



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 курсу "Архитектура ЭВМ"

Тема Разработка ускорителей вычислений средствами САПР высокоуровневого
синтеза Xilinx Vitis HLS

Студент Цветков И.А.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Дубровин Е.Н.

Москва — 2021 г.

Содержание

1	Теоритические основы	4
1.1	Методология ускорения вычислений на основе ПЛИС	4
1.2	Оптимизация времени обработки и пропускной способности . .	6
1.2.1	Конвейерная обработка циклов	6
1.2.2	Разворачивание циклов	7
1.2.3	Потоковая обработка	7
2	Выполнение лабораторной работы	8
2.1	Функции ядра по индивидуальному варианту	8
2.2	Режим Emulation-SW	12
2.3	Режим Emulation-HW	12
2.4	Режим HardWare	13
2.5	Обоснование результатов и выводы	15
	Заключение	16

Введение

Основной целью данной работы является 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 C/C++, изучаются принципы работы IDE Xilinx Vitis HLS и методика анализа и отладки устройств. В ходе работы необходимо разработать ускоритель вычислений по индивидуальному заданию, разработать код для тестирования ускорителя, реализовать ускоритель с помощью средств высоко-уровневого синтеза, выполнить его отладку.

1 Теоритические основы

При выполнении лабораторной работы будет использоваться ускоритель вычислений на **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 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 (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.

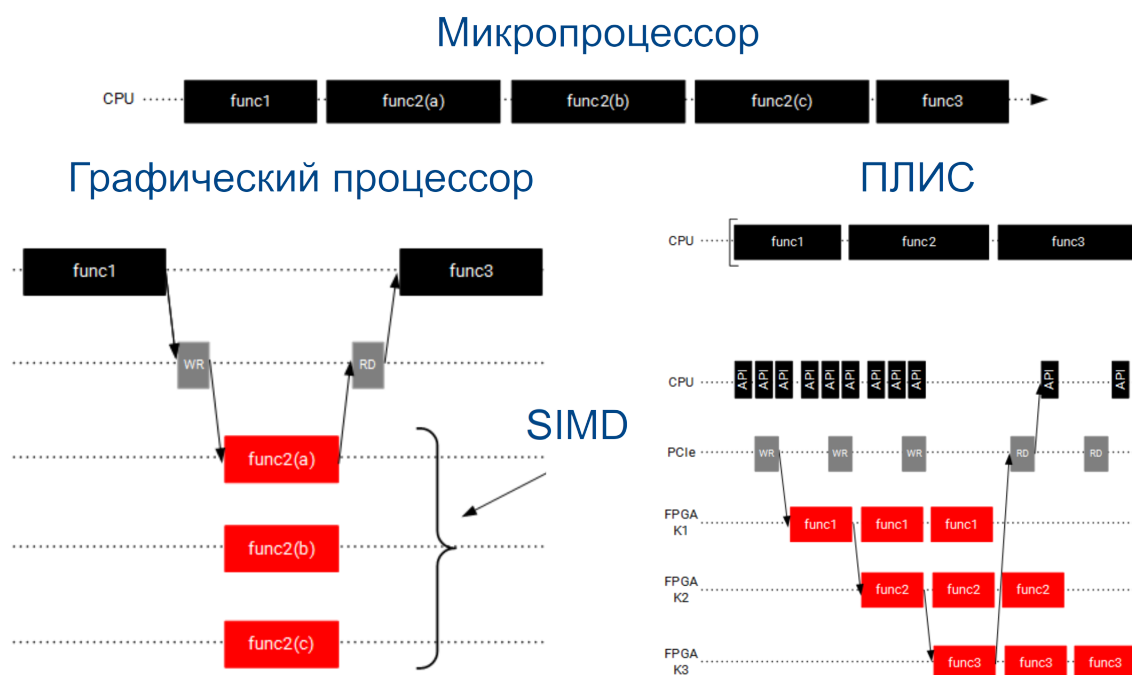


Рисунок 1.1 – Принципы организации вычислений на различных платформах

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) 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 больший 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 Потокковая обработка

Реализация механизма потокковой 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, потокковая 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 Выполнение лабораторной работы

Для изучения технологии будут выполнены следующие задания.

2.1 Функции ядра по индивидуальному варианту

В листинге 2.1 представлен листинг программы без оптимизаций. Также в листинге 2.1 представлена конвейрная оптимизация прогаммы, а в листинге 2.3 находится оптимизация частично развернутым циклом. В листиге 2.4 – конвейрная реализация с частично развернутым циклом.

Листинг 2.1 – Код программы (без изменений)

```
1  extern "C" {
2      void var019(int* c, const int* a, const int* b, const int
        len)
3      {
4          int tmpA;
5          int tmpB;
6          int tmpC;
7
8          for(int i = 0; i < len; i++)
9          {
10             tmpA = a[i];
11             tmpB = b[i];
12
13             for (int j = 0; j < 8; j++)
14             {
15                 tmpA = a[i] & 0x7;
16                 tmpB = b[i] & 0x7;
17                 tmpC = (tmpA + tmpB) & 0x7;
18
19                 tmpA = tmpA>>4;
20                 tmpB = tmpB>>4;
21                 tmpC = tmpA<<4;
22             }
23
24             c[i] = tmpC;
```



```

25         }
26     }
27 }

```

Листинг 2.2 – Код программы (конвертная)

```

1  extern "C" {
2      void var019_pipelined(int* c, const int* a, const int* b,
3          const int len)
4      {
5          #pragma HLS PIPELINE
6
7          int tmpA;
8          int tmpB;
9          int tmpC;
10
11         for(int i = 0; i < len; i++)
12         {
13             tmpA = a[i];
14             tmpB = b[i];
15
16             for (int j = 0; j < 8; j++)
17             {
18                 tmpA = a[i] & 0x7;
19                 tmpB = b[i] & 0x7;
20                 tmpC = (tmpA + tmpB) & 0x7;
21
22                 tmpA = tmpA>>4;
23                 tmpB = tmpB>>4;
24                 tmpC = tmpA<<4;
25             }
26
27             c[i] = tmpC;
28         }
29     }

```

Листинг 2.3 – Код программы (частично развернутый цикл)

```

1  extern "C" {
2      void var019_unrolled(int* c, const int* a, const int* b,
3          const int len)

```

```

3      {
4          #pragma HLS UNROLL
5
6          int tmpA;
7          int tmpB;
8          int tmpC;
9
10         for(int i = 0; i < len; i++)
11         {
12             tmpA = a[i];
13             tmpB = b[i];
14
15             for (int j = 0; j < 8; j++)
16             {
17                 tmpA = a[i] & 0x7;
18                 tmpB = b[i] & 0x7;
19                 tmpC = (tmpA + tmpB) & 0x7;
20
21                 tmpA = tmpA>>4;
22                 tmpB = tmpB>>4;
23                 tmpC = tmpA<<4;
24             }
25
26             c[i] = tmpC;
27         }
28     }
29 }

```

Листинг 2.4 – Код программы (конвейрный с частично развернутым циклом)

```

1      extern "C" {
2          void var019_pipe_unroll(int* c, const int* a, const int* b,
3                                  const int len)
4          {
5              #pragma HLS DATAFLOW
6
7              int tmpA;
8              int tmpB;
9              int tmpC;
10
11             for(int i = 0; i < len; i++)

```

```

11      {
12          tmpA = a[i];
13          tmpB = b[i];
14
15          for (int j = 0; j < 8; j++)
16          {
17              tmpA = a[i] & 0x7;
18              tmpB = b[i] & 0x7;
19              tmpC = (tmpA + tmpB) & 0x7;
20
21              tmpA = tmpA>>4;
22              tmpB = tmpB>>4;
23              tmpC = tmpA<<4;
24          }
25
26          c[i] = tmpC;
27      }
28  }
29  }

```

2.2 Режим Emulation-SW

На рисунке 2.1 представлен результат работы приложения в режиме **Emulation-SW**.

```
[Console output redirected to file: /iu_home/lu7137/workspace/lu7_53b_19_lab02/Emulation-SW/SystemDebugger/lu7_53b_19_lab02_system/lu7_53b_19_lab02.launch.log]
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/lu7137/workspace/lu7_53b_19_lab02_system/Emulation-SW/binary_container_1.xclbin
Loading: /iu_home/lu7137/workspace/lu7_53b_19_lab02_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) |
|.....|
| var019_no_pragmas | 4833434 |
|.....|
| var019_unrolled | 3351891 |
|.....|
| var019_pipelined | 1994515 |
|.....|
| var019_pipe_unroll | 2583365 |
|.....|
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 2.1 – Результаты работы приложения в режиме Emulation-SW

2.3 Режим Emulation-HW

На рисунке 2.2 представлена копия экрана **Assistant View** для **Emulation-HW**, а на рисунке 2.3 представлен результат работы приложения в режиме **Emulation-HW**. А также на рисунке 2.4 представлена схема отладчика Vivado.

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"/>
var019_no_pragmas	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
var019_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							
var019_pipe_unroll	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
var019_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							
var019_pipelined	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>
var019_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							
var019_unrolled	1	Auto	Auto	<input type="checkbox"/>	Default (Counters + Trace)	✓	<input type="checkbox"/>

Рисунок 2.2 – Assistant View для Emulation-HW

```
[Console output redirected to file:/iu_home/lu7137/workspace/lu7_53b_19_lab02/Emulation-HW/SystemDebugger/lu7_53b_19_lab02_system/lu7_53b_19_1
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/lu7137/workspace/lu7_53b_19_lab02_system/Emulation-HW/binary_container_1.xclbin
Loading: /iu_home/lu7137/workspace/lu7_53b_19_lab02_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
Device[0]: program successful!

-----
| Kernel | Wall-Clock Time (ns) |
|-----|-----|
| var019_no_pragmas | 7006204711 |
| var019_unrolled | 6003397749 |
|-----|-----|
INFO: [Vitis-EM 22] [Time elapsed: 3 minute(s) 21 seconds, Emulation time: 0.181231 ms]
Data transfer between kernel(s) and global memory(s)
var019_no_pragmas_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 4.000 KB
var019_pipe_unroll_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 4.000 KB
var019_pipelined_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 2.000 KB
var019_unrolled_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 4.000 KB

| var019_pipelined | 10005026316 |
|-----|-----|
| var019_pipe_unroll | 7003231552 |
|-----|-----|
Notes: 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: 5 minute(s) 57 seconds, Emulation time: 0.326327 ms]
Data transfer between kernel(s) and global memory(s)
var019_no_pragmas_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 4.000 KB
var019_pipe_unroll_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 4.000 KB
var019_pipelined_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 4.000 KB
var019_unrolled_1:m_axi_gmem-DDR[1] RD = 0.000 KB WR = 4.000 KB

INFO: [HW-EMU 06-0] Waiting for the simulator process to exit
INFO: [HW-EMU 06-1] All the simulator processes exited successfully
```

Рисунок 2.3 – Результаты работы приложения в режиме Emulation-HW

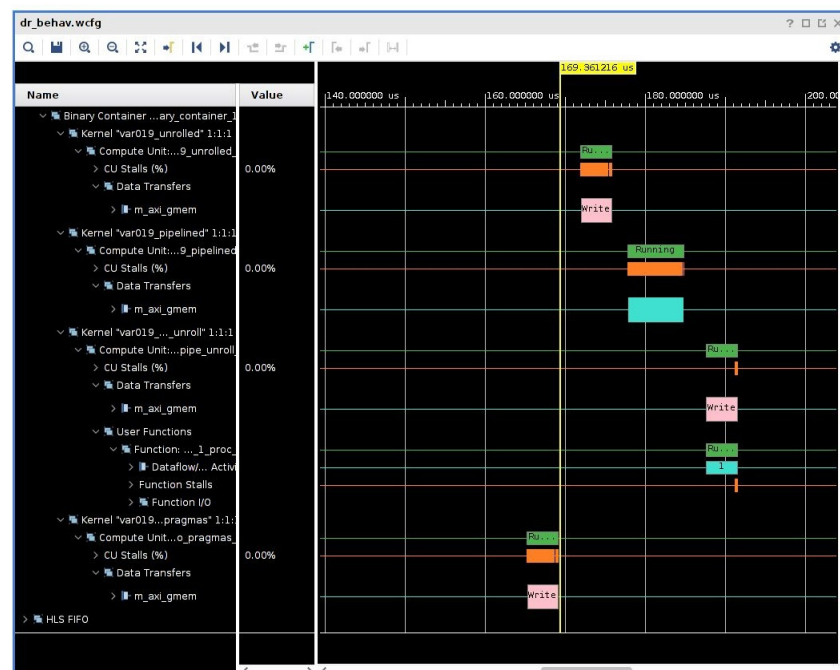


Рисунок 2.4 – Схема отладчика Vivado для Emulation-HW

2.4 Режим HardWare

На рисунке ?? представлен результат работы приложения в режиме **HardWare**. Также на рисунке 2.5 представлен снимок экрана вкладки **Summary**, на рисунке 2.6 – вкладки **System Diagram**, на рисунке 2.7 – вкладки **Platform Diagram**. При этом на рисунках 2.8–2.9 представлены снимки экрана **HLS Synthesis** для каждой из функций.

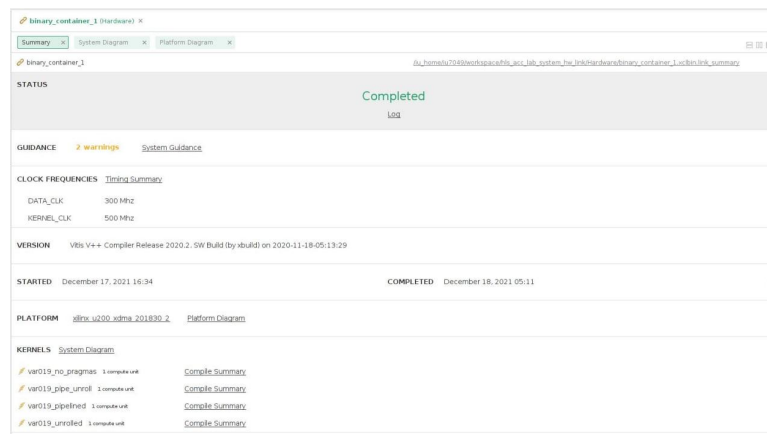


Рисунок 2.5 – Вкладка Summary

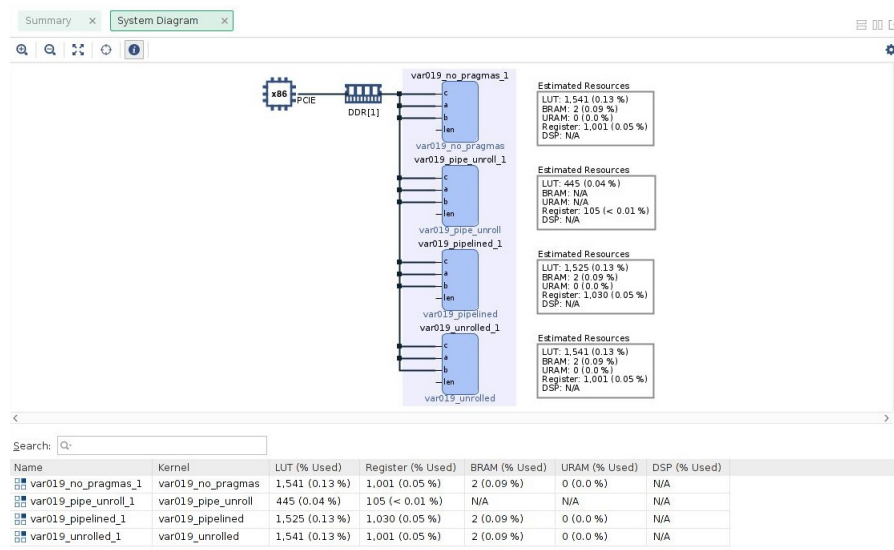


Рисунок 2.6 – Вкладка System Diagram

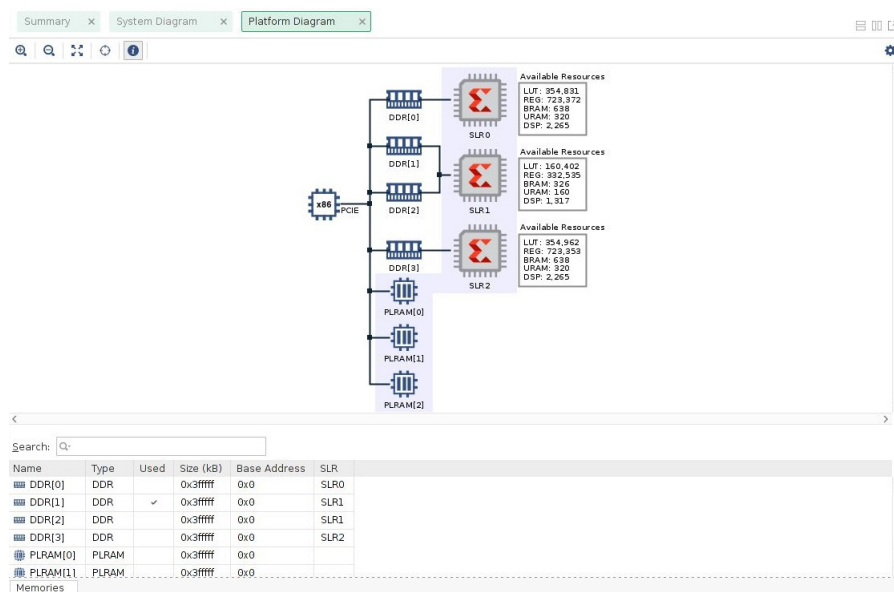


Рисунок 2.7 – Вкладка Platform Diagram

DATE: Sat Dec 25 01:15:06 2021

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

PROJECT: var019_no_pragmas

SOLUTION: solution (Vitis Kernel Flow Target)

T

Q

+

+

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var019_no_pragmas					0		no	2	~0	0	0	1001	~0	1541	~0	0.00
VTIS_LOOP_8_1				2	1		yes									

Рисунок 2.8 – Вкладка HLS Synthesis для var019_no_pragmas

DATE: Sat Dec 25 01:15:08 2021

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

PROJECT: [var019_pipe_unroll](#)

SOLUTION: solution (Vitis Kernel Flow Target)

T

Q

+

+



Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var019_pipe_unroll							dataflow	2	~0	0	0	904	~0	1513	~0	0.00
 Loop_VTIS_LOOP_10_1_proc							no	0	0	0	0	105	~0	445	~0	0.00
 VTIS_LOOP_10_1				2	1		yes									

Рисунок 2.9 – Вкладка HLS Synthesis для var019_pipe_unroll

DATE: Sat Dec 25 01:15:08 2021

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

PROJECT: var019_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
var019_pipelined						0										
VTIS_LOOP_10_1				2			no	2	~0	0	0	1030	~0	1525	~0	0.00
							no									

Рисунок 2.10 – Вкладка HLS Synthesis для var019_pipelined

DATE: Sat Dec 25 01:15:07 2021

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

PROJECT: var019_unrolled

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
var019_unrolled					0		no	2	~0	0	0	1001	~0	1541	~0	0.00
VTIS_LOOP_10_1				2	1		yes									

Рисунок 2.11 – Вкладка HLS Synthesis для var019_unrolled

2.5 Обоснование результатов и выводы

Как видно из результатов, наибольшее время выполнения у цикла без оптимизаций. При этом наименьшее время было достигнуто при развернутом цикле. Это связано с тем, что все развернутые итерации выполняются параллельно, при этом уменьшается количество итераций.

Заключение

В данной лабораторной работе были рассмотрены и изучены технологии синтеза аппаратных устройств ускорения вычислений по языкам высокого уровня.