СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является изучение методики и технологии синтеза аппаратных устройств ускорения вычислений по описаниям на языках высокого уровня.

Для достижения данной цели необходимо выполнить 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;

1.1 Исходные тексты программ

Ниже в листинге 1.1 приведен изначальный код программы (ядра для ускорителя) в соответствии с моим вариантом.

Листинг 1.1 — Код исходной программы

```
1   extern "C" {
2       void var014(int* c, const int* a, const int* b, const int len) {
3           int tmpA = 0;
4           int tmpB = 0;
5           for (int i = 0; i < len; i++) {
6               tmpA += a[i] * i;
7               tmpB += b[i] * i;
8           }
9           for (int i = 0; i < len; i+=2) {
10              c[i] = tmpA;
11              c[i+1] = tmpB;
12          }
13      }
14  }
```

Ниже в листингах 1.2 - 1.4 приведен код программ, в которых использовались директивы, указывающие компилятору оптимизировать программу.

Листинг 1.2 — Конвейерное исполнение

```
1   extern "C" {
2       void var014(int* c, const int* a, const int* b, const int len) {
3           int tmpA = 0;
4           int tmpB = 0;
5           for (int i = 0; i < len; i++) {
6               #pragma HLS PIPELINE
7               tmpA += a[i] * i;
8               tmpB += b[i] * i;
9           }
10          for (int i = 0; i < len; i+=2) {
11              #pragma HLS PIPELINE
12              c[i] = tmpA;
13              c[i+1] = tmpB;
14          }
15      }
16  }
```

Листинг 1.3 — Развернутый цикл

```
1   extern "C" {
2       void var014(int* c, const int* a, const int* b, const int len) {
3           int tmpA = 0;
4           int tmpB = 0;
5           for (int i = 0; i < len; i++) {
6               #pragma HLS UNROLL  factor=2
7               tmpA += a[i] * i;
8               tmpB += b[i] * i;
9           }
10          for (int i = 0; i < len; i+=2) {
11              #pragma HLS UNROLL  factor=2
12              c[i] = tmpA;
13              c[i+1] = tmpB;
14          }
15      }
16  }
```

Листинг 1.4 — Развернутый цикл и конвейерное исполнение

```
1   extern "C" {
2       void var014(int* c, const int* a, const int* b, const int len) {
3           int tmpA = 0;
4           int tmpB = 0;
5           for (int i = 0; i < len; i++) {
6               #pragma HLS UNROLL  factor=2
7               #pragma HLS PIPELINE
8               tmpA += a[i] * i;
9               tmpB += b[i] * i;
10          }
11          for (int i = 0; i < len; i+=2) {
12              #pragma HLS UNROLL  factor=2
13              #pragma HLS PIPELINE
14              c[i] = tmpA;
15              c[i+1] = tmpB;
16          }
17      }
18  }
```

1.2 Результаты работы

Ниже на рисунке 1.1 приведены результаты эмуляции выполнения ядра на центральном процессоре.

```
[Console output redirected to file:/iu_home/iu7044/workspace/lb2/hls_acc_lab/Emulation-SW/SystemDebu
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7044/workspace/lb2/hls_acc_lab_system/Emulation-SW/binary_container_1.xclbi
Loading: '/iu_home/iu7044/workspace/lb2/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)   |
|-------------------------+-------------------------|
| var014_no_pragmas       |                 2196574 |
|-------------------------+-------------------------|
| var014_unrolled         |                 2309105 |
|-------------------------+-------------------------|
| var014_pipelined        |                 3578290 |
|-------------------------+-------------------------|
| var014_pipe_unroll      |                 1709413 |
|-------------------------+-------------------------|
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 1.1 — Результаты программной эмуляции

Ниже на рисунках 1.2 - 1.3 приведена копия экрана с открытым Assistant view.

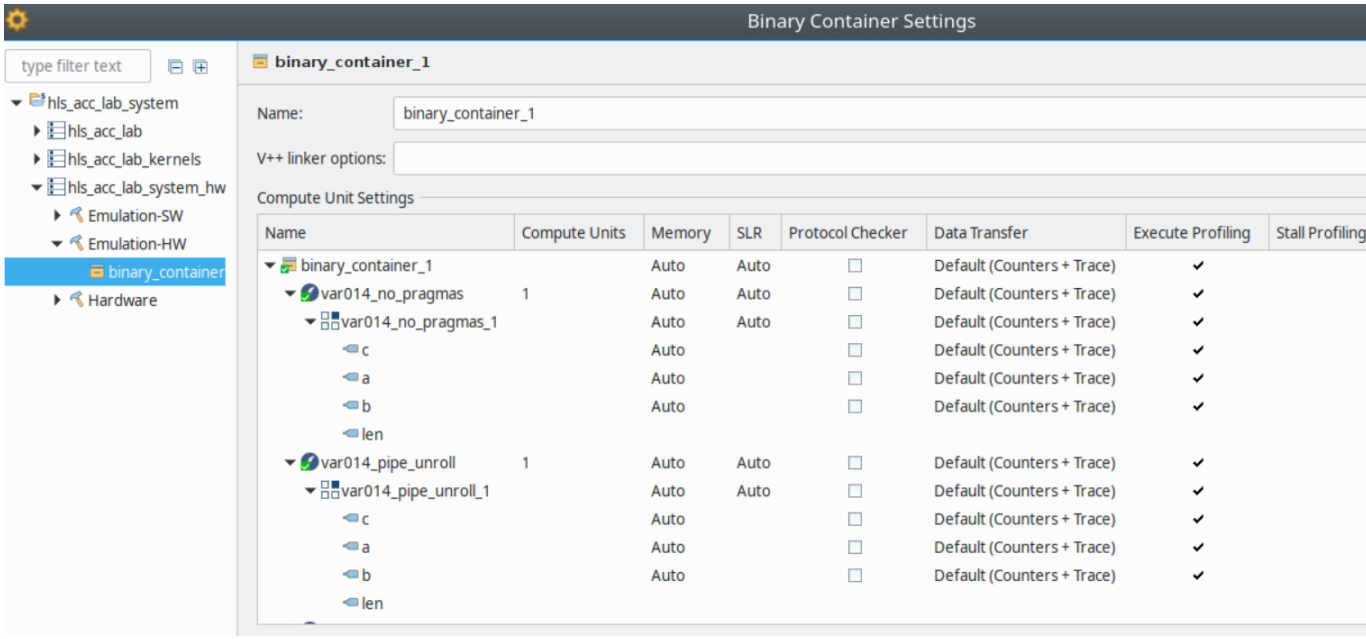


Рисунок 1.2 — Копия экрана с открытым Assistant view

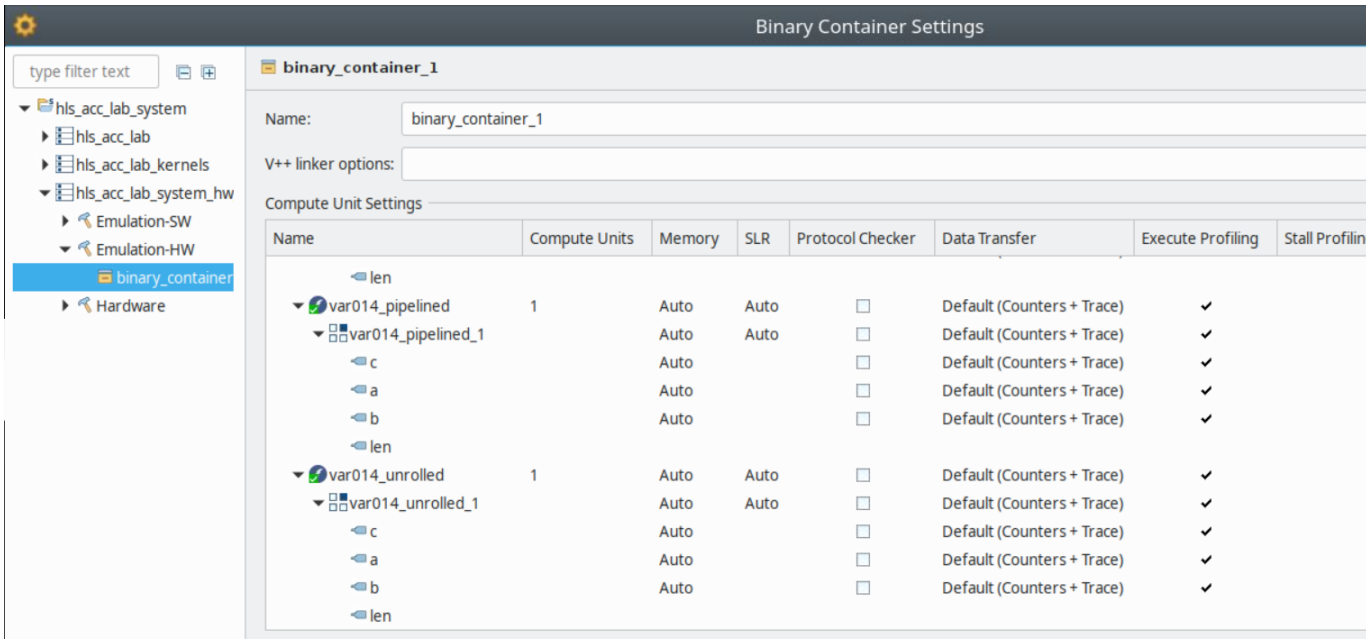


Рисунок 1.3 — Копия экрана с открытым Assistant view (продолжение)

Ниже на рисунке 1.4 приведены результаты работы приложения в режиме Emulation-HW.

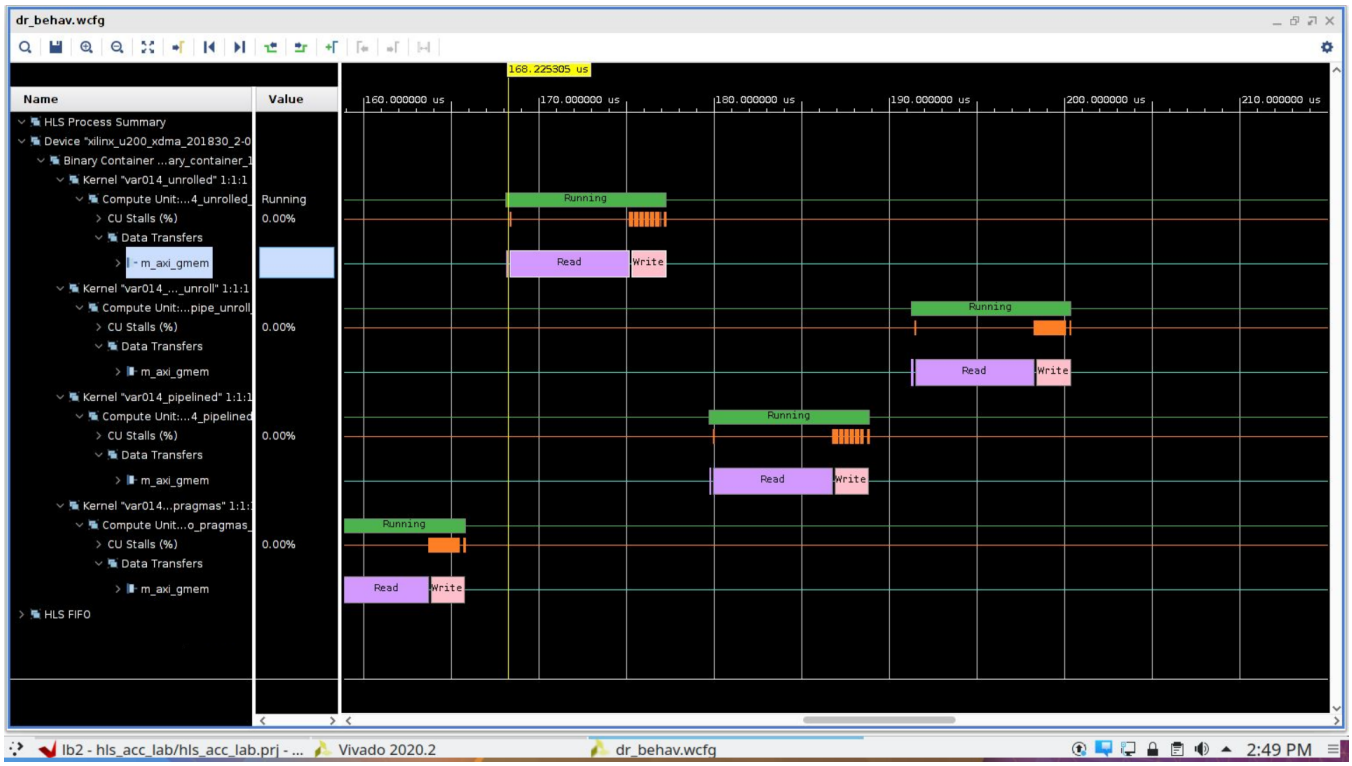


Рисунок 1.4 — Результаты работы приложения в режиме Emulation-HW

```
Device[0]: program successful!
|-------------------------+-------------------------|
| Kernel                  |    Wall-Clock Time (ns) |
|-------------------------+-------------------------|
| var014_no_pragmas       |             46023862455 |
|-------------------------+-------------------------|
| var014_unrolled         |             46024943015 |
|-------------------------+-------------------------|
| var014_pipelined        |             41027334211 |
|-------------------------+-------------------------|
| var014_pipe_unroll      |             38019985380 |
|-------------------------+-------------------------|
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: 12 minute(s) 23 seconds, Emulation time: 0.223836 ms]
Data transfer between kernel(s) and global memory(s)
var014_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 16.000 KB              WR = 4.000 KB
var014_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 16.000 KB              WR = 4.000 KB
var014_pipelined_1:m_axi_gmem-DDR[1]          RD = 16.000 KB              WR = 4.000 KB
var014_unrolled_1:m_axi_gmem-DDR[1]          RD = 16.000 KB              WR = 4.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 17 minute(s) 10 seconds, Emulation time: 0.309073 ms]
Data transfer between kernel(s) and global memory(s)
var014_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 16.000 KB              WR = 4.000 KB
var014_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 16.000 KB              WR = 4.000 KB
var014_pipelined_1:m_axi_gmem-DDR[1]          RD = 16.000 KB              WR = 4.000 KB
var014_unrolled_1:m_axi_gmem-DDR[1]          RD = 16.000 KB              WR = 4.000 KB

INFO::[ Vitis-EM 22 ] [Time elapsed: 18 minute(s) 43 seconds, Emulation time: 0.335645 ms]
Data transfer between kernel(s) and global memory(s)
var014_no_pragmas_1:m_axi_gmem-DDR[1]          RD = 16.000 KB              WR = 4.000 KB
var014_pipe_unroll_1:m_axi_gmem-DDR[1]         RD = 16.000 KB              WR = 4.000 KB
var014_pipelined_1:m_axi_gmem-DDR[1]          RD = 16.000 KB              WR = 4.000 KB
var014_unrolled_1:m_axi_gmem-DDR[1]          RD = 16.000 KB              WR = 4.000 KB
```

Рисунок 1.5 — Сравнительная таблица с временем выполнения

Ниже на рисунках 1.6 - 1.8 приведены результаты синтеза программы для ускорителя.

```
[Console output redirected to file:/iu_home/iu7044/workspace/lb2/hls_acc_lab/Hardware/SystemDebugg
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7044/workspace/lb2/hls_acc_lab_system/Hardware/binary_container_1.xclbin
Loading: '/iu_home/iu7044/workspace/lb2/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) |
|-------------------------+-------------------------|
| var014_no_pragmas       |               108710865 |
|-------------------------+-------------------------|
| var014_unrolled         |               106970670 |
|-------------------------+-------------------------|
| var014_pipelined        |               107143691 |
|-------------------------+-------------------------|
| var014_pipe_unroll      |               107024103 |
|-------------------------+-------------------------|
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 1.6 — Сравнительная таблица с временем выполнения

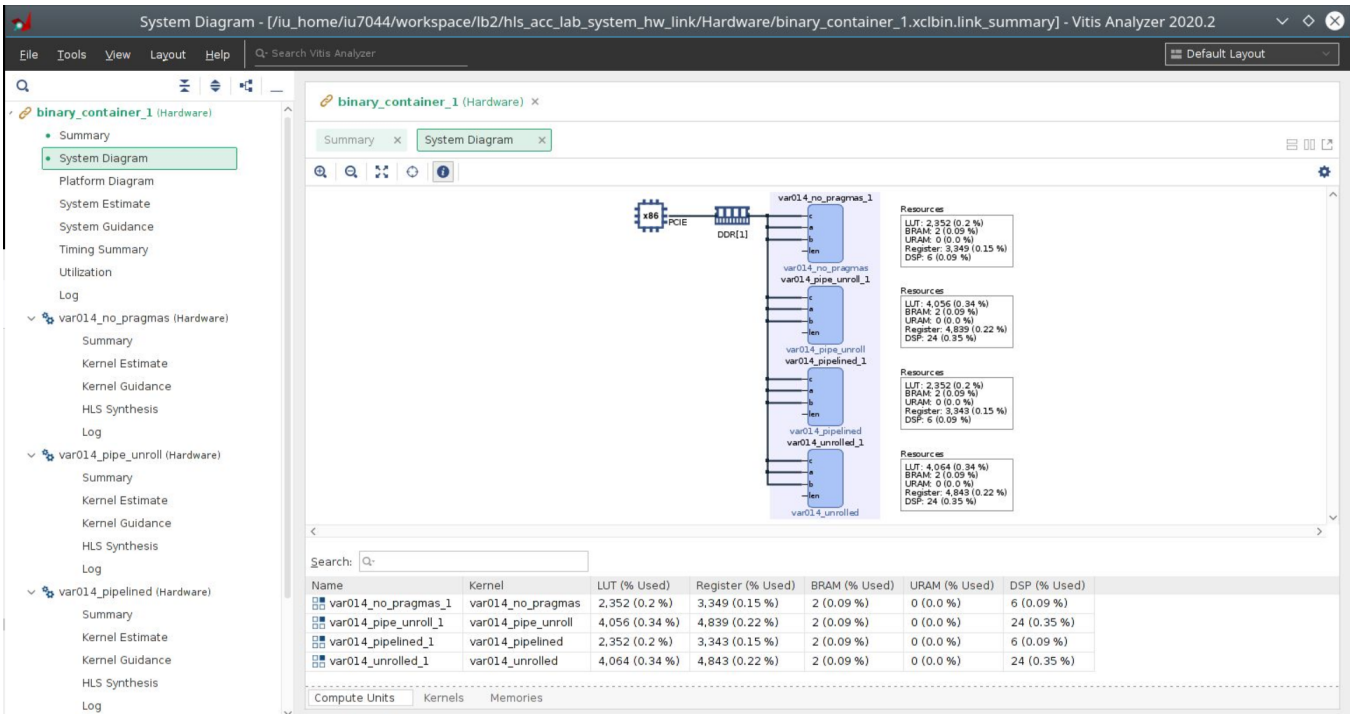


Рисунок 1.7 — Диаграмма системы

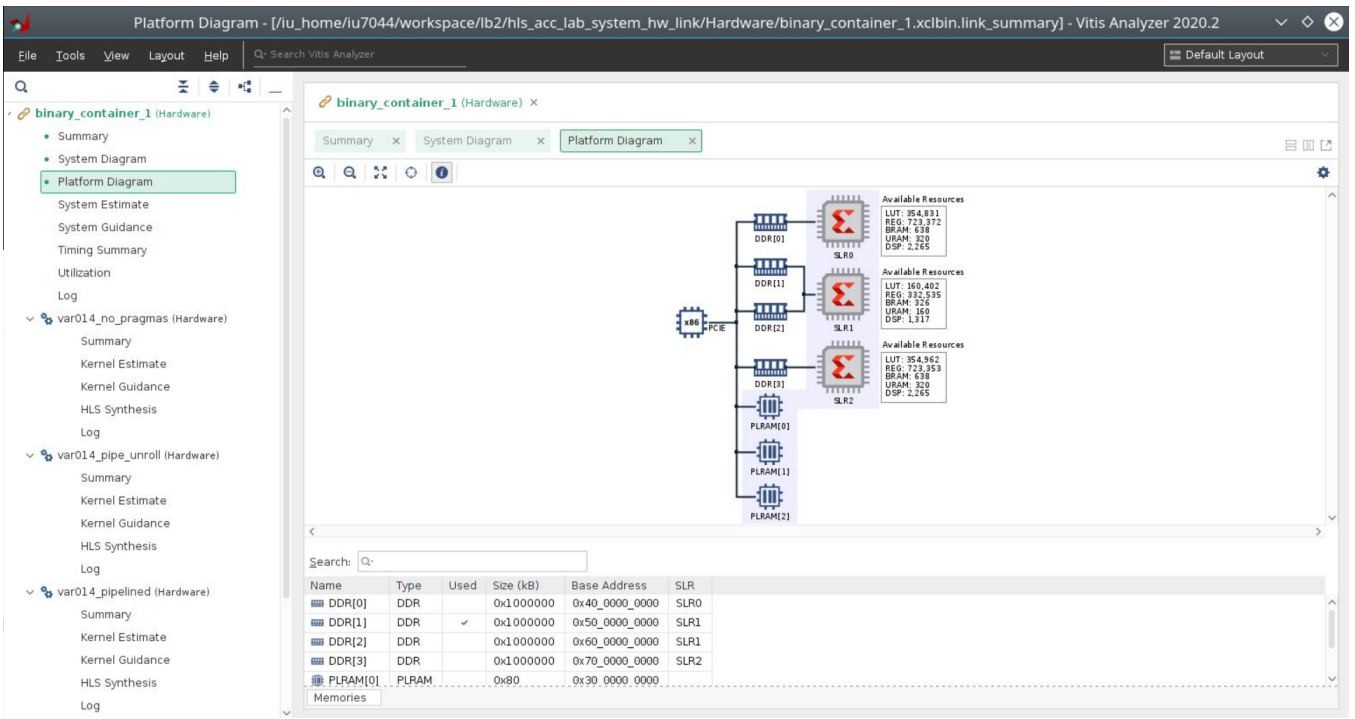
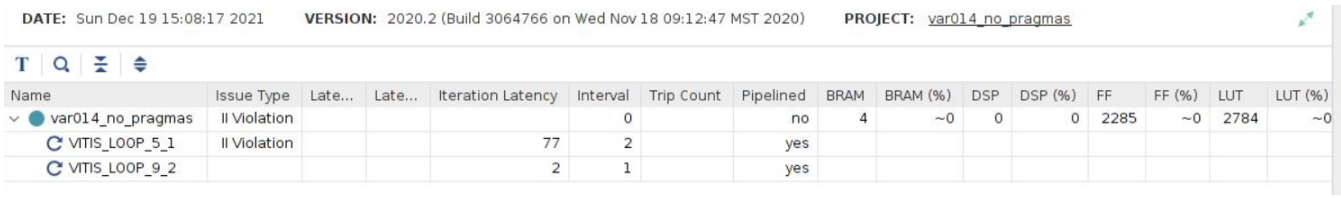
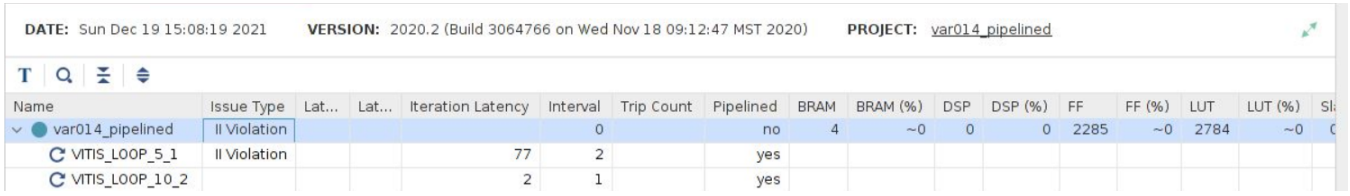


Рисунок 1.8 — Диаграмма платформы


DATE: Sun Dec 19 15:08:17 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: var014_no_pragmas

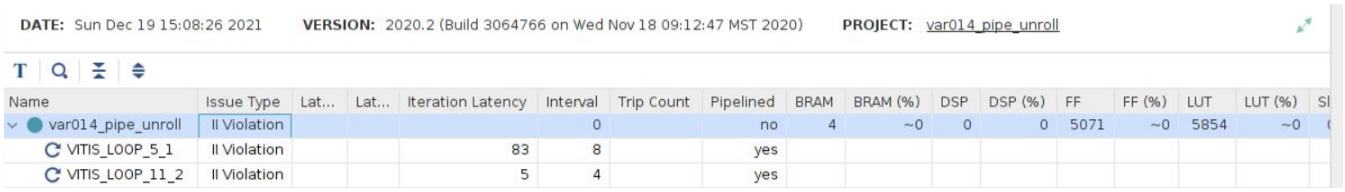
Name	Issue Type	Late...	Late...	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)
var014_no_pragmas	II Violation				0		no	4	~0	0	0	2285	~0	2784	~0
VITIS_LOOP_5_1	II Violation			77	2		yes								
VITIS_LOOP_9_2				2	1		yes								

Рисунок 1.9 — Информация о компонентах для реализации без оптимизации


DATE: Sun Dec 19 15:08:19 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: var014_pipelined

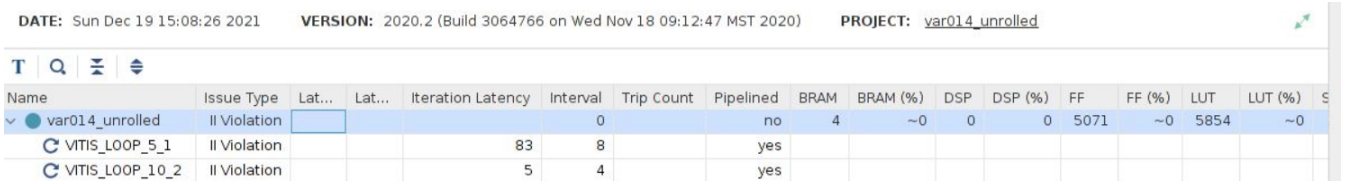
Name	Issue Type	Lat...	Lat...	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)
var014_pipelined	II Violation				0		no	4	~0	0	0	2285	~0	2784	~0
VITIS_LOOP_5_1	II Violation			77	2		yes								
VITIS_LOOP_10_2				2	1		yes								

Рисунок 1.10 — Информация о компонентах для реализации с конвейеризацией


DATE: Sun Dec 19 15:08:26 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: var014_pipe_unroll

Name	Issue Type	Lat...	Lat...	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)
var014_pipe_unroll	II Violation				0		no	4	~0	0	0	5071	~0	5854	~0
VITIS_LOOP_5_1	II Violation			83	8		yes								
VITIS_LOOP_11_2	II Violation			5	4		yes								

Рисунок 1.11 — Информация о компонентах для реализации с развертыванием циклов


DATE: Sun Dec 19 15:08:26 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: var014_unrolled

Name	Issue Type	Lat...	Lat...	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	BRAM (%)	DSP	DSP (%)	FF	FF (%)	LUT	LUT (%)
var014_unrolled	II Violation				0		no	4	~0	0	0	5071	~0	5854	~0
VITIS_LOOP_5_1	II Violation			83	8		yes								
VITIS_LOOP_10_2	II Violation			5	4		yes								

Рисунок 1.12 — Информация о компонентах для реализации с конвейеризацией и развертыванием циклов

Заключение

При конвейерной обработке цикл может начинать последующие итерации цикла менее чем за три такта. Конвейерная обработка цикла приводит к разворачиванию любых циклов, вложенных внутрь конвей ерного 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.

Разворачивание циклов является общепризнанным механизмом снижения времени выполнения 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 помогли уменьшить количество тактов на выполнение цикла. Комбинирование двух вариантов оптимизации также привело к оптимизации программы.

Ответы на контрольные вопросы

1. Преимущества и недостатки аппаратных ускорителей на ПЛИС по сравнению с CPU и графическими ускорителями?

В отличие от CPU и GPU, программируемые устройства 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 (Рисунок 1). В таком случает, 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.

2. Основные способы оптимизации циклических конструкций ЯВУ, реализуемых в виде аппаратных ускорителей?

1. Конвейеризация. Позволяет повысить пропускная способность, за счет увеличения времени синтеза программы и требований к ресурса ми 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 упро щение преобразований позволяет уменьшить число последовательных ячеек от триггера до триггера, повысив, таким образом, частоту. Для указания компилятору о необходимости конвейеризировать циклическую обработку используется директива PIPELINE;

2. Разворачивание циклов. Для указания компилятору о необходи мости развернуть цикл используется директива UNROLL: его итерации начинают выполняться параллельно на собственном наборе оборудования. Количество тактов на выполнение всего цикла уменьшается за счет роста размера схемы.

3. Назовите 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

   а. Устанавливаем необходимые для тестирования значения;

   б. Копируем содержимое буферов в DDR память ускорительной карты;

   в. Заупскаем задачу на исполнение и ждем готовности по преры ванию;

   г. Читаем метки времени исполнения задачи;

   д. Читаем данные из DDR памяти устройства в буфер результатов;

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

При отладке в режиме программной эмуляцим код ядра компили руется 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.

Для отладки в режиме аппаратной эмуляции код ядра компили руется в аппаратную модель (RTL), которая запускается в специальном симуляторе на ЦПУ. Этот вариант сборки и запуска занимает больше вре мени, но обеспечивает подробное и точное представление активности ядра. Данный вариант сборки полезен для тестирования функциональности ускорителя и получения начальных оценок производительности.

Для отладки в режиме аппаратного обеспечения (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?

- отладчик, имеющий графический интерфейс;

- использование конструкций, указывающих компилятору путь оптимизации (прагмы и директивы ($set\_directive\_*$));

- средство анализа Vivado IDE, позволяющее в том числе оценивать время и затраты после синтеза или размещения, выполнять симуляцию выполнения программы на ускорителе. Также Vivado IDE позволяет после высокоуровнего синтеза оптимизировать проекты на уровне межрегистро вых передач.