

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Сборка и отладка проекта в режиме программной эмуляции (Emulation-SW)</b>	<b>4</b>
<b>2 Сборка и отладка проекта в режиме аппаратной эмуляции (Emulation-HW)</b>	<b>7</b>
<b>3 Сборка и отладка проекта в режиме аппаратного исполнения (Hardware)</b>	<b>10</b>
<b>Заключение</b>	<b>15</b>
<b>Ответы на контрольные вопросы</b>	<b>16</b>

# Введение

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.

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 (Рисунок 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. Понимание этих преимуществ является необходимым условием для разработки вычислительных устройств и достижения наилучшего уровня ускорения.

**Цель работы:** Изучение методики и технологии синтеза аппаратных устройств ускорения вычислений по описаниям на языках высокого уровня. В ходе лабораторной работы рассматривается маршрут проектирования устройств, представленных в виде синтаксических конструкций ЯВУ С/C++, изучаются принципы работы IDE Xilinx Vitis HLS и методика анализа и отладки устройств. Для достижения данной цели необходимо выполнить следующие задачи:

- разработать ускоритель вычислений по индивидуальному заданию;
- разработать код для тестирования ускорителя;
- реализовать ускоритель с помощью средств высоко-уровненного синтеза;
- выполнить отладку ускорителя.

Все задания выполняются в соответствии с **вариантом №4**.

# 1 | Сборка и отладка проекта в режиме программной эмуляции (Emulation-SW)

Функции ядра на основе индивидуального задания с использованием не оптимизированного цикла, конвейерного цикла, частично развернутого цикла и конвейерного, частично развернутого цикла приведены в листингах 1.1, 1.2, 1.3 и 1.4 соответственно.

Листинг 1.1: Не оптимизированный цикл

```
1 extern "C" {
2 void var004_no_pragmas(int* c, const int* a, const int* b, const int len) {
3     for (int x = 0; x < len; ++x) {
4         int sumA = 0;
5         int sumB = 0;
6         int volA = a[x];
7         int volB = b[x];
8         for (int i = 0; i < 32; i++) {
9             sumA += (volA) & 1;
10            volA = volA /2;
11            sumB += (volB) & 1;
12            volB = volB /2;
13        }
14        if (sumA > sumB)
15            c[x] = sumA;
16        else
17            c[x] = sumB;
18    }
19 }
20 }
21 }
```

Листинг 1.2: Конвейерная организация цикла

```
1 extern "C" {
2 void var004_pipelined(int* c, const int* a, const int* b, const int len) {
3     for (int x = 0; x < len; ++x) {
4         #pragma HLS PIPELINE
5             int sumA = 0;
6             int sumB = 0;
7             int volA = a[x];
8             int volB = b[x];
9             for (int i = 0; i < 32; i++) {
10                 #pragma HLS PIPELINE
11                     sumA += (volA) & 1;
12                     volA = volA/2;
13                     sumB += (volB) & 1;
14                     volB = volB/2;
15                 }
16                 if (sumA > sumB)
17                     c[x] = sumA;
18                 else
19                     c[x] = sumB;
20
21             }
22         }
23 }
```

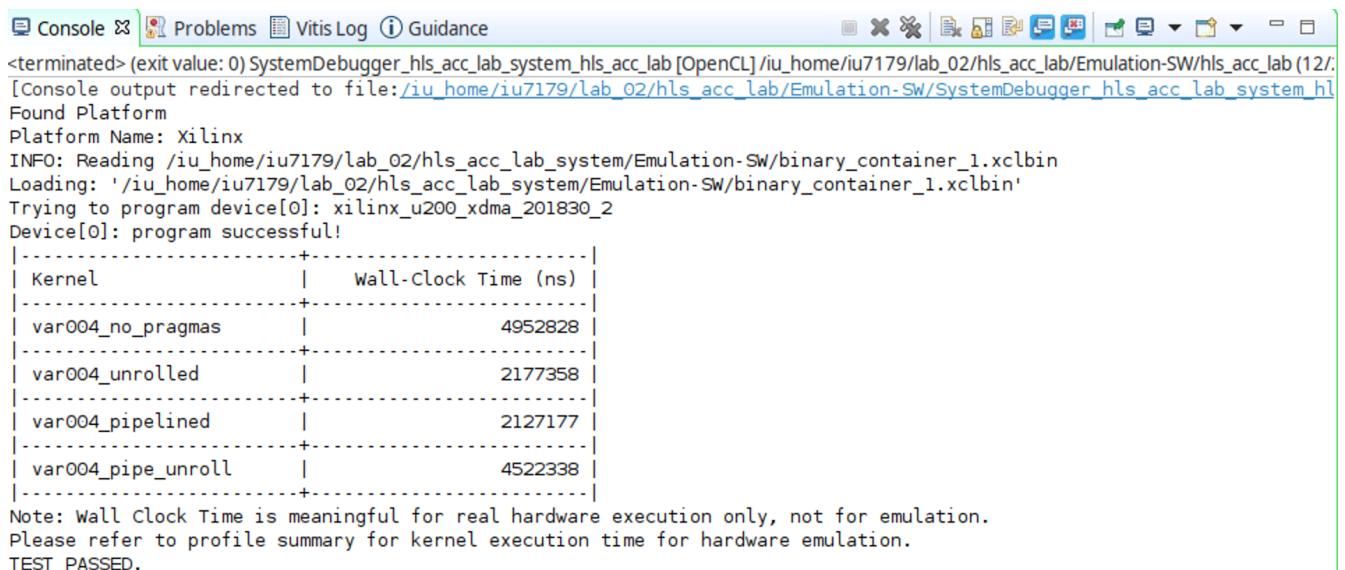
Листинг 1.3: Частично развернутый цикл

```
1 extern "C" {
2 void var004_unrolled(int* c, const int* a, const int* b, const int len) {
3     for (int x = 0; x < len; ++x) {
4         #pragma HLS UNROLL factor=2
5             int sumA = 0;
6             int sumB = 0;
7             int volA = a[x];
8             int volB = b[x];
9             for (int i = 0; i < 32; i++) {
10                 sumA += (volA) & 1;
11                 volA = volA/2;
12                 sumB += (volB) & 1;
13                 volB = volB/2;
14             }
15             if (sumA > sumB)
16                 c[x] = sumA;
17             else
18                 c[x] = sumB;
19
20         }
21     }
22 }
23 }
```

Листинг 1.4: Конвейерный и частично развернутый цикл

```
1 extern "C" {
2 void var004_pipe_unroll(int* c, const int* a, const int* b, const int len) {
3     for (int x = 0; x < len; ++x) {
4         #pragma HLS PIPELINE
5         #pragma HLS UNROLL factor=2
6         int sumA = 0;
7         int sumB = 0;
8         int volA = a[x];
9         int volB = b[x];
10        for (int i = 0; i < 32; i++) {
11            sumA += (volA) & 1;
12            volA = volA/2;
13            sumB += (volB) & 1;
14            volB = volB/2;
15        }
16        if (sumA > sumB)
17            c[x] = sumA;
18        else
19            c[x] = sumB;
20    }
21 }
22 }
23 }
```

Результаты запуска приложения, которые были выданы на вкладку Console в режиме Emulation-SW приведены на рисунке 1.1.



The screenshot shows the Vitis IDE interface with the 'Console' tab selected. The output window displays the following text:

```
<terminated> (exit value: 0) SystemDebugger_hls_acc_lab_system_hls_acc_lab [OpenCL] /iu_home/iu7179/lab_02/hls_acc_lab/Emulation-SW/hls_acc_lab (12).
[Console output redirected to file: /iu_home/iu7179/lab_02/hls_acc_lab/Emulation-SW/SystemDebugger_hls_acc_lab_system_hls_acc_lab]
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7179/lab_02/hls_acc_lab_system/Emulation-SW/binary_container_1.xclbin
Loading: '/iu_home/iu7179/lab_02/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) |
+-----+
| var004_no_pragmas | 4952828 |
+-----+
| var004_unrolled | 2177358 |
+-----+
| var004_pipelined | 2127177 |
+-----+
| var004_pipe_unroll | 4522338 |
+-----+
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.
```

Рис. 1.1: Результаты запуска приложения в режиме Emulation-SW

## 2 | Сборка и отладка проекта в режиме аппаратной эмуляции (Emulation-HW)

Копия экрана Assistant View для сборки Emulation-HW приведена на рисунках 2.1 и 2.2.

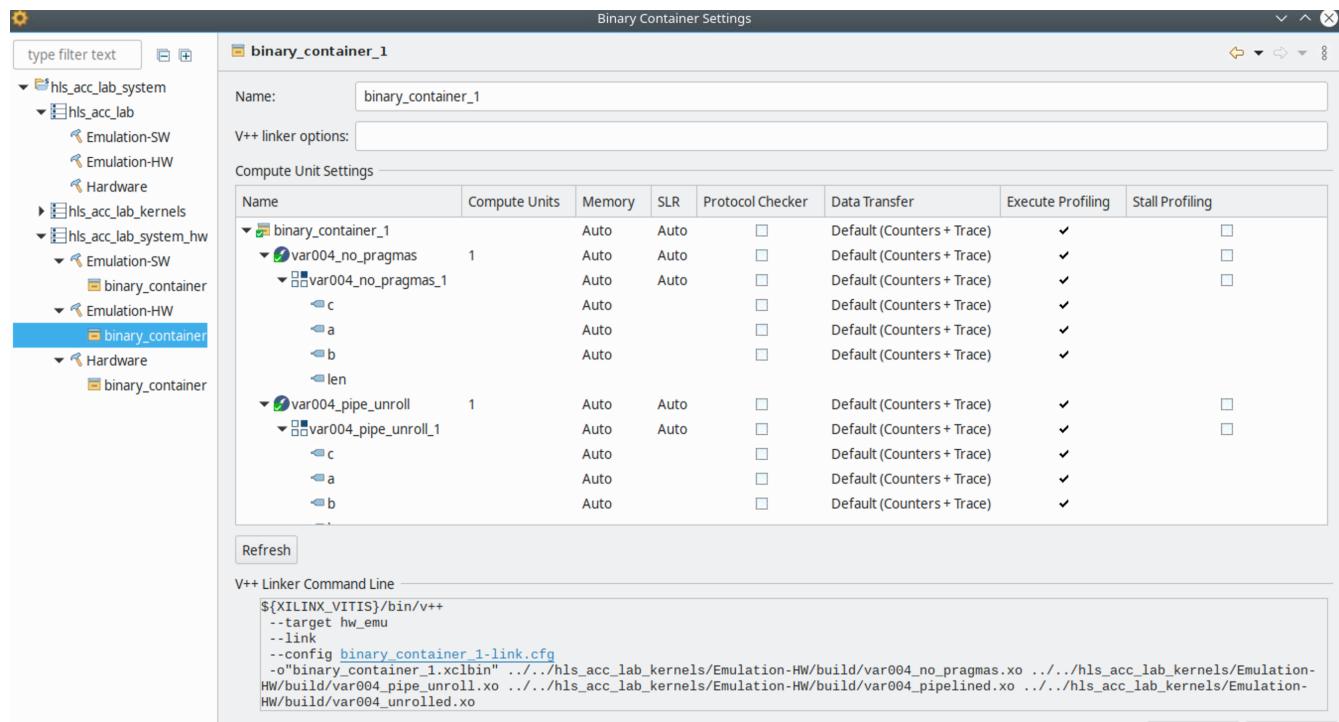


Рис. 2.1: Экран №1 Assistant View для сборки Emulation-HW

Результаты запуска приложения, которые были выданы на вкладку Console в режиме Emulation-HW приведены на рисунке 2.3.

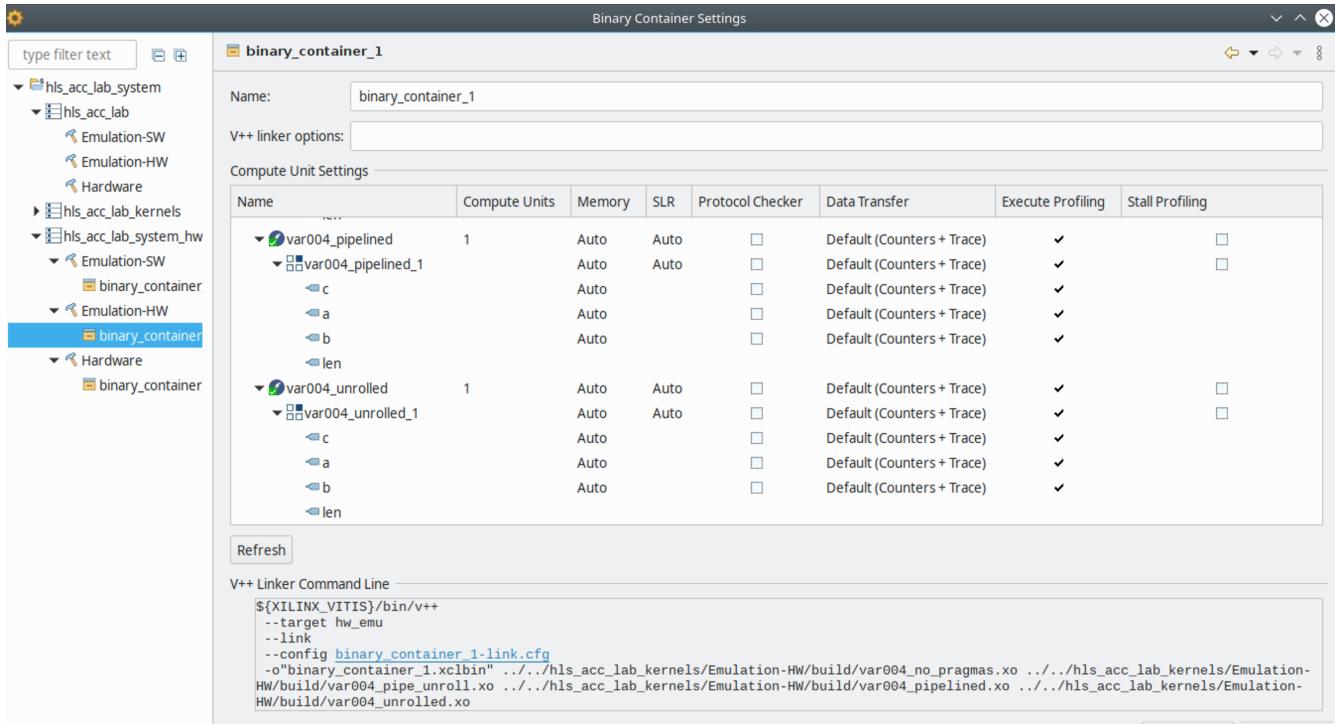


Рис. 2.2: Экран №2 Assistant View для сборки Emulation-HW

```

Console Problems Vitis Log Guidance
hw_no_sim_hls_acc_lab [OpenCL] hls_acc_lab/Emulation-HW/hls_acc_lab (12/20/21, 4:42 AM)
INFO: Reading /iu_home/iu7179/lab_02/hls_acc_lab_system/Emulation-HW/binary_container_1.xclbin
Loading: '/iu_home/iu7179/lab_02/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 simulation times. It is recommended to use a smaller data set for faster simulation.
INFO::[ Vitis-EM 22 ] [Time elapsed: 4 minute(s) 8 seconds, Emulation time: 0.085825 ms]
Data transfer between kernel(s) and global memory(s)
var004_no_pragmas_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var004_pipe_unroll_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var004_pipelined_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var004_unrolled_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB

Device[0]: program successful!
-----+-----+
| Kernel | Wall-Clock Time (ns) |
-----+-----+
| var004_no_pragmas | 53021036256 |
-----+-----+
INFO::[ Vitis-EM 22 ] [Time elapsed: 9 minute(s) 9 seconds, Emulation time: 0.187282 ms]
Data transfer between kernel(s) and global memory(s)
var004_no_pragmas_1:m_axi_gmem-DDR[1] RD = 8.000 KB WR = 4.000 KB
var004_pipe_unroll_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var004_pipelined_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 0.000 KB
var004_unrolled_1:m_axi_gmem-DDR[1] RD = 8.000 KB WR = 4.000 KB

| var004_unrolled | 54019315294 |
-----+-----+
| var004_pipelined | 49011220874 |
-----+-----+
| var004_pipe_unroll | 51025609755 |
-----+-----+
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.

```

Рис. 2.3: Результаты запуска приложения в режиме Emulation-HW

На рисунке 2.4 приведена диаграмма работы четырех ядер.

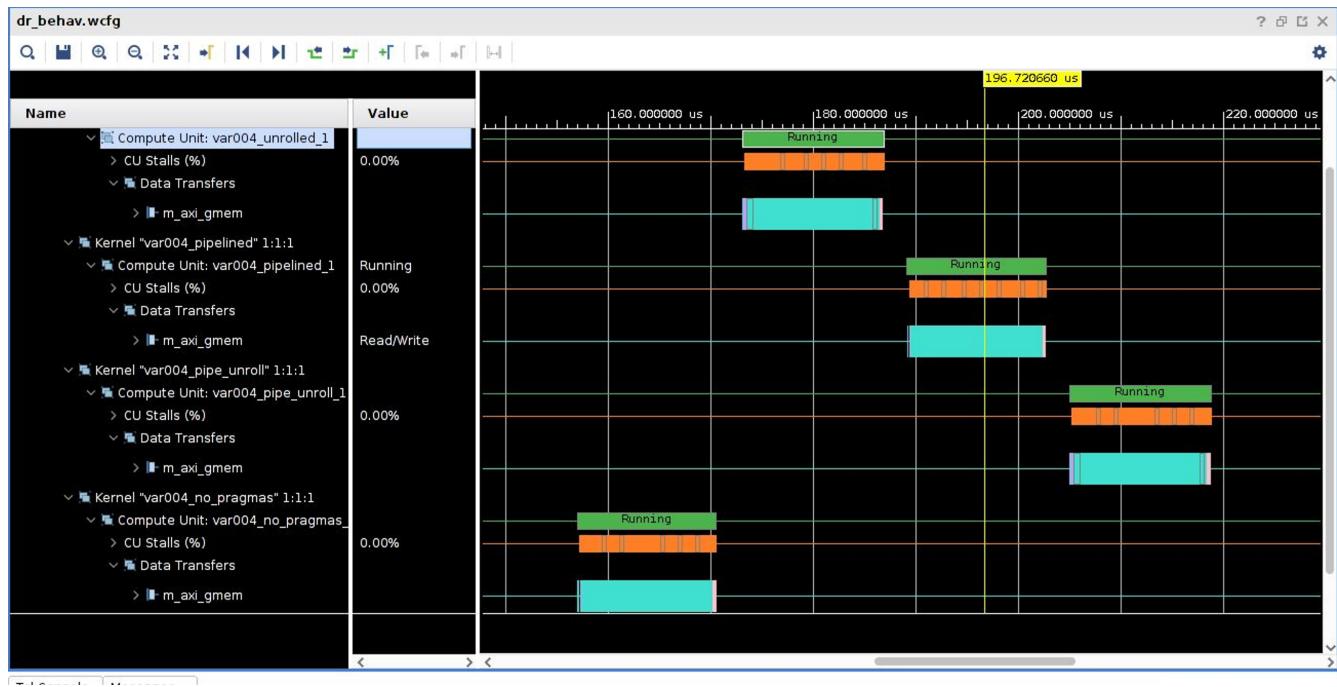
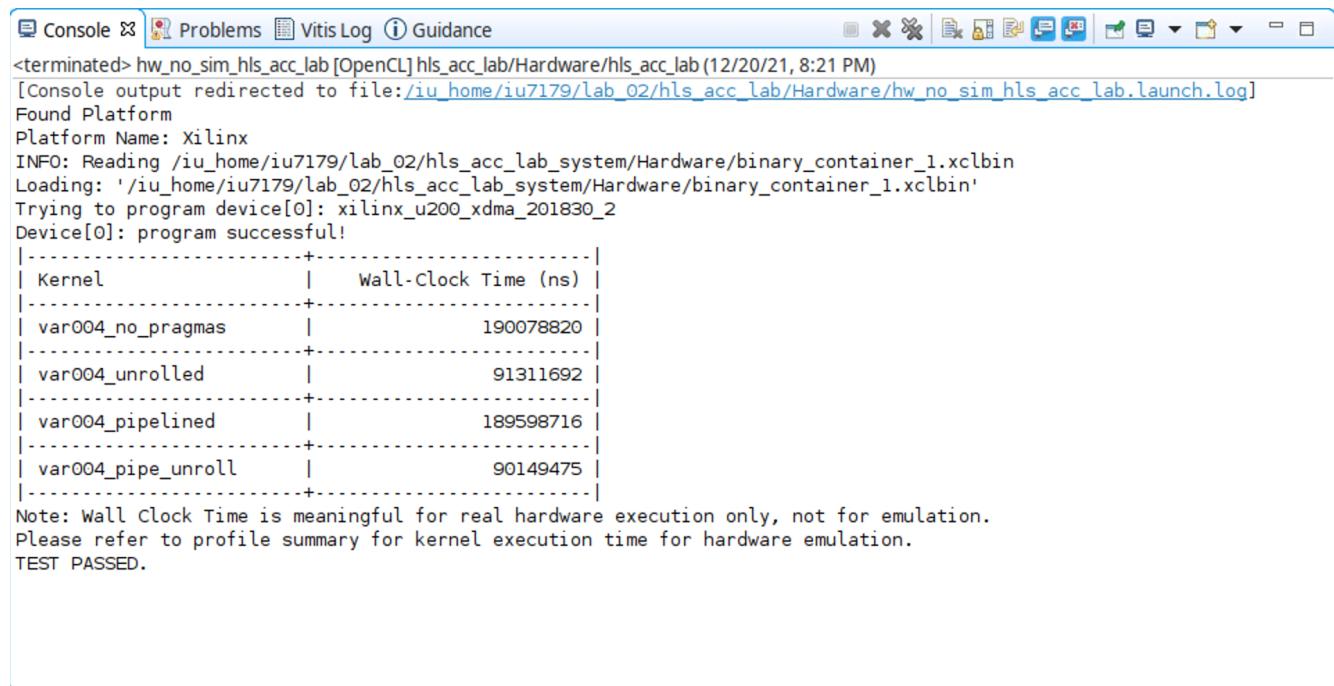


Рис. 2.4: Диаграмма работы четырех ядер

### 3 | Сборка и отладка проекта в режиме аппаратного исполнения (Hardware)

Результаты запуска приложения, которые были выданы на вкладку Console в режиме Hardware приведены на рисунке 3.1.



The screenshot shows the Vitis IDE interface with the 'Console' tab selected. The output window displays the following text:

```
<terminated> hw_no_sim_hls_acc_lab[OpenCL] hls_acc_lab/Hardware/hls_acc_lab (12/20/21, 8:21 PM)
[Console output redirected to file:/iu_home/iu7179/lab_02/hls_acc_lab/Hardware/hw_no_sim_hls_acc_lab.launch.log]
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7179/lab_02/hls_acc_lab_system/Hardware/binary_container_1.xclbin
Loading: '/iu_home/iu7179/lab_02/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) |
+-----+
| var004_no_pragmas | 190078820 |
| var004_unrolled | 91311692 |
| var004_pipelined | 189598716 |
| var004_pipe_unroll | 90149475 |
+-----+
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.
```

Рис. 3.1: Результаты запуска приложения в режиме Hardware

Копии экранов для вкладок «Summary», «System Diagram», «Platform Diagram» и четыре вкладки «HLS Synthesis» для каждого ядра приведены на рисунках 3.2 - 3.8.

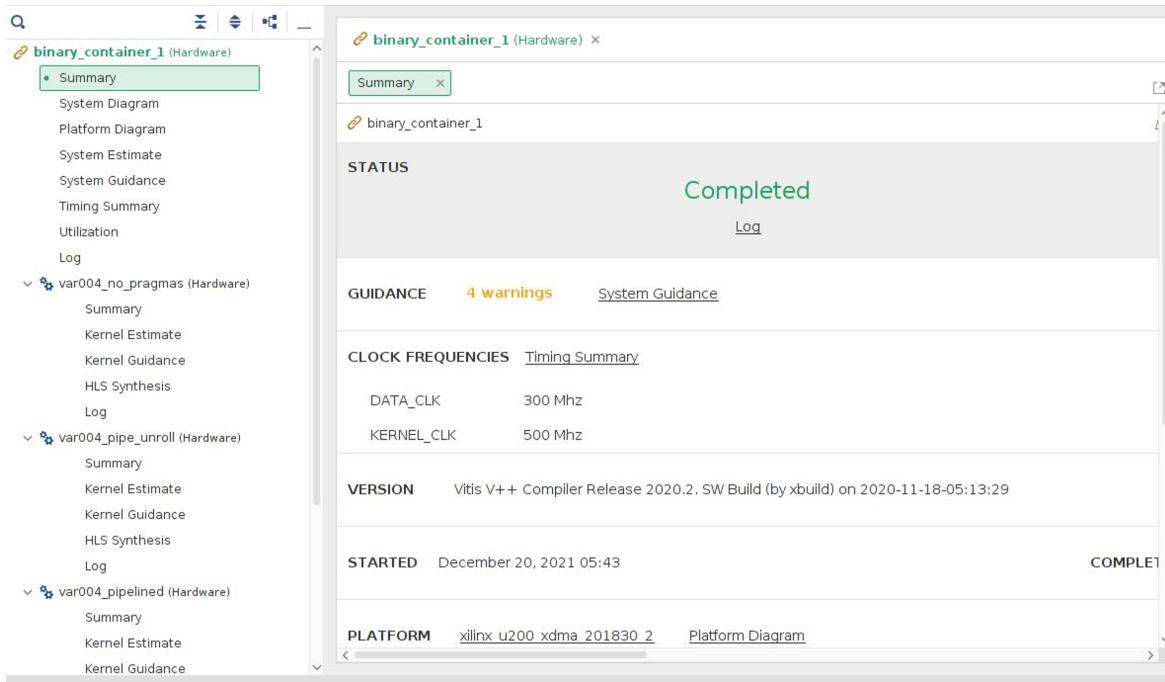


Рис. 3.2: Копия экрана для вкладки «Summary»

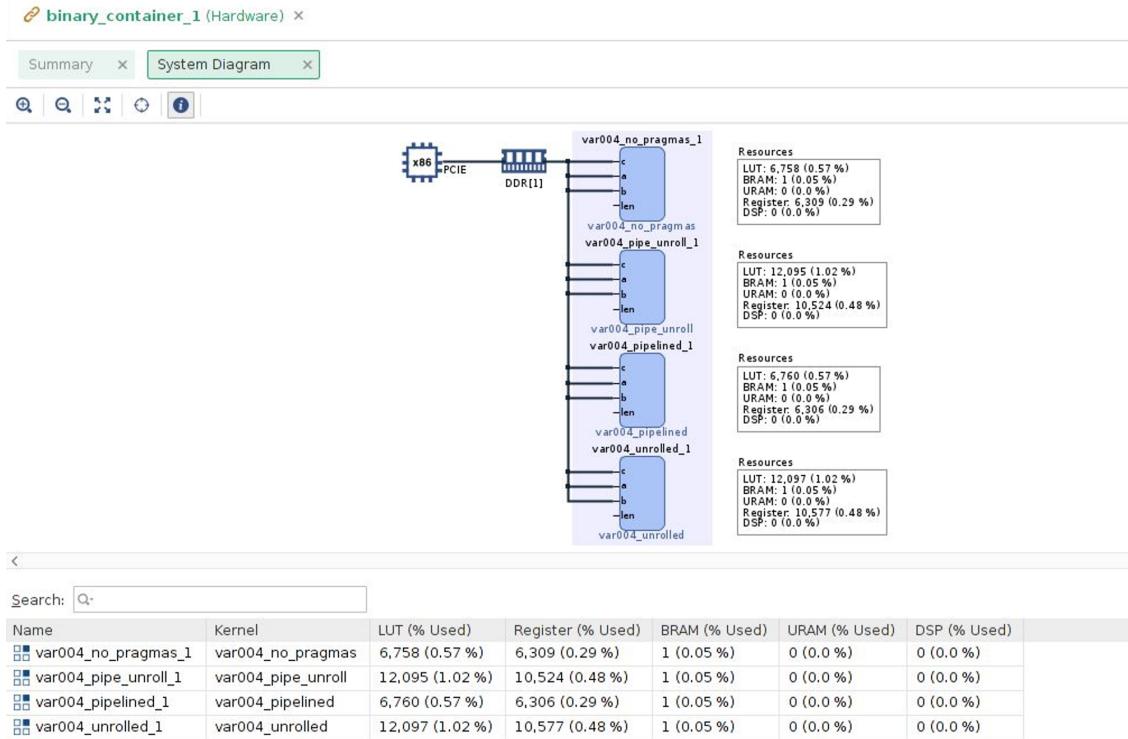


Рис. 3.3: Копия экрана для вкладки «System Diagram»

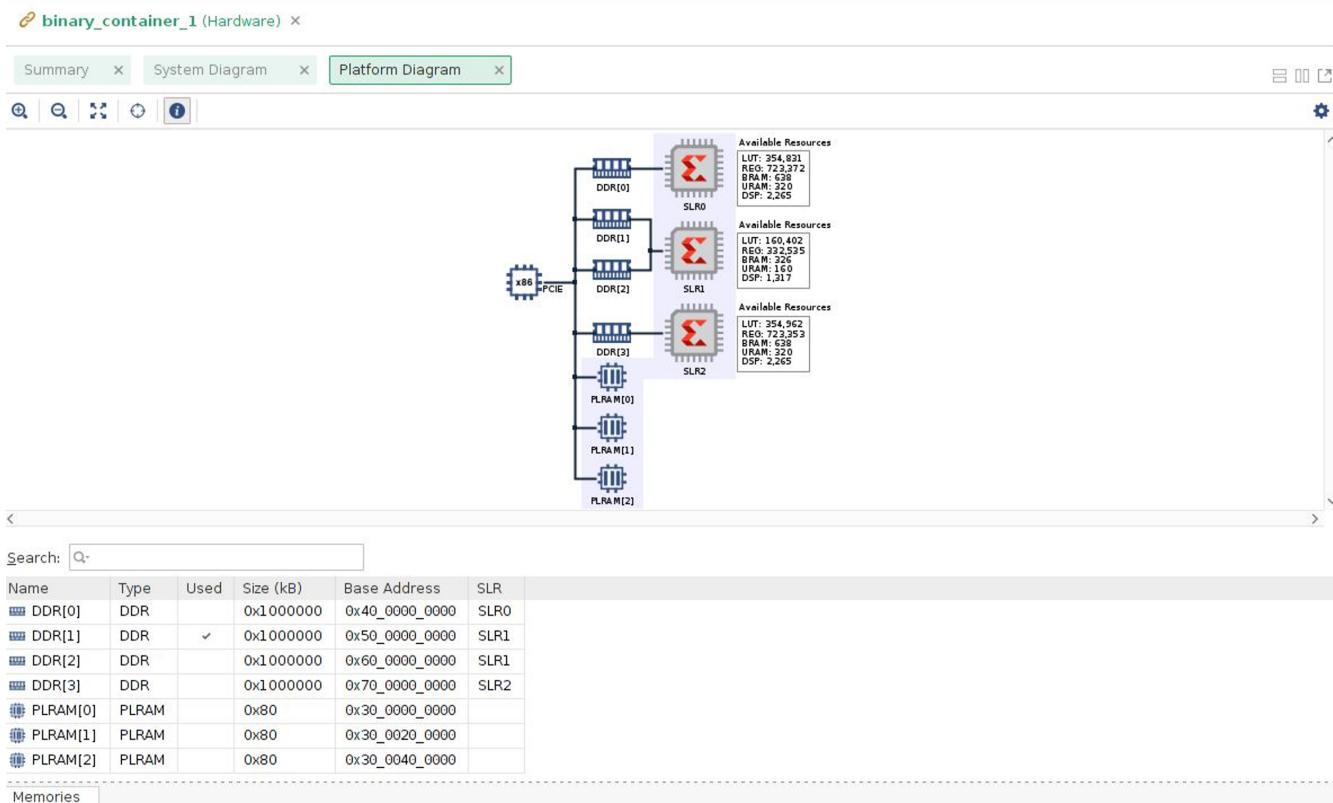


Рис. 3.4: Копия экрана для вкладки «Platform Diagram»

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var004_no_pragmas	II Violation						no	2	~0	0	0	6826	~0	9486	~0	0.00
VITIS_LOOP_3_1	II Violation			174	2		yes									

Рис. 3.5: Копия экрана для вкладки «HLS Synthesis» для ядра без оптимизации цикла

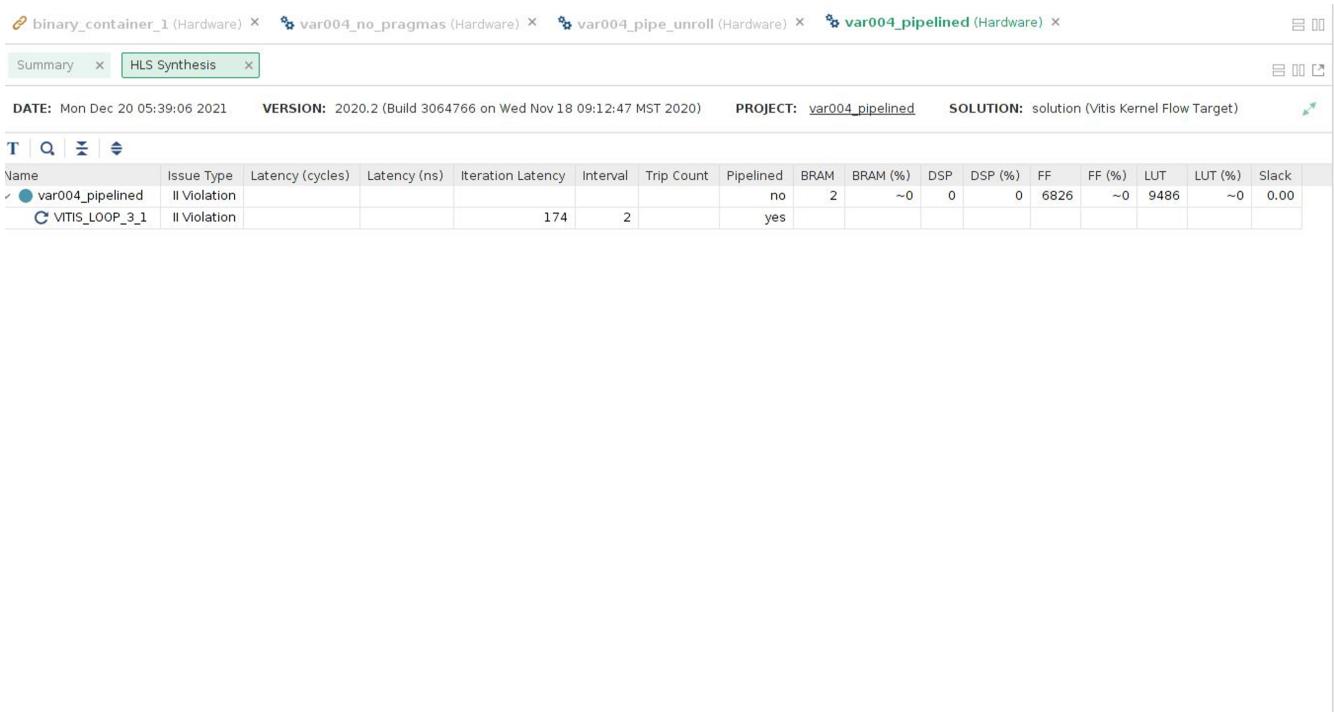


Рис. 3.6: Копия экрана для вкладки «HLS Synthesis» для ядра с конвейерной организацией цикла

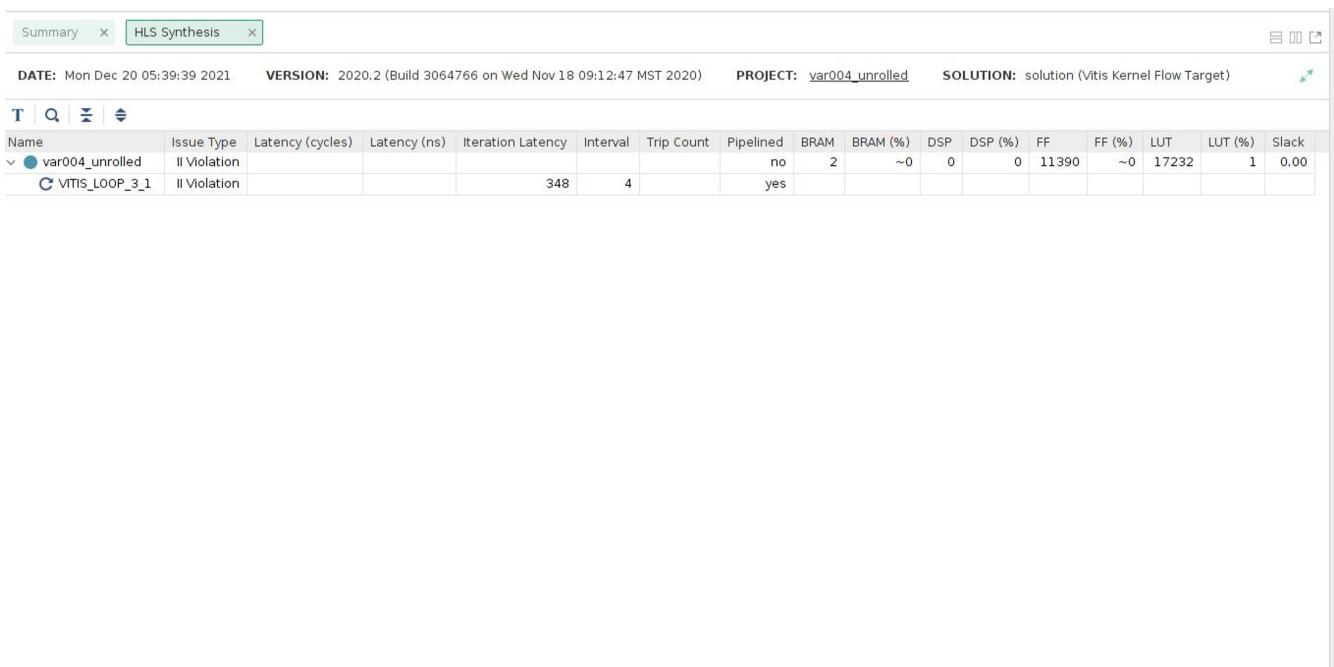


Рис. 3.7: Копия экрана для вкладки «HLS Synthesis» для ядра с частично развернутым циклом

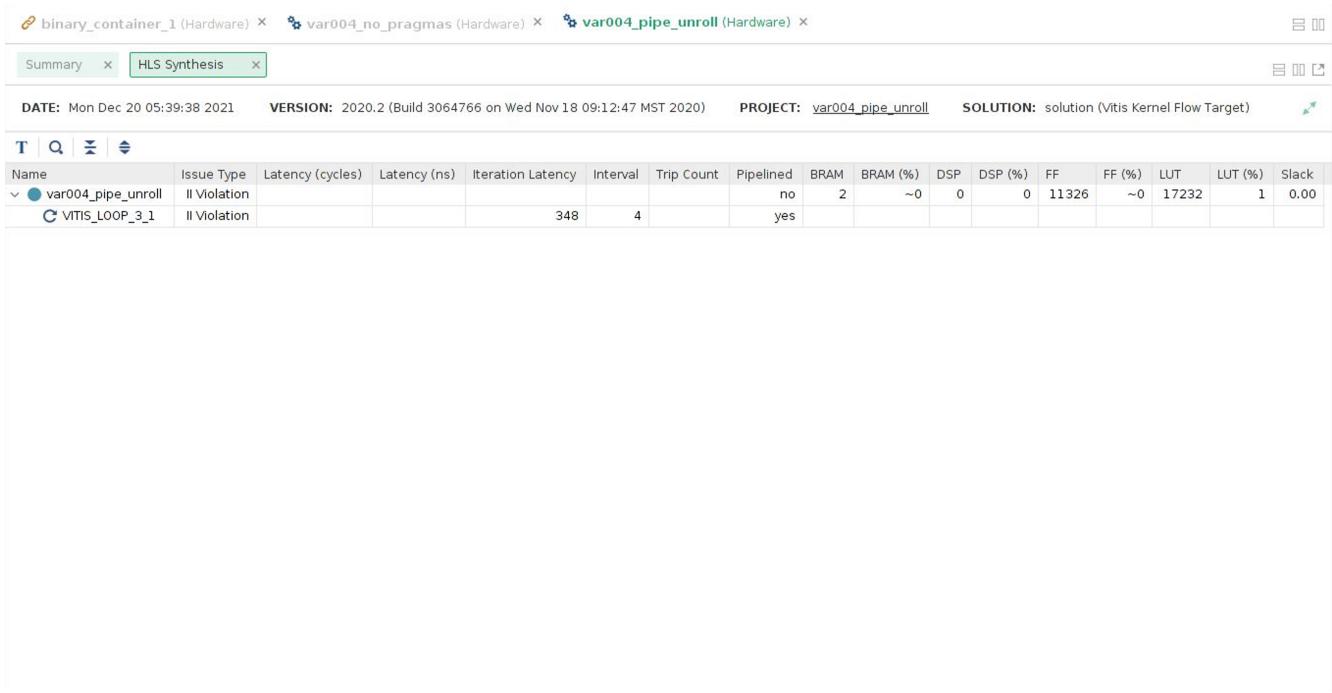


Рис. 3.8: Копия экрана для вкладки «HLS Synthesis» для ядра с конвейерным и частично развернутым циклом

В результате сборки проекта было выяснено, что самым быстрым является ядро с конвейерным, частично развернутым циклом, так как данный подход объединяет оба метода оптимизации: разворачивание циклов и конвейеризацию, а самым медленным является ядро без оптимизаций. При этом стоит отметить, что в исходном коде разворачивание циклов было настроено с параметром разворачивания 2, что также можно увидеть в полученных результатах: время выполнения развернутого цикла примерно в 2 раза быстрее, чем обычного. Конвейеризация же за счёт вложенного цикла и зависимости итераций по данным не привела к заметному повышению производительности.

# Заключение

В ходе выполнения лабораторной работы были изучены методики и технологии синтеза аппаратных устройств ускорения вычислений по описаниям на языках высокого уровня. Также был рассмотрен маршрут проектирования устройств, представленных в виде синтаксических конструкций ЯВУ C/C++, изучены принципы работы IDE Xilinx Vitis HLS и методика анализа и отладки устройств, был разработан ускоритель вычислений по индивидуальному заданию, разработан код для тестирования ускорителя, реализован ускоритель с помощью средств высоко-уровненного синтеза, выполнена его отладка.

# Ответы на контрольные вопросы

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

CPU более универсален и подходит для самых разнообразных задач, но ввиду своей архитектуры CPU не столь эффективны для параллельных вычислений. Графические процессоры более приспособлены для целей параллельных вычислений, но работают в основном в нише отображения информации на экране. Аппаратные ускорители на ПЛИС в отличие от универсального и графического процессоров можно перепрограммировать в соответствии с особенностями решаемой на них вычислительной задачи. Получается синтез специализированного процессора под конкретную задачу.

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

Конвейерная обработка циклов, разворачивание циклов, потоковая обработка.

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

Этап 1: Инициализируется среда OpenCL. На этом этапе хост должен инициировать доступ к подключенному устройство Xilinx, и загрузить в него двоичный файл .xclbin. Затем создается очередь команд и объект ядра.

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). Это делается с помощью конструктора cl::Buffer, после чего буфер должен быть сопоставлен с локальной памятью ядра. Такое сопоставление производится при вызове методов setArg().

Этап 3. Запуск задачи на исполнение. Хост-программа устанавливает аргументы ядра через вызов setArg(), затем посыпает команду на выполнение и читает результаты обратно в память хоста (ОЗУ хост-системы). Эти операции помещаются в очередь команд, объявленную на этапе 1. Указанные вызовы функций не являются блокирующими, а соблюдение порядка выполнения гарантируется упорядоченной очередью q. Также существует возможность организовать очередь с переупорядочиванием команд, что позволит реализовать выполнение команд по готовности ядра, а не по их порядку в очереди. Вызов q.finish() позволяет ожидать завершения всех поставленных в очередь команд.

Этап 4: После завершения работы всех команд выходной буфер Rbuf содержит результаты работы ядра. Его чтение было выполнено в конце работы вызовом метода q.enqueueMigrateMemObjects(). После этого результаты работы могут читаться как обычный массив.

## **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?**

Для оптимизаций ускорителей могут быть использованы директивы такие, как pragma HLS UNROLL для разворачивания циклов, pragma HLS PIPELINE для конвейеризации.