



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

О Т Ч Е Т

по лабораторной работе № 5

Название: Разработка ускорителей вычислений средствами САПР
высокоуровневого синтеза Xilinx Vitis HLS

Дисциплина: Архитектура ЭВМ

Студент

ИУ7-52Б

(Группа)

(Подпись, дата)

И.С. Климов

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

А.Ю. Попов

(И.О. Фамилия)

Москва, 2021

Содержание

Введение.....	3
1. Исходные данные	4
2. Режим программной эмуляции (Emulation-SW)	6
3. Режим аппаратной эмуляции (Emulation-HW).....	7
4. Режим аппаратного исполнения (Hardware)	9
Вывод	13
Контрольные вопросы	14
1. Вопрос 1.....	14
2. Вопрос 2.....	14
3. Вопрос 3.....	15
4. Вопрос 4.....	16
5. Вопрос 5.....	16

Введение

Цель работы – изучить методики и технологии синтеза аппаратных устройств ускорения вычислений по описаниям на языках высокого уровня. Для достижения цели были поставлены и решены следующие **задачи**:

- изучить маршрут проектирования устройств, представленных в виде синтаксических конструкций языков высокого уровня C/C++;
- изучить основные возможности, средства отладки и анализа, которые предоставляет IDE Xilinx Vitis HLS для разработчиков ускорителей;
- разработать программу для ускорителя вычислений по индивидуальному заданию и протестировать ее.

1. Исходные данные

На рисунке 1.1 представлен изначальный код программы в соответствии с вариантом 8.

```
1 extern "C" {  
2 void var008_no_pragmas(int* c, const int* a, const int* b, const int len) {  
3     int min = 1000;  
4     for (int i = 0; i < len; i++) {  
5         if (min > a[i]) min = a[i];  
6         if (min > b[i]) min = b[i];  
7     }  
8     for (int i = 0; i < len; i++) {  
9         c[i] = min + i;  
10    }  
11 }  
12 }
```

Рисунок 1.1 – изначальный код программы

На рисунках 1.2 – 1.4 приведены коды программ, в которых используются директивы, указывающие компилятору, как оптимизировать программу.

```
1 extern "C" {  
2 void var008_pipelined(int* c, const int* a, const int* b, const int len) {  
3     int min = 1000;  
4     for (int i = 0; i < len; i++) {  
5         #pragma HLS PIPELINE  
6         if (min > a[i]) min = a[i];  
7         if (min > b[i]) min = b[i];  
8     }  
9     for (int i = 0; i < len; i++) {  
10        #pragma HLS PIPELINE  
11        c[i] = min + i;  
12    }  
13 }  
14 }
```

Рисунок 1.2 – Оптимизированный код программы с директивой, указывающей препроцессору на конвейерную обработку циклов

```
1 extern "C" {  
2 void var008_unrolled(int* c, const int* a, const int* b, const int len) {  
3     int min = 1000;  
4     for (int i = 0; i < len; i++) {  
5         #pragma HLS UNROLL factor=2  
6         if (min > a[i]) min = a[i];  
7         if (min > b[i]) min = b[i];  
8     }  
9     for (int i = 0; i < len; i++) {  
10        #pragma HLS UNROLL factor=2  
11        c[i] = min + i;  
12    }  
13 }  
14 }
```

Рисунок 1.3 – Оптимизированный код программы с директивой, указывающей препроцессору на разворачивание циклов

```

1 extern "C" {
2 void var008_pipe_unroll(int* c, const int* a, const int* b, const int len) {
3     int min = 1000;
4     for (int i = 0; i < len; i++) {
5         #pragma HLS UNROLL factor=2
6         if (min > a[i]) min = a[i];
7         if (min > b[i]) min = b[i];
8     }
9     for (int i = 0; i < len; i++) {
10        #pragma HLS PIPELINE
11        c[i] = min + i;
12    }
13 }
14 }

```

Рисунок 1.4 – Оптимизированный код программы с директивами, указывающими препроцессору на конвейерную обработку одного цикла и разворачивание второго

2. Режим программной эмуляции (Emulation-SW)

На рисунке 2.1 представлен результат работы в режиме программной эмуляции.

```
[Console output redirected to file:/iu_home/iu7038/workspace/lab_05/Emulation-SW/SystemDebug
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7038/workspace/lab_05_system/Emulation-SW/binary_container_1.xclbin
Loading: '/iu_home/iu7038/workspace/lab_05_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) |
|-----+-----|
| var008_no_pragmas | 10132818 |
|-----+-----|
| var008_unrolled | 2853731 |
|-----+-----|
| var008_pipelined | 986147 |
|-----+-----|
| var008_pipe_unroll | 858336 |
|-----+-----|
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 – Результат работы в режиме Emulation-SW (сравнительная таблица со временем выполнения)

3. Режим аппаратной эмуляции (Emulation-HW)

На рисунках 3.1 – 3.2 представлены копии экрана с открытым Assistant View.

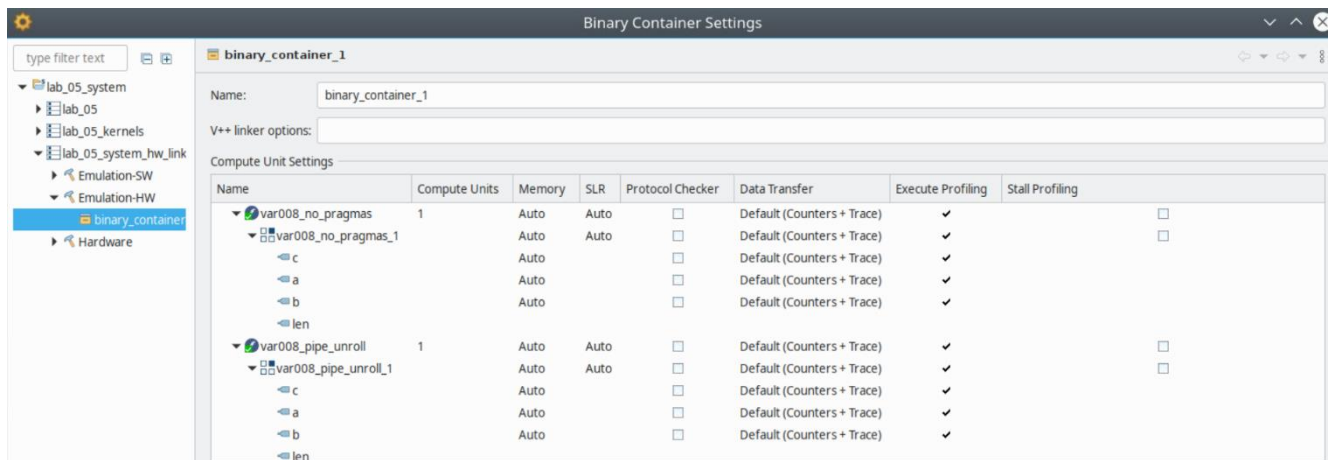


Рисунок 3.1 – Копия экрана Assistant View (часть 1)

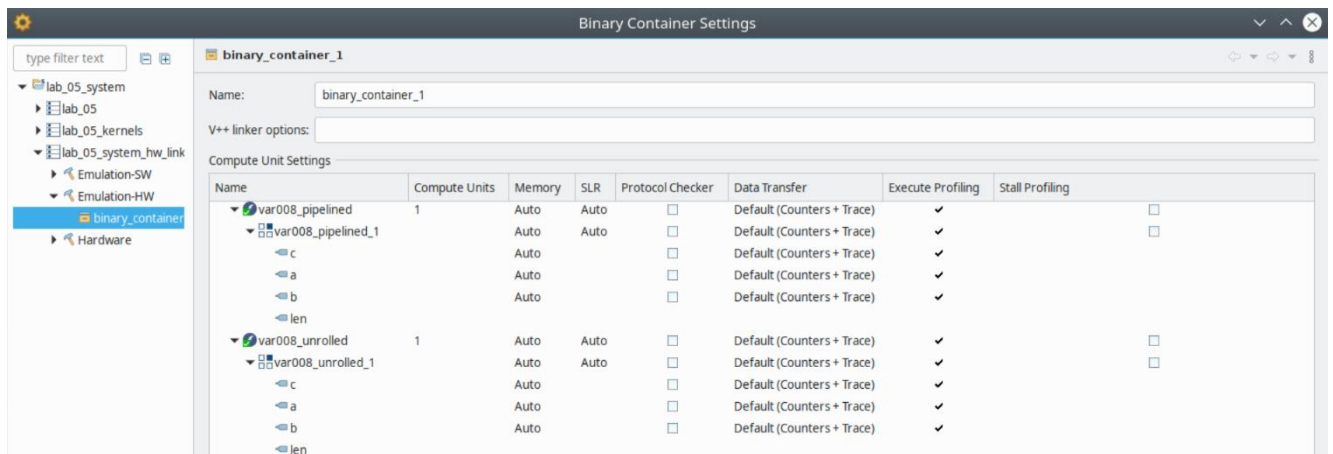


Рисунок 3.2 – Копия экрана Assistant View (часть 2)

На рисунке 3.3 представлен результат работы приложения в режиме аппаратной эмуляции (окно внутрисхемного отладчика Vivado с диаграммами работы четырех ядер).

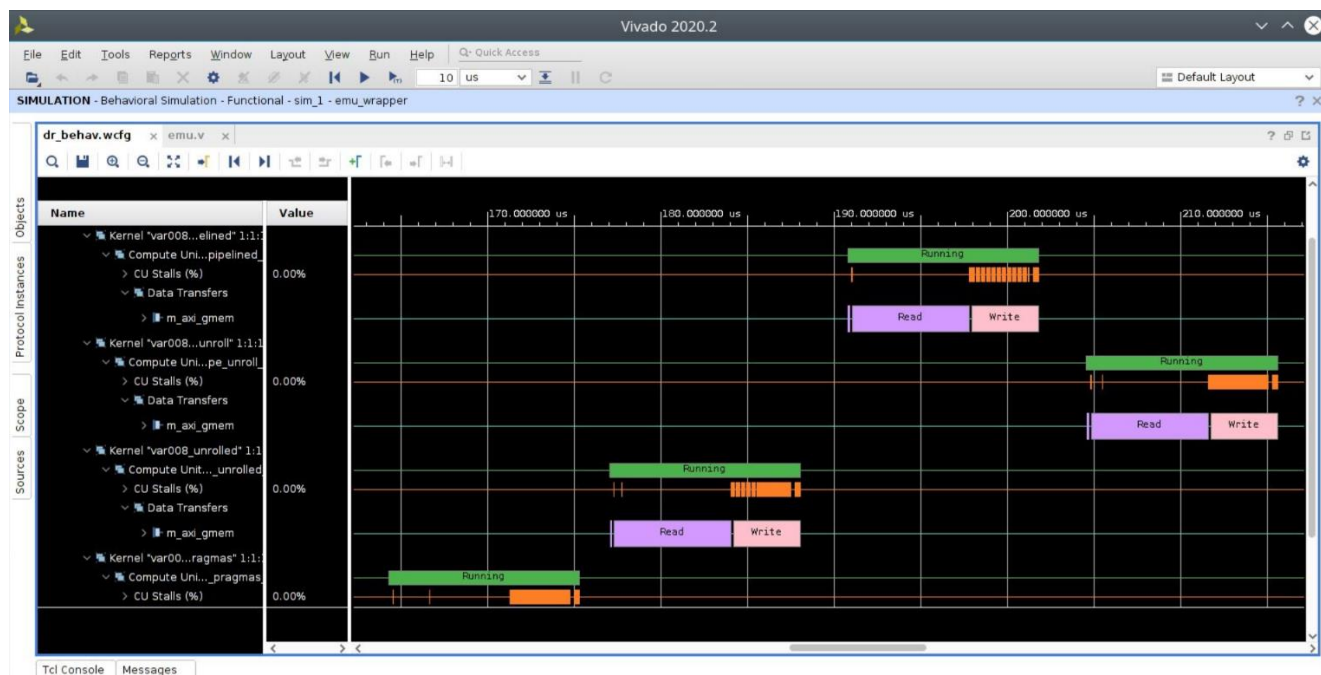


Рисунок 3.3 – Результат работы в режиме Emulation-SW (диаграммы работы четырех ядер)

На рисунке 3.4 представлена сравнительная таблица со временем выполнения.

```
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7038/workspace/lab_05_system/Emulation-HW/binary_container_1.xclbin
Loading: '/iu_home/iu7038/workspace/lab_05_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
Device[0]: program successful!

+-----+
| Kernel | Wall-Clock Time (ns) |
+-----+
| var008_no_pragmas | 10005845174 |
+-----+
| var008_unrolled | 11004073230 |
+-----+
| var008_pipelined | 11003678374 |
+-----+
| var008_pipe_unroll | 11003006746 |
+-----+

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 3.4 – Результат работы в режиме Emulation-HW (сравнительная таблица со временем выполнения)

4. Режим аппаратного исполнения (Hardware)

На рисунке 4.1 представлена сравнительная таблица со временем выполнения.

```
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7038/workspace/lab_05_system/Hardware/binary_container_1.xclbin
Loading: '/iu_home/iu7038/workspace/lab_05_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)
var008_no_pragmas	111757149
var008_unrolled	109381902
var008_pipelined	109974944
var008_pipe_unroll	108813841

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 4.1 – Результат работы в режиме Hardware (сравнительная таблица со временем выполнения)

На рисунках 4.2 – 4. представлены копии экранов со вкладок из Link Summary.

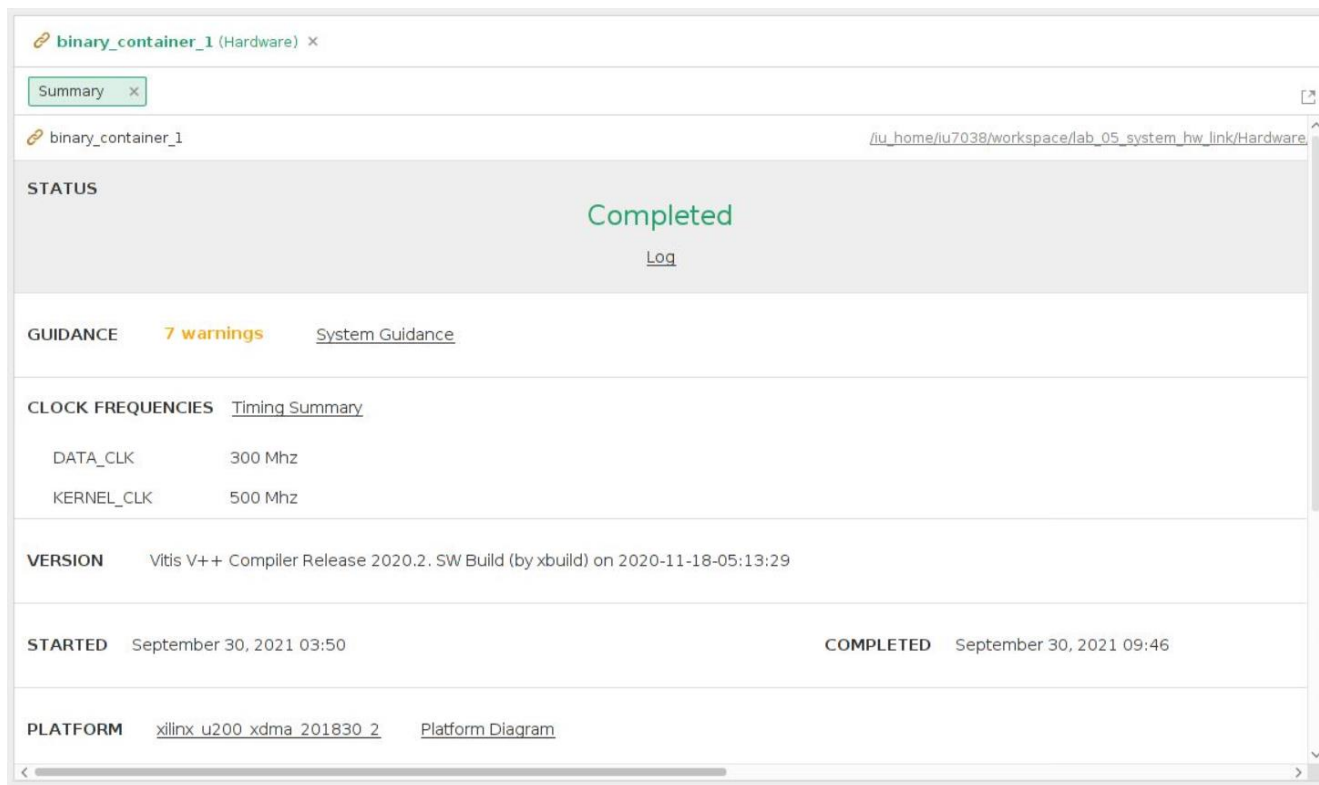


Рисунок 4.2 – Копия экрана со вкладкой Summary

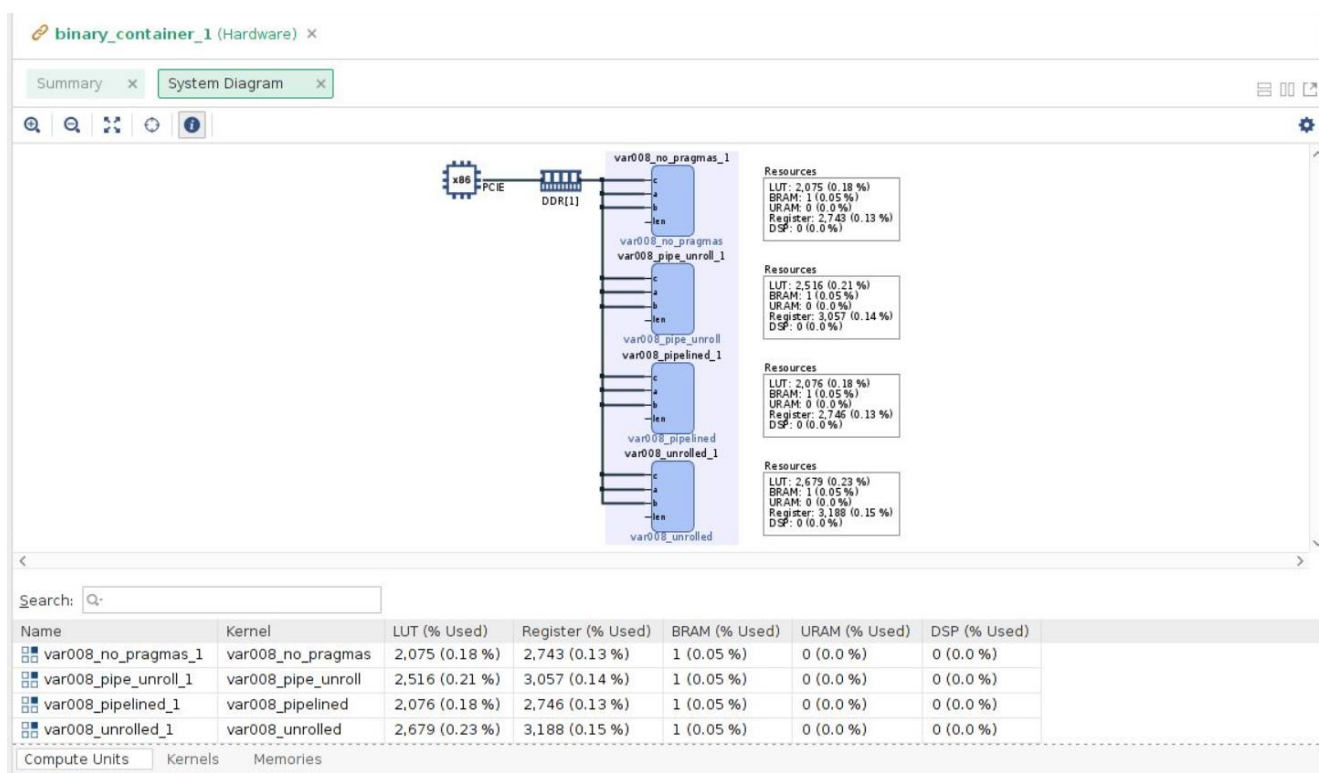


Рисунок 4.3 – Копия экрана со вкладки System Diagram

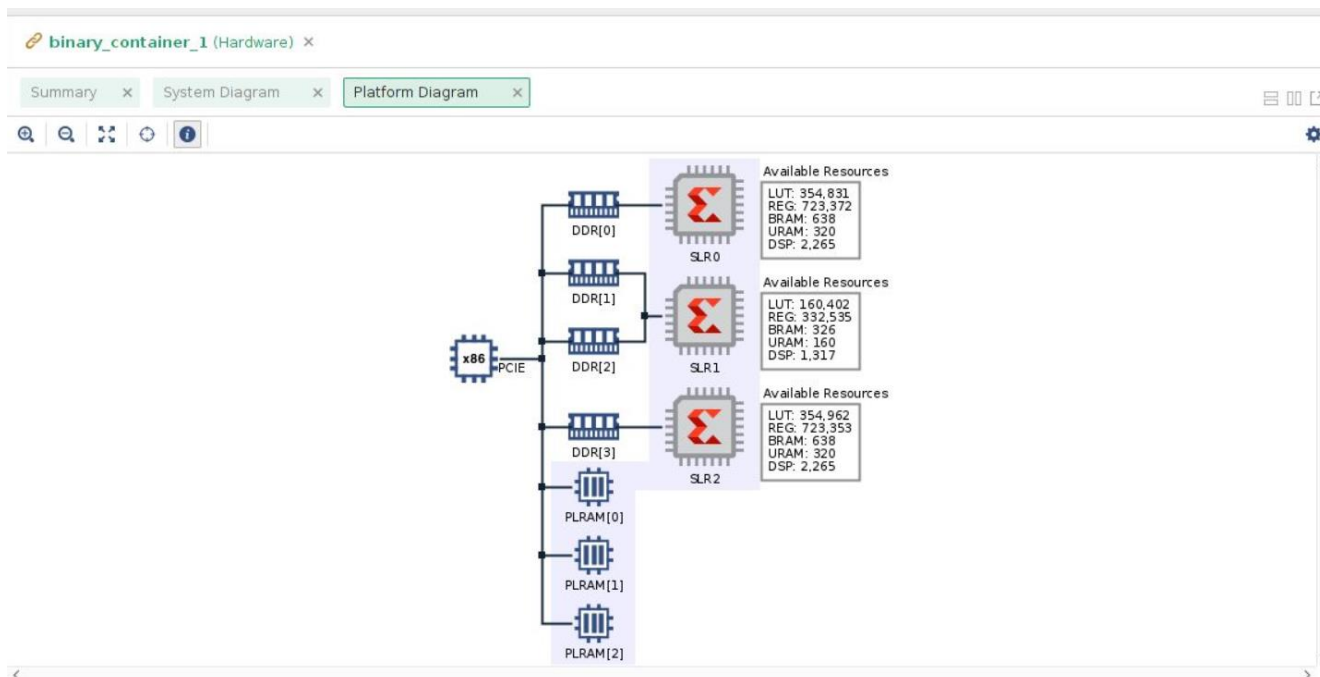


Рисунок 4.4 – Копия экрана со вкладки Platform Diagram (часть 1)

Search: Q

Name	Type	Used	Size (kB)	Base Address	SLR
DDR[0]	DDR		0x1000000	0x40_0000_0000	SLR0
DDR[1]	DDR	✓	0x1000000	0x50_0000_0000	SLR1
DDR[2]	DDR		0x1000000	0x60_0000_0000	SLR1
DDR[3]	DDR		0x1000000	0x70_0000_0000	SLR2
PLRAM[0]	PLRAM		0x80	0x30_0000_0000	
PLRAM[1]	PLRAM		0x80	0x30_0020_0000	
PLRAM[2]	PLRAM		0x80	0x30_0040_0000	

Memories

Рисунок 4.5 – Копия экрана со вкладки Platform Diagram (часть 2)

binary_container_1 (Hardware) x var008_no_pragmas (Hardware) x

Summary x HLS Synthesis x

DATE: Thu Sep 30 03:47:11 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: var008_no_pragmas SOLUTION: solution (Vitis Kernel Flow Target)

Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var008_no_pragmas	II Violation					0	no	2	~0	0	0	1517	~0	2017	~0	0.00
VTIS_LOOP_4_1	II Violation			74	2		yes									
VTIS_LOOP_8_2				2	1		yes									

Рисунок 4.6 – Копия экрана со вкладки HLS Synthesis (без оптимизаций)

<div> binary_container_1 (Hardware) × var008_no_pragmas (Hardware) × var008_pipe_unroll (Hardware) × var008_pipelined (Hardware) × </div>																
<div> Summary × HLS Synthesis × </div>																
<div> DATE: Thu Sep 30 03:47:11 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: <u>var008_pipelined</u> SOLUTION: solution (Vitis Kernel Flow Target) </div>																
<div> T Q ≡ ⚙ </div>																
Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var008_pipelined	II Violation				0		no	2	~0	0	0	1517	~0	2017	~0	0.00
VTIS_LOOP_4_1	II Violation			74	2		yes									
VTIS_LOOP_9_2	II Violation			2	1		yes									

Рисунок 4.6 – Копия экрана со вкладки HLS Synthesis (конвейерная обработка циклов)

<div> binary_container_1 (Hardware) × var008_no_pragmas (Hardware) × var008_pipe_unroll (Hardware) × var008_pipelined (Hardware) × var008_unrolled (Hardware) × </div>																
<div> Summary × HLS Synthesis × </div>																
<div> DATE: Thu Sep 30 03:47:12 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: <u>var008_unrolled</u> SOLUTION: solution (Vitis Kernel Flow Target) </div>																
<div> T Q ≡ ⚙ </div>																
Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var008_unrolled	II Violation				0		no	2	~0	0	0	2022	~0	2720	~0	0.00
VTIS_LOOP_4_1	II Violation			76	4		yes									
VTIS_LOOP_9_2	II Violation			3	2		yes									

Рисунок 4.6 – Копия экрана со вкладки HLS Synthesis (развертывание циклов)

<div> binary_container_1 (Hardware) × var008_no_pragmas (Hardware) × var008_pipe_unroll (Hardware) × </div>																
<div> Summary × HLS Synthesis × </div>																
<div> DATE: Thu Sep 30 03:47:11 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: <u>var008_pipe_unroll</u> SOLUTION: solution (Vitis Kernel Flow Target) </div>																
<div> T Q ≡ ⚙ </div>																
Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)	Slack
var008_pipe_unroll	II Violation				0		no	2	~0	0	0	1922	~0	2476	~0	0.00
VTIS_LOOP_4_1	II Violation			76	4		yes									
VTIS_LOOP_9_2	II Violation			2	1		yes									

Рисунок 4.6 – Копия экрана со вкладки HLS Synthesis (конвейеризация и развертывание циклов)

Вывод

Таким образом, были изучены методики и технологии синтеза аппаратных устройств ускорения вычислений по описаниям на языках высокого уровня.

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.

При разворачивании циклов все развернутые итераций цикла будут выполняться параллельно, поэтому для реализации такого варианта оптимизации потребуется больший объем программируемых логических ресурсов. В результате компилятор может столкнуться с проблемами, связанными с таким большим количеством ресурсов, и с проблемами емкости, которые замедляют процесс компиляции ядра.

Оба варианта оптимизации привели к уменьшению количества тактов, и как следствие, к уменьшению времени выполнения. Комбинирование двух методов также привело к более эффективному решению.

Контрольные вопросы

Ниже представлены ответ на контрольные вопросы.

1. Вопрос 1

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

Аппаратные ускорители в отличие от CPU и графических ускорителей представляют собой полностью настраиваемую архитектуру, которую разработчик может использовать для размещения вычислительных блоков с требуемой функциональностью. Возможность настроить аппаратное обеспечение под специализированную задачу позволяет достичь высокой производительности. Также ПЛИС позволяет достичь лучшего показателя энергоэффективности. Графические процессоры масштабируют производительность за счет большого количества ядер и использования параллелизма SIMD/SIMT.

2. Вопрос 2

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

К основным способам оптимизации циклических конструкций относится:

1. **Конвейеризация** – позволяет повысить пропускную способность за счет увеличения времени синтеза программы и требований к ресурсам ПЛИС. При конвейеризации вместо комплексного преобразования входных данных в одной сложной схеме используются последовательные простые операции, каждая из которых выполняется в своем цифровом узле, а промежуточные результаты запоминаются в триггерах. Это упрощение преобразований позволяет уменьшить число последовательных ячеек от триггера до триггера. Для указания компилятору о необходимости конвейеризировать циклическую обработку используется директива PIPELINE;
2. **Разворачивание циклов** – итерации начинают выполняться параллельно на собственном наборе оборудования. Количество тактов на выполнение

всего цикла уменьшается за счет роста размера схемы. Для указания компилятору о необходимости развернуть цикл используется директива UNROLL.

3. Вопрос 3

Назовите этапы работы программной части ускорителя в хост системе.

Этапы работы программной части ускорителя в хост системе:

- 1) вычисление размера массива;
- 2) объявление и инициализация исходных массивов;
- 3) получение списка устройств и инициализация контекста;
- 4) создание контекста и очередей команд к устройствам;
- 5) получение необходимой информации об устройстве;
- 6) создание программного объекта OpenCL и загрузка программы в двоичном формате на ускоритель;
- 7) выделение памяти под буферы устройства;
- 8) для запуска каждой реализации
 - a. установка необходимых для тестирования значения;
 - b. копирование содержимое буферов в DDR память ускорительной карты;
 - c. запуск задачи на исполнение и ожидание готовности по прерыванию;
 - d. чтение метки времени исполнения задачи;
 - e. чтение данных из DDR памяти устройства в буфер результатов.

4. Вопрос 4

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

1. При отладке в режиме программной эмуляции (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, понимания или проверки поведения системы.
2. Для отладки в режиме аппаратной эмуляции (Emulation-HW) код ядра компилируется в аппаратную модель (RTL), которая запускается в специальном симуляторе на ЦПУ. Этот вариант сборки занимает больше времени, но обеспечивает подробное и точное представление активности ядра. Данный вариант сборки полезен для тестирования функциональности ускорителя и получения начальных оценок производительности.
3. Для отладки в режиме аппаратного обеспечения (Hardware) код ядра компилируется в RTL, а затем реализуется на FPGA. В результате чего формируется двоичный файл xclbin, который будет работать на реальной FPGA.

5. Вопрос 5

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

Для оптимизации ускорителей можно использовать:

- 1) отладчик, имеющий графический интерфейс;
- 2) конструкции, указывающие компилятору путь оптимизации (прагмы и директивы (*set_directive_**));
- 3) средство анализа Vivado IDE, позволяющее также оценивать время и затраты после синтеза или размещения, выполнять симуляцию выполнения

программы на ускорителе, после высокоуровневого синтеза оптимизировать проекты на уровне межрегистровых передач.