



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 №5 по курсу «Архитектура ЭВМ»

«Разработка ускорителей вычислений средствами САПР  
высокоуровневого синтеза Xilinx Vitis HLS»

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

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

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Основные теоретические сведения</b>	<b>4</b>
1.1 Методология ускорения вычислений на основе ПЛИС . . . . .	4
1.2 Оптимизация времени обработки и пропускной способности .	5
1.2.1 Конвейерная обработка циклов . . . . .	5
1.2.2 Разворачивание циклов . . . . .	6
1.2.3 Поточковая обработка . . . . .	6
<b>2 Практическая часть</b>	<b>7</b>
2.1 Функции ядра по индивидуальному варианту . . . . .	7
2.2 Вариант сборки Emulation-SW . . . . .	8
2.3 Вариант сборки Emulation-HW . . . . .	9
2.4 Вариант сборки Hardware . . . . .	13
2.5 Вывод . . . . .	16
<b>3 Контрольные вопросы</b>	<b>17</b>
<b>Заключение</b>	<b>19</b>

# Введение

**Целью данной работы** является 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-уровневого синтеза;
- выполнить его отладку.

# 1 Основные теоретические сведения

В данном разделе описаны методология ускорения вычислений на основе ПЛИС и способы оптимизации времени обработки и пропускной способности на ускорителе вычислений **Xilinx Alveo**.

## 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.

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 1.1). 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.

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

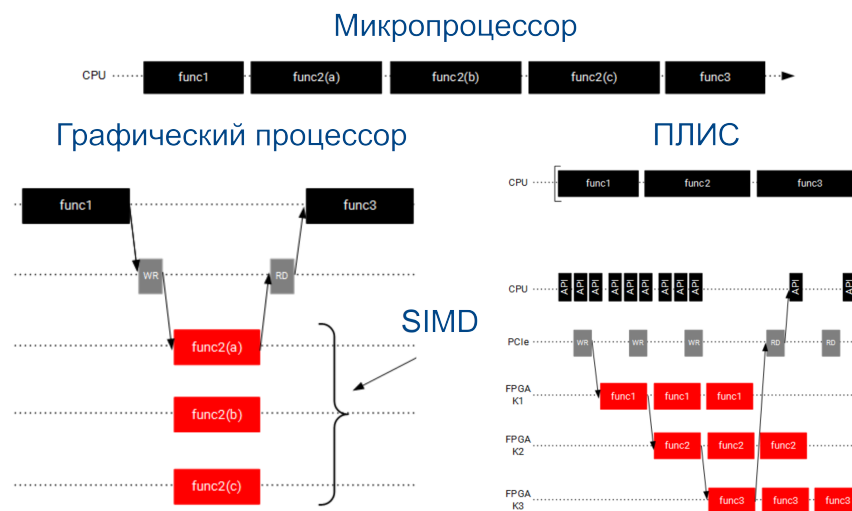


Рисунок 1.1 – Принципы организации вычислений на различных платформах

- 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) в таком варианте может быть организован конвейер выполнения. Для этого необходимо поместить в тело цикла прагму `<HLS PIPELINE>`.

Без конвейерной обработки каждая последующая итерация цикла начинается в каждом третьем такте. При конвейерной обработке цикл может начинать последующие итерации цикла менее чем за три такта, например, в каждом втором такте или в каждом такте. Для указания конкретного значения интервала запуска заданий используется свойство `Initiation interval (II)`. Например: `#pragma HLS PIPELINE II = 1`

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, и результатом может быть большой интервал инициации.

## **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 Поточковая 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 Практическая часть

В данном разделе представлен ход выполнения лабораторной работы.

### 2.1 Функции ядра по индивидуальному варианту

На листинге 2.1 представлен код функции ядра без оптимизаций, на листинге 2.2 – код функции ядра с конвейерной оптимизацией, на листинге 2.3 – код функции ядра с развернутым циклом, на листинге 2.4 – код функции ядра с конвейерной оптимизацией и развернутым циклом.

Листинг 2.1 – Программа варианта без оптимизаций

```
1 extern "C"
2 {
3     void var012_no_pragmas(int* c, const int* a, const int* b, const int len)
4     {
5         int ptr = 0;
6         for (int i = 0; i < len; i++)
7         {
8             ptr = b[i] % len;
9             c[i] = a[ptr] + i;
10        }
11    }
12 }
```

Листинг 2.2 – Программа варианта с конвейерной организацией

```
1 extern "C"
2 {
3     void var012_pipelined(int* c, const int* a, const int* b, const int len)
4     {
5         int ptr = 0;
6         for (int i = 0; i < len; i++)
7         {
8             #pragma HLS PIPELINE
9             ptr = b[i] % len;
10            c[i] = a[ptr] + i;
11        }
12    }
13 }
```

### Листинг 2.3 – Программа варианта с разворачиванием цикла

```
1 extern "C"
2 {
3     void var012_unrolled(int* c, const int* a, const int* b, const int len)
4     {
5         int ptr = 0;
6         for (int i = 0; i < len; i++)
7         {
8             #pragma HLS UNROLL
9             ptr = b[i] % len;
10            c[i] = a[ptr] + i;
11        }
12    }
13 }
```

### Листинг 2.4 – Программа варианта с конвейерной организацией и разворачиванием

```
1 extern "C"
2 {
3     void var012_pipe_unroll(int* c, const int* a, const int* b, const int len)
4     {
5         int ptr = 0;
6         for (int i = 0; i < len; i++)
7         {
8             #pragma HLS DATAFLOW
9             ptr = b[i] % len;
10            c[i] = a[ptr] + i;
11        }
12    }
13 }
```

## 2.2 Вариант сборки Emulation-SW

На листинге 2.5 представлены результат работы приложения в режиме Emulation-SW.

### Листинг 2.5 – Результаты Emulation-SW

```
1 [Console output redirected to file:/iu_home/iu7072/workspace/Alveo_lab2/
   Emulation-SW/SystemDebugger_Alveo_lab2_system_Alveo_lab2.launch.log]
2 Found Platform
3 Platform Name: Xilinx
4 INFO: Reading /iu_home/iu7072/workspace/Alveo_lab2_system/Emulation-SW/
```



## Листинг 2.5 (продолжение)

```

    binary_container_1.xclbin
5 Loading:  '/iu_home/iu7072/workspace/Alveo_lab2_system/Emulation-SW/
    binary_container_1.xclbin '
6 Trying to program device[0]: xilinx_u200_xdma_201830_2
7 Device[0]: program successful!
8 |-----+-----|
9 | Kernel                | Wall-Clock Time (ns) |
10 |-----+-----|
11 | var012_no_pragmas      |                2070100 |
12 |-----+-----|
13 | var012_unrolled        |                1262177 |
14 |-----+-----|
15 | var012_pipelined       |                438559  |
16 |-----+-----|
17 | var012_pipe_unroll     |                539648  |
18 |-----+-----|
19 Note: Wall Clock Time is meaningful for real hardware execution only, not for
    emulation.
20 Please refer to profile summary for kernel execution time for hardware emulation
    .
21 TEST PASSED.

```

## 2.3 Вариант сборки Emulation-HW

На рисунках 2.1-2.2 представлена копия экрана **Assistant View** для варианта сборки **Emulation-HW**. На листинге 2.6 представлены результаты работы приложения в данном варианте сборки. На рисунке 2.3 представлена копия окна внутрисхемного отладчика Vivado.

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"/>
▼ var012_no_pragmas	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
▼ var012_no_pragmas_1		Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
c		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
a		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
b		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
len							
▼ var012_pipelined	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
▼ var012_pipelined_1		Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
c		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
a		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
b		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
len							

Рисунок 2.1 – Копия экрана Assistant View для Emulation-HW (часть 1)

var012_unrolled	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
var012_unrolled_1		Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
c		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
a		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
b		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
len							
var012_pipe_unroll	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
var012_pipe_unroll_1		Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
c		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
a		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
b		Auto		<input type="checkbox"/>	Default (Counters + Trace)	✓	
len							

Рисунок 2.2 – Копия экрана Assistant View для Emulation-HW (часть 2)

## Листинг 2.6 – Результаты Emulation-HW

```

1 [Console output redirected to file:/iu_home/iu7072/workspace/Alveo_lab2/
  Emulation-HW/SystemDebugger_Alveo_lab2_system_Alveo_lab2.launch.log]
2 Found Platform
3 Platform Name: Xilinx
4 INFO: Reading /iu_home/iu7072/workspace/Alveo_lab2_system/Emulation-HW/
  binary_container_1.xclbin
5 Loading: '/iu_home/iu7072/workspace/Alveo_lab2_system/Emulation-HW/
  binary_container_1.xclbin'
6 Trying to program device[0]: xilinx_u200_xdma_201830_2
7 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.
8 Device[0]: program successful!
9 |-----+-----|
10 | Kernel          | Wall-Clock Time (ns) |
11 |-----+-----|
12 | var012_no_pragmas      | 15008592508 |
13 |-----+-----|
14 INFO:.[ Vitis-EM 22 ] [Time elapsed: 3 minute(s) 38 seconds, Emulation time:
  0.200175 ms]
15 Data transfer between kernel(s) and global memory(s)
16 var012_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
  4.000 KB
17 var012_pipe_unroll_1:m_axi_gmem-DDR[1]          RD = 0.000 KB          WR =
  0.000 KB
18 var012_pipelined_1:m_axi_gmem-DDR[1]          RD = 0.000 KB          WR =
  0.000 KB
19 var012_unrolled_1:m_axi_gmem-DDR[1]          RD = 0.301 KB          WR =
  0.125 KB
20
21 INFO:.[ Vitis-EM 22 ] [Time elapsed: 8 minute(s) 39 seconds, Emulation time:
  0.466535 ms]
22 Data transfer between kernel(s) and global memory(s)

```

## Листинг 2.6 (продолжение)

```

23 var012_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
    4.000 KB
24 var012_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 0.000 KB          WR =
    0.000 KB
25 var012_pipelined_1:m_axi_gmem-DDR[1]           RD = 0.000 KB          WR =
    0.000 KB
26 var012_unrolled_1:m_axi_gmem-DDR[1]            RD = 3.750 KB          WR =
    1.812 KB
27
28 INFO::[ Vitis-EM 22 ] [Time elapsed: 13 minute(s) 39 seconds, Emulation time:
    0.771576 ms]
29 Data transfer between kernel(s) and global memory(s)
30 var012_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
    4.000 KB
31 var012_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 0.000 KB          WR =
    0.000 KB
32 var012_pipelined_1:m_axi_gmem-DDR[1]           RD = 0.000 KB          WR =
    0.000 KB
33 var012_unrolled_1:m_axi_gmem-DDR[1]            RD = 7.699 KB          WR =
    3.812 KB
34
35 | var012_unrolled          |          653277479951 |
36 |-----+-----|
37 | var012_pipelined          |          24012090705 |
38 |-----+-----|
39 INFO::[ Vitis-EM 22 ] [Time elapsed: 18 minute(s) 40 seconds, Emulation time:
    1.05055 ms]
40 Data transfer between kernel(s) and global memory(s)
41 var012_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
    4.000 KB
42 var012_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 2.270 KB          WR =
    1.133 KB
43 var012_pipelined_1:m_axi_gmem-DDR[1]           RD = 8.000 KB          WR =
    4.000 KB
44 var012_unrolled_1:m_axi_gmem-DDR[1]            RD = 8.000 KB          WR =
    4.000 KB
45
46 INFO::[ Vitis-EM 22 ] [Time elapsed: 23 minute(s) 40 seconds, Emulation time:
    1.25502 ms]
47 Data transfer between kernel(s) and global memory(s)
48 var012_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
    4.000 KB
49 var012_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 4.195 KB          WR =
    2.094 KB
50 var012_pipelined_1:m_axi_gmem-DDR[1]           RD = 8.000 KB          WR =
    4.000 KB
51 var012_unrolled_1:m_axi_gmem-DDR[1]            RD = 8.000 KB          WR =

```

## Листинг 2.6 (продолжение)

```

52      4.000 KB
53 INFO::[ Vitis-EM 22 ] [Time elapsed: 28 minute(s) 46 seconds, Emulation time:
      1.51806 ms]
54 Data transfer between kernel(s) and global memory(s)
55 var012_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
56 var012_pipe_unroll_1:m_axi_gmem-DDR[1]          RD = 6.672 KB          WR =
      3.332 KB
57 var012_pipelined_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
58 var012_unrolled_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
59
60 | var012_pipe_unroll          |          1028355914876 |
61 |-----+-----|
62 Note: Wall Clock Time is meaningful for real hardware execution only, not for
      emulation.
63 Please refer to profile summary for kernel execution time for hardware emulation
      .
64 TEST PASSED.
65 INFO::[ Vitis-EM 22 ] [Time elapsed: 33 minute(s) 46 seconds, Emulation time:
      1.73059 ms]
66 Data transfer between kernel(s) and global memory(s)
67 var012_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
68 var012_pipe_unroll_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
69 var012_pipelined_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
70 var012_unrolled_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
71
72 INFO::[ Vitis-EM 22 ] [Time elapsed: 35 minute(s) 39 seconds, Emulation time:
      1.79479 ms]
73 Data transfer between kernel(s) and global memory(s)
74 var012_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
75 var012_pipe_unroll_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
76 var012_pipelined_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
77 var012_unrolled_1:m_axi_gmem-DDR[1]          RD = 8.000 KB          WR =
      4.000 KB
78
79 INFO: [HW-EMU 06-0] Waiting for the simulator process to exit

```

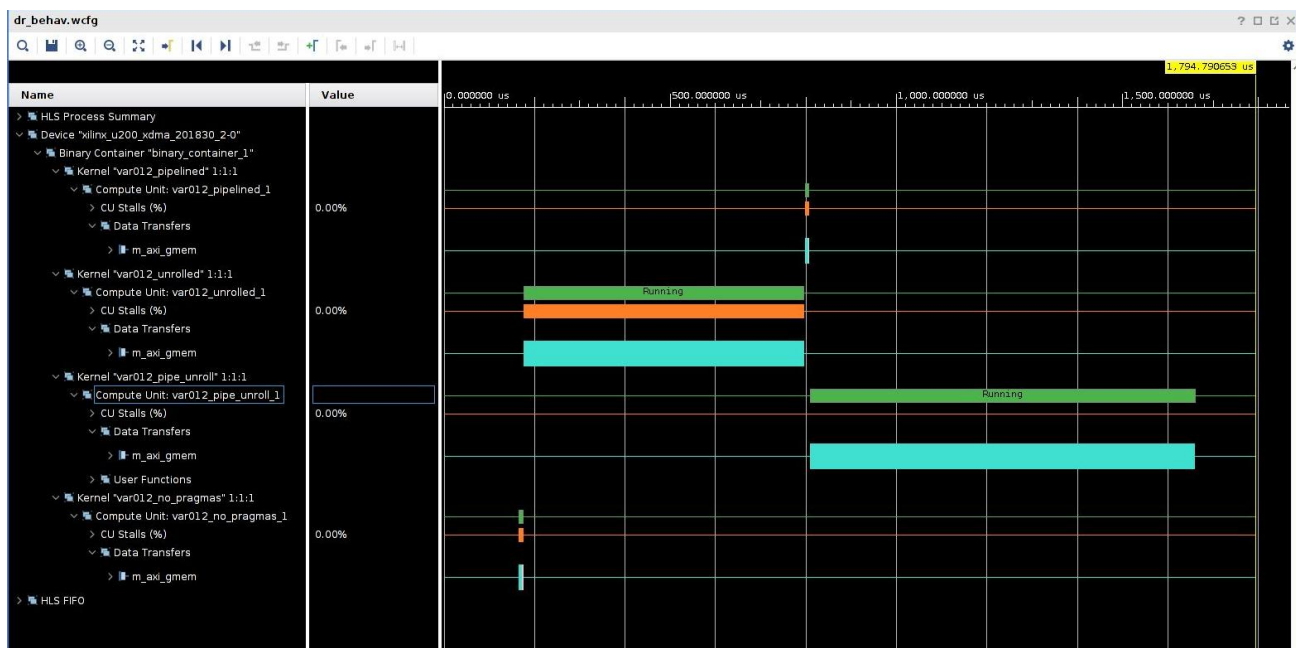


Рисунок 2.3 – Окно внутрисхемного отладчика Vivado для сборки в режиме Emulation-HW

## 2.4 Вариант сборки Hardware

На листинге 2.7 представлен результат работы приложения в режиме Hardware. На рисунках 2.4-2.10 представлены копии экрана для вкладок «Summary», «System Diagram», «Platform Diagram» и четыре вкладки «HLS Synthesis» для каждого ядра сборки Hardware в соответствующем порядке.

Листинг 2.7 – Результаты Hardware

```

1 [Console output redirected to file:/iu_home/iu7072/workspace/Alveo_lab2/Hardware
  /SystemDebugger_Alveo_lab2_system_Alveo_lab2.launch.log]
2 Found Platform
3 Platform Name: Xilinx
4 INFO: Reading /iu_home/iu7072/workspace/Alveo_lab2_system/Hardware/
  binary_container_1.xclbin
5 Loading: '/iu_home/iu7072/workspace/Alveo_lab2_system/Hardware/
  binary_container_1.xclbin'
6 Trying to program device[0]: xilinx_u200_xdma_201830_2
7 Device[0]: program successful!
8 |-----+-----|
9 | Kernel                | Wall-Clock Time (ns) |
10 |-----+-----|
11 | var012_no_pragmas      |          45233097 |
12 |-----+-----|
13 | var012_urolled        |          662424184 |
14 |-----+-----|

```

## Листинг 2.7 (продолжение)

```

15 | var012_pipelined          |          44303683 |
16 |-----+-----|
17 | var012_pipe_unroll       |          913769872 |
18 |-----+-----|
19 Note: Wall Clock Time is meaningful for real hardware execution only, not for
    emulation.
20 Please refer to profile summary for kernel execution time for hardware emulation
    .
21 TEST PASSED.

```

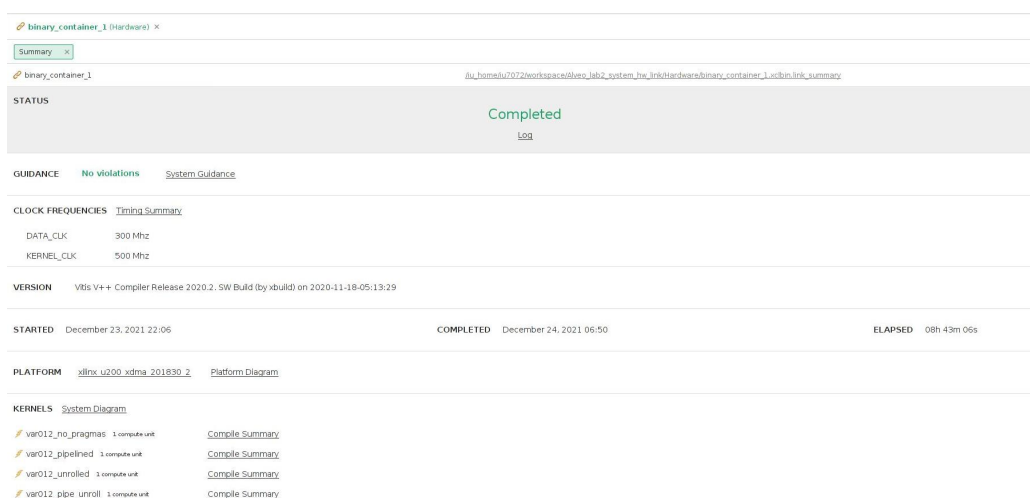


Рисунок 2.4 – Копия экрана вкладки Summary

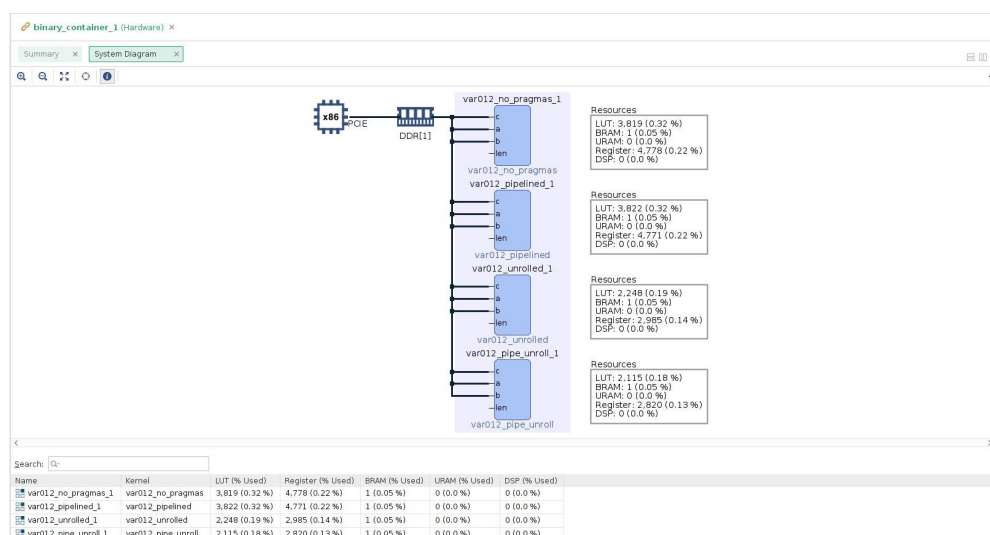


Рисунок 2.5 – Копия экрана вкладки System Diagram

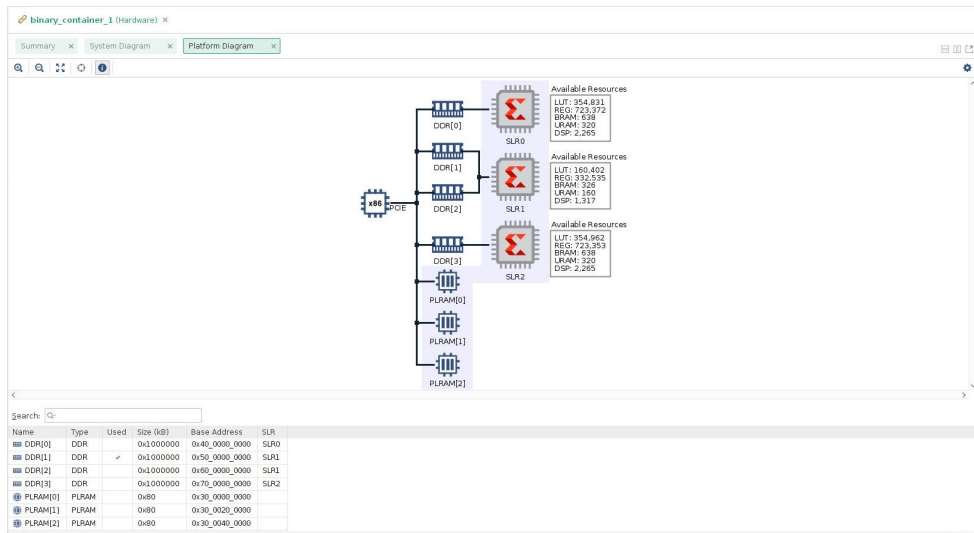


Рисунок 2.6 – Копия экрана вкладки Platform Diagram

binary\_container\_1 (Hardware) ×

var012\_no\_pragmas (Hardware) ×

Summary ×

HLS Synthesis ×

DATE: Thu Dec 23 21:56:27 2021

VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)

PROJECT: [var012\\_no\\_pragmas](#)

SOLUTION: solution (Vitis Kernel Flow Target)

PRODUCT FAMILY: virtexuplus

T

Q

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var012_no_pragmas	II Violation				0		no	2	~0	0	0	3756	~0	3588	~0	0.00
VITIS_LOOP_6_1	II Violation			181	2		yes									

Рисунок 2.7 – Копия экрана вкладки HLS Synthesis (без прагм)

binary\_container\_1 (Hardware)

var012\_no\_pragmas (Hardware)

var012\_pipelined (Hardware)

Summary

HLS Synthesis

DATE: Thu Dec 23 21:56:27 2021

VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)

PROJECT: var012\_pipelined

SOLUTION: solution (Vitis Kernel Flow Target)

PRODUCT FAMILY: virtexuplus

T

Q

≡

⌕

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var012_pipelined	II Violation					0	no	2	~0	0	0	3756	~0	3588	~0	0.00
VITIS_LOOP_6_1	II Violation			181	2		yes									

Рисунок 2.8 – Копия экрана вкладки HLS Synthesis (конвейер)

binary\_container\_1 (Hardware) ×

var012\_no\_pragmas (Hardware) ×

var012\_pipelined (Hardware) ×

var012\_unrolled (Hardware) ×

Summary ×

HLS Synthesis ×

DATE: Thu Dec 23 21:56:27 2021

VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)

PROJECT: var012\_no\_pragmas

SOLUTION: solution (Vitis Kernel Flow Target)

PRODUCT FAMILY: virtexuplus

T

Q

⌵

⌶

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var012_no_pragmas	II Violation				0		no	2	~0	0	0	3756	~0	3588	~0	0.00
VITIS_LOOP_6_1	II Violation			181	2		yes									

Рисунок 2.9 – Копия экрана вкладки HLS Synthesis (разворачивание)

binary\_container\_1 (Hardware)

var012\_no\_pragmas (Hardware)

var012\_pipelined (Hardware)

var012\_unrolled (Hardware)

var012\_pipe\_unroll (Hardware)

Summary

HLS Synthesis

DATE: Thu Dec 23 21:56:59 2021

VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)

PROJECT: var012\_pipe\_unroll

SOLUTION: solution (Vitis Kernel Flow Target)

PRODUCT FAMILY: virtexuplus

T

Q

⌕

⌵

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var012_pipe_unroll							no	2	~0	0	0	2239	~0	3079	~0	0.00
dataflow_in_loop_vitis_loop_6_1		248	827.000			249	dataflow	0	0	0	0	932	~0	1891	~0	0.00
Block_entry_proc_proc		248	827.000			248	no	0	0	0	0	932	~0	1891	~0	0.00
VITIS_LOOP_6_1							no									

Рисунок 2.10 – Копия экрана вкладки HLS Synthesis (конвейер + разворачивание)

## 2.5 Вывод

По представленным результатам работы приложения в различных вариантах сборки можно сделать следующие выводы:

- при компиляции кода ядра для работы на ЦПУ хост-системы с помощью прагм оптимизации удалось уменьшить время обработки массива, при этом конвейеризация дала уменьшение времени в 5 раз, развернутый цикл дал выигрыш только 1.7 раза, а совмещение методов оптимизации – в 4 раза (что объясняется значительным уменьшением времени обработки за счет конвейерной обработки и незначительным увеличением за счет развернутого цикла);
- при компиляции кода ядра в аппаратную модель и её запуске в специальном симуляторе на ЦПУ, время обработки массива значительно увеличилось по сравнению с работой приложения на ЦПУ хост-системы, такие результаты можно объяснить собственно наличием посредника для выполнения приложения;
- при компиляции кода ядра в аппаратную модель и её реализации на FPGA не удалось достичь оптимизации по времени выполнения: конвейерная реализация работает только в 1.02 раза быстрее реализации без оптимизаций, при этом обе реализации с развернутым циклом работают в 13-18 раз медленнее реализации без оптимизаций, что можно объяснить замедлением загрузки данных из памяти для данного индивидуального задания при развертке цикла, а также использованием автоматической развертки цикла.



### 3 Контрольные вопросы

В данном разделе представлены ответы на контрольные вопросы.

- **Назовите преимущества и недостатки аппаратных ускорителей на ПЛИС по сравнению с CPU и графическими ускорителями**

Преимущества:

- создание специализированной вычислительной структуры для реализации желаемой функциональности;
- низкая стоимость в сравнении с обычными аппаратными ускорителями;
- большая частота эмуляции;
- компактность.

Недостатки:

- 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.

- **Назовите основные способы оптимизации циклических конструкций ЯВУ, реализуемых в виде аппаратных ускорителей**

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

- **Назовите этапы работы программной части ускорителя в хост системе**

Этап 1. Инициализация среды OpenCL.

Этап 2. Создание трех буферов, необходимых для обмена данными с ядром: два буфера для передачи исходных данных и один для вывода результата.

Этап 3. Запуск задачи на исполнение.

Этап 4. Чтение выходного буфера, содержащего результаты работы ядра, после завершения работы всех команд.

— **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.

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

- компилятор Xilinx Vitis v++;
- Assistant View (получение отчетов о сборке аппаратных ядер);
- внутрисхемный отладчик Vivado (отслеживание любых сигналов ускорителя для анализа событий);
- сводный отчет Link Summary (Vitis Analyzer, получение диаграмм системы и платформы и др.)

# Заключение

В ходе лабораторной работы были изучены методики и технологии синтеза аппаратных устройств ускорения вычислений по описаниям на языках высокого уровня.

Были выполнены следующие задачи:

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

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