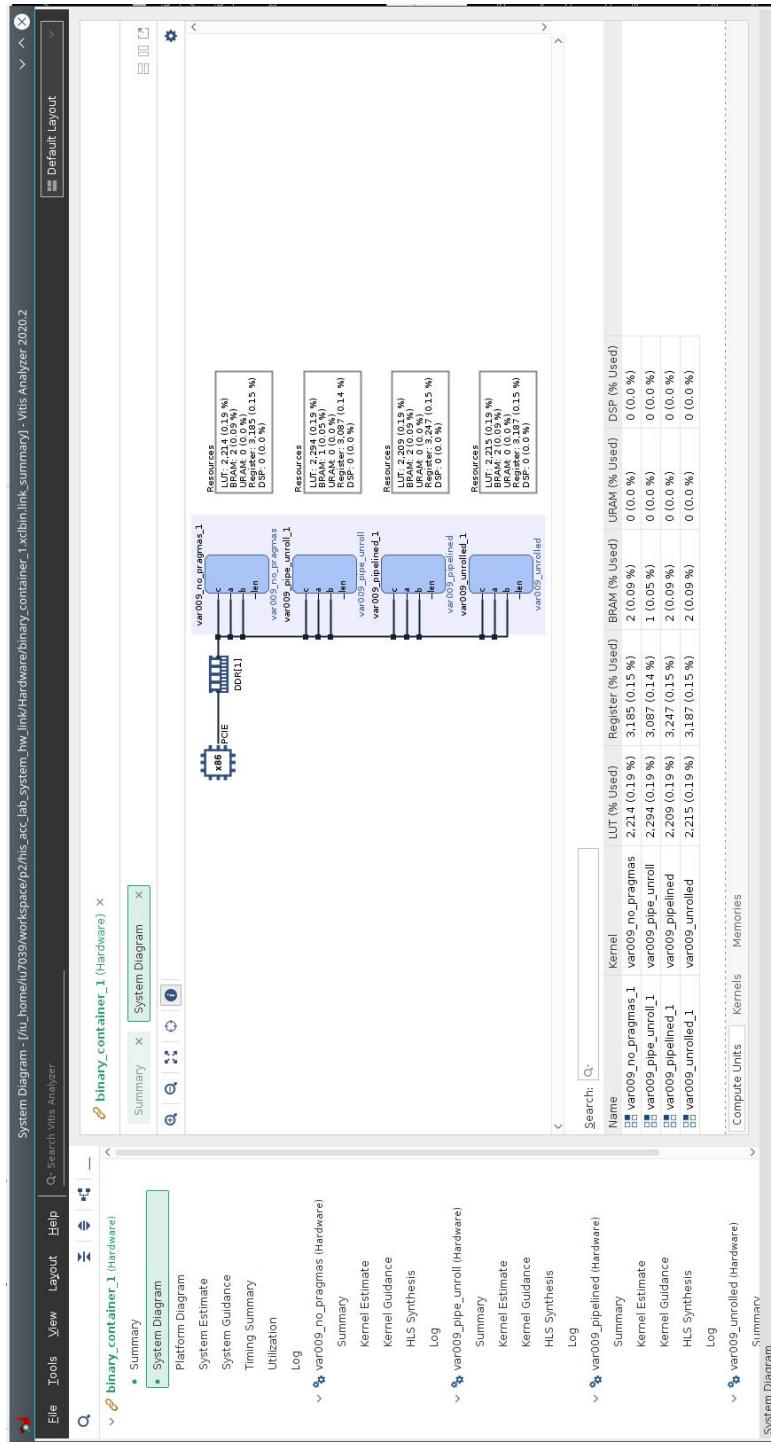


Рисунок 1.8 – Копия экрана вкладки «System Diagram»

На рисунке 1.9 представлена копия экрана вкладки «Platform Diagram».

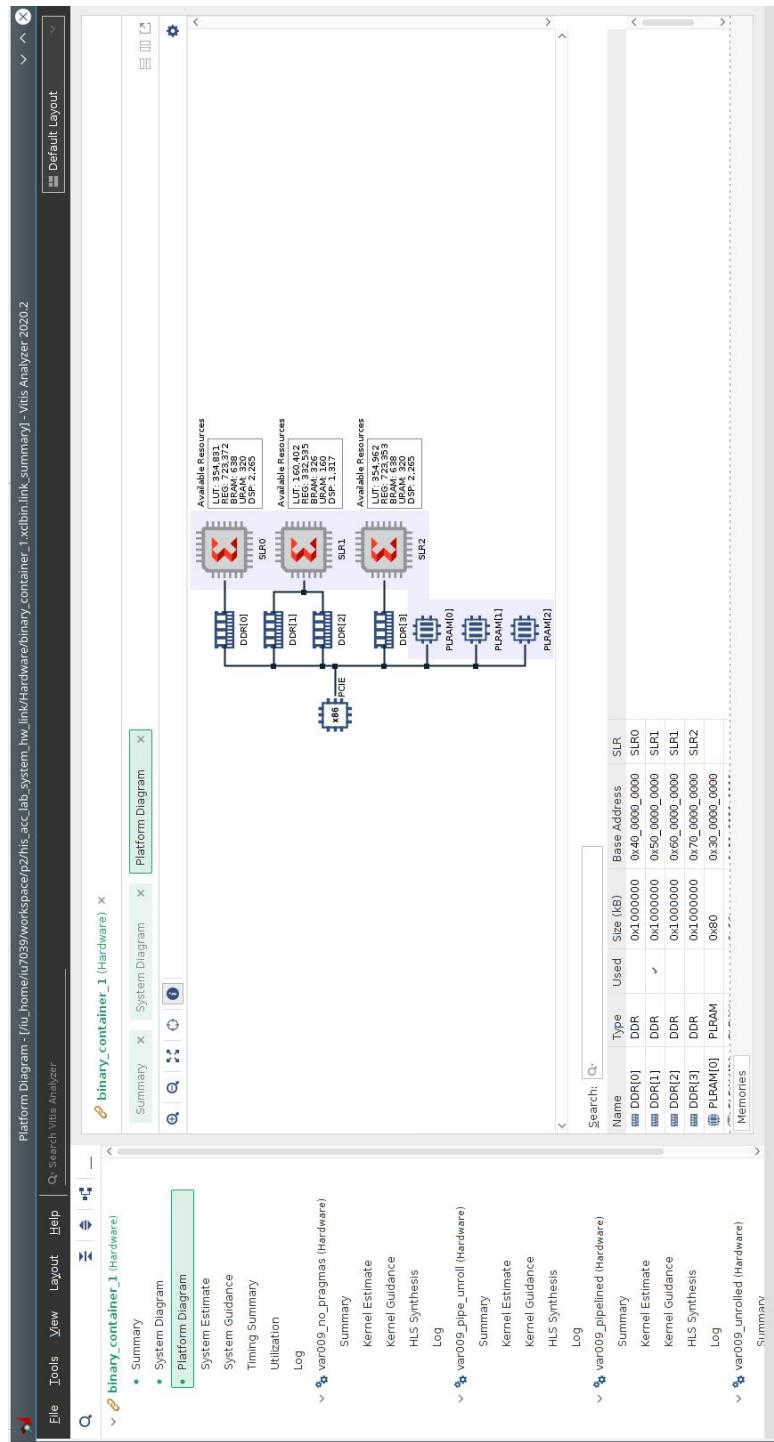


Рисунок 1.9 – Копия экрана вкладки «Platform Diagram»

На рисунке 1.10 представлена копия экрана вкладки «HLS Synthesis».

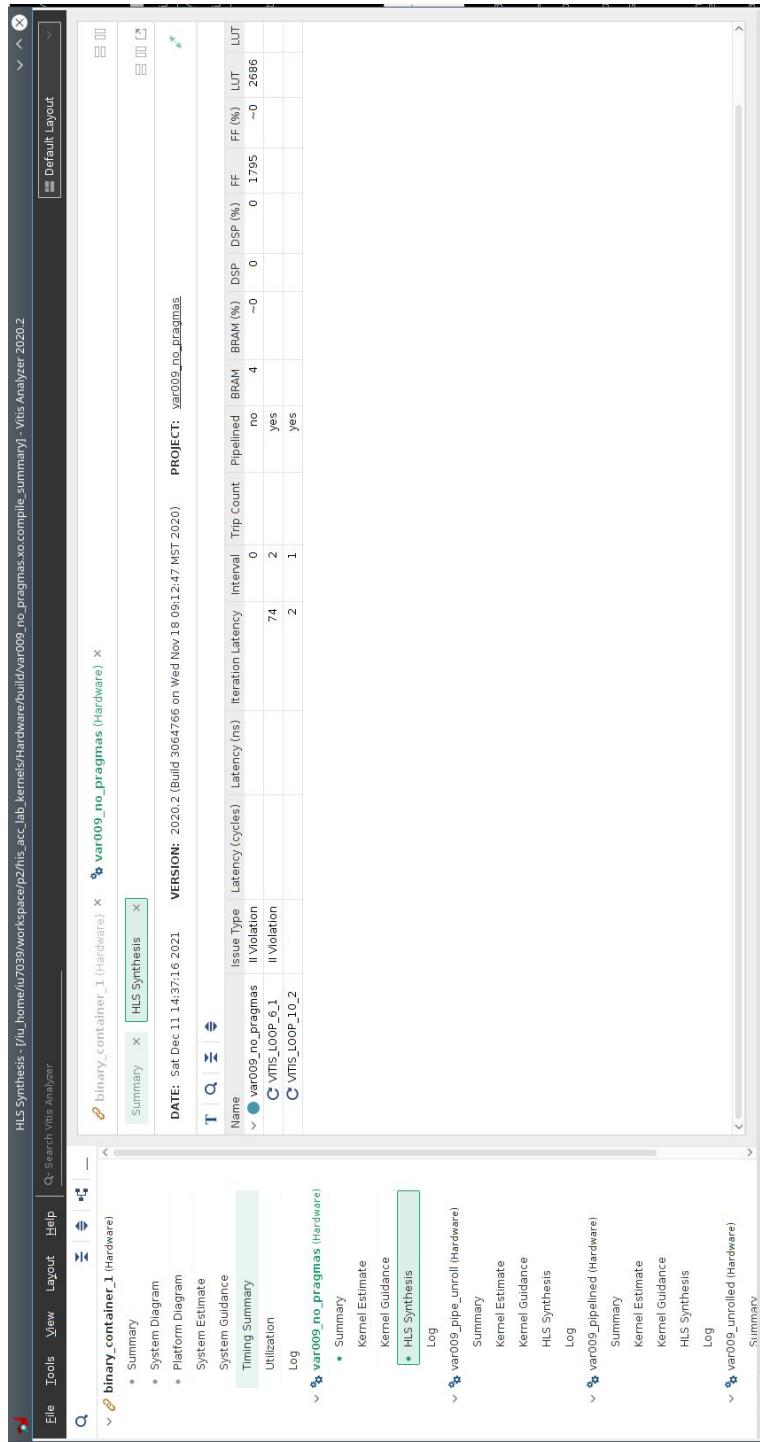


Рисунок 1.10 – Копия экрана вкладки «HLS Synthesis»

На рисунке 1.11 представлена копия экрана вкладки «HLS Synthesis».

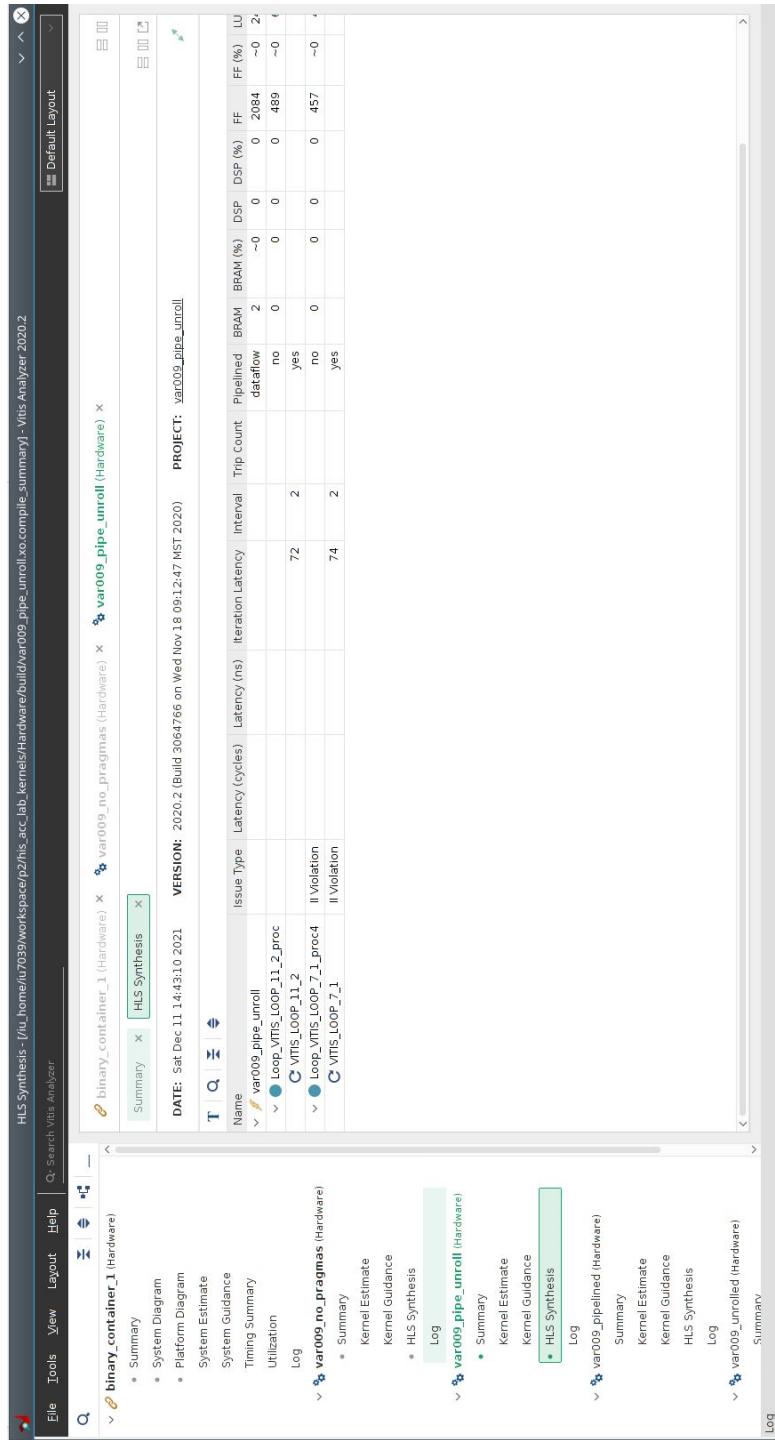


Рисунок 1.11 – Копия экрана вкладки «HLS Synthesis»

На рисунке 1.12 представлена копия экрана вкладки «HLS Synthesis».

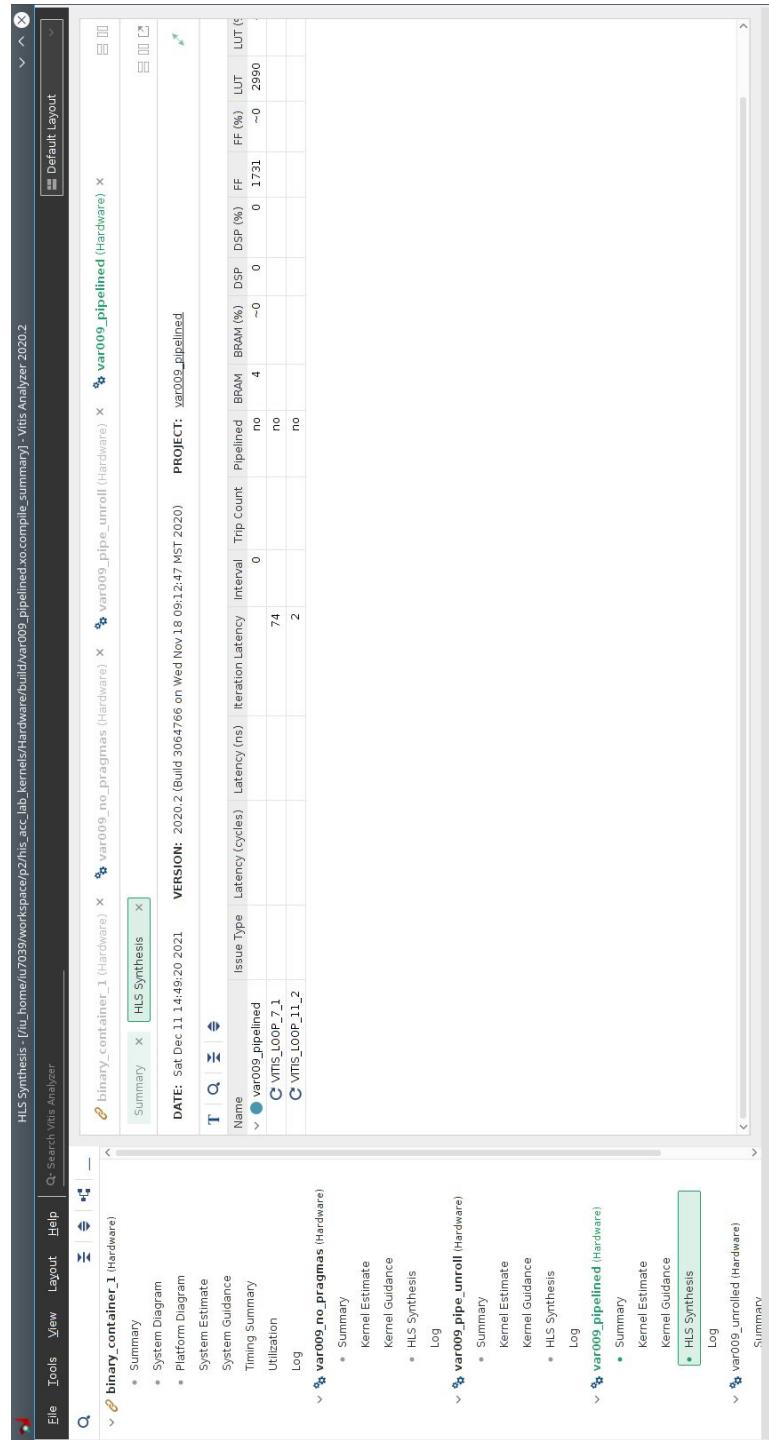


Рисунок 1.12 – Копия экрана вкладки «HLS Synthesis»

На рисунке 1.13 представлена копия экрана вкладки «HLS Synthesis».

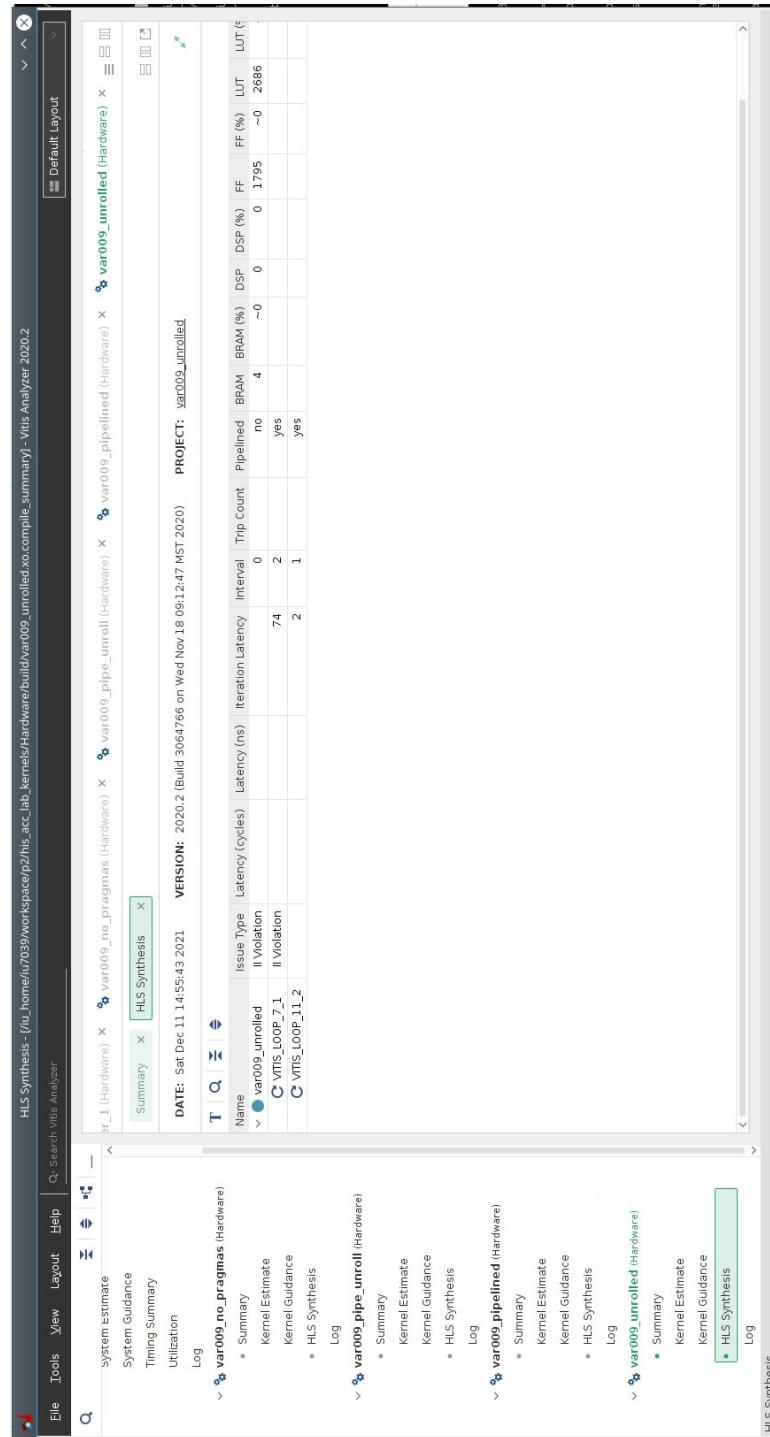


Рисунок 1.13 – Копия экрана вкладки «HLS Synthesis»

Обоснование результатов тестов и выводы.

Из рисунка 1.6 видно, что наибольшее время выполнения у цикла без оптимизаций.

Наименьшее время выполнения было достигнуто при частично развернутом цикле, так как в нем все развернутые итерации выполняются параллельно и количество итераций тем самым уменьшается.

Одновременное применение конвейеризации и частично развертывания тела цикла позволило ускорить обработку по сравнению с не оптимизированным циклом, однако уступило по времени организации, при которой использовалась только конвейерная организация. Вероятнее всего это вызвано неудачно подобранными параметрами развертывания.

Контрольные вопросы

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

Достоинствами данной системы являются:

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

Основными недостатками аппаратных эмуляторов на ПЛИС являются:

- 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?

1. Этап 1: Инициализируется среда OpenCL.
2. Этап 2. Приложение создает три буфера, необходимых для обмена данными с ядром: два буфера для передачи исходных данных и один для вывода результата (память должна быть выделена с выравниванием 4 КБ).

3. Этап 3. Запуск задачи на исполнение.
4. Этап 4: После завершения работы всех команд выходной буфер R_buf содержит результаты работы ядра.

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

- Программная эмуляция (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.
- Аппаратная эмуляция (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.
- Аппаратное обеспечение (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++ является одним из наиболее удачных проектов в этой области, и позволяет генерировать из описаний на языке C/C++ синтезируемые низкоуровневые RTL проекты, которые затем отображаются на структуру ПЛИС.

Заключение

Изучены методики и технологии синтеза аппаратных устройств ускорения вычислений по описаниям на языках высокого уровня.

Были выполнены следующие задачи:

- рассмотреть маршрут проектирования устройств, представленных в виде синтаксических конструкций ЯВУ C/C++;
- изучены принципы работы IDE Xilinx Vitis HLS;
- изучена методика анализа и отладки устройств;
- разработан ускоритель вычислений по индивидуальному заданию;
- разработан код для тестирования ускорителя;
- реализован ускоритель с помощью средств высоко-уровненного синтеза, выполнить его отладку.

Поставленная цель достигнута.