**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 (ИУ7)**

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 ИУ7-51Б _____ Р.Е. Костев
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ А.Ю. Попов
(Подпись, дата) (И.О. Фамилия)

Москва, 2021

# Введение

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 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-уровненного синтеза, выполнить его отладку.

# Ход работы

Реализовать 4 варианта ядра:

1. Неоптимизированный цикл
2. Конвейерная организация цикла
3. Частично развернутый цикл
4. Конвейерный и частично развернутый цикл

Исходный код функции ядра для моего варианта (14):

```
extern "C" {
      void var014_no_pragmas(int* c, const int* a, const int* b, const int len) {
          int tmpA = 0;
          int tmpB = 0;
          for (int i = 0; i < len; i++) {
                tmpA += a[i] * i;
                tmpB += b[i] * i;
          }
          for (int i = 0; i < len; i+=2) {
                c[i] = tmpA;
                c[i+1] = tmpB;
          }
      }
}
```

С конвейерной организацией:

```
extern "C" {
      void var014_pipelined(int* c, const int* a, const int* b, const int len) {
          int tmpA = 0;
          int tmpB = 0;
          for (int i = 0; i < len; i++) {
                #pragma HLS PIPELINE
                tmpA += a[i] * i;
                tmpB += b[i] * i;
          }
          for (int i = 0; i < len; i+=2) {
                #pragma HLS PIPELINE
                c[i] = tmpA;
                c[i+1] = tmpB;
          }
      }
}
```

С разворачиванием циклов:

```
extern "C" {
      void var014_unrolled(int* c, const int* a, const int* b, const int len) {
          int tmpA = 0;
          int tmpB = 0;
          for (int i = 0; i < len; i++) {
                #pragma HLS UNROLL factor=2
                tmpA += a[i] * i;
                tmpB += b[i] * i;
          }
          for (int i = 0; i < len; i+=2) {
                #pragma HLS UNROLL factor=2
                c[i] = tmpA;
```

```
                c[i+1] = tmpB;
            }
        }
}
```

С конвейерной организацией и разворачиванием циклов:

```
extern "C" {
      void var014_pipe_unroll(int* c, const int* a, const int* b, const int len) {
          int tmpA = 0;
          int tmpB = 0;
          for (int i = 0; i < len; i++) {
               #pragma HLS UNROLL factor=2
               #pragma HLS PIPELINE
                tmpA += a[i] * i;
                tmpB += b[i] * i;
          }
          for (int i = 0; i < len; i+=2) {
               #pragma HLS UNROLL factor=2
               #pragma HLS PIPELINE
                c[i] = tmpA;
                c[i+1] = tmpB;
          }
      }
}
```

# Emulation-SW

Ниже представлены результаты работы приложения в режиме Emultaion-SW:

```
[Console output redirected to
file:/iu_home/iu7134/workspace/KostevD_lab2/Emulation-
SW/SystemDebugger_KostevD_lab2_system_KostevD_lab2.launch.log]
Found Platform
Platform Name: Xilinx
INFO: Reading /iu_home/iu7134/workspace/KostevD_lab2_system/Emulation-
SW/binary_container_1.xclbin
Loading: '/iu_home/iu7134/workspace/KostevD_lab2_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 |                10218358 |
|-------------------------+-------------------------|
| var014_unrolled       |                    566109 |
|-------------------------+-------------------------|
| var014_pipelined     |                   754077 |
|-------------------------+-------------------------|
| var014_pipe_unroll |                   959448 |
|-------------------------+-------------------------|
Note: Wall Clock Time is meaningful for real hardware execution only, not for
emulation.\nPlease refer to profile summary for kernel execution time for hardware
emulation.\nTEST PASSED.
```

# Emulation-HW

В режиме сборки проекта Emulation-HW можно проанализировать код путем изучения отчетов о сборке ядер в Assistant View. На рисунке 1 приведена копия экрана Assistant View.

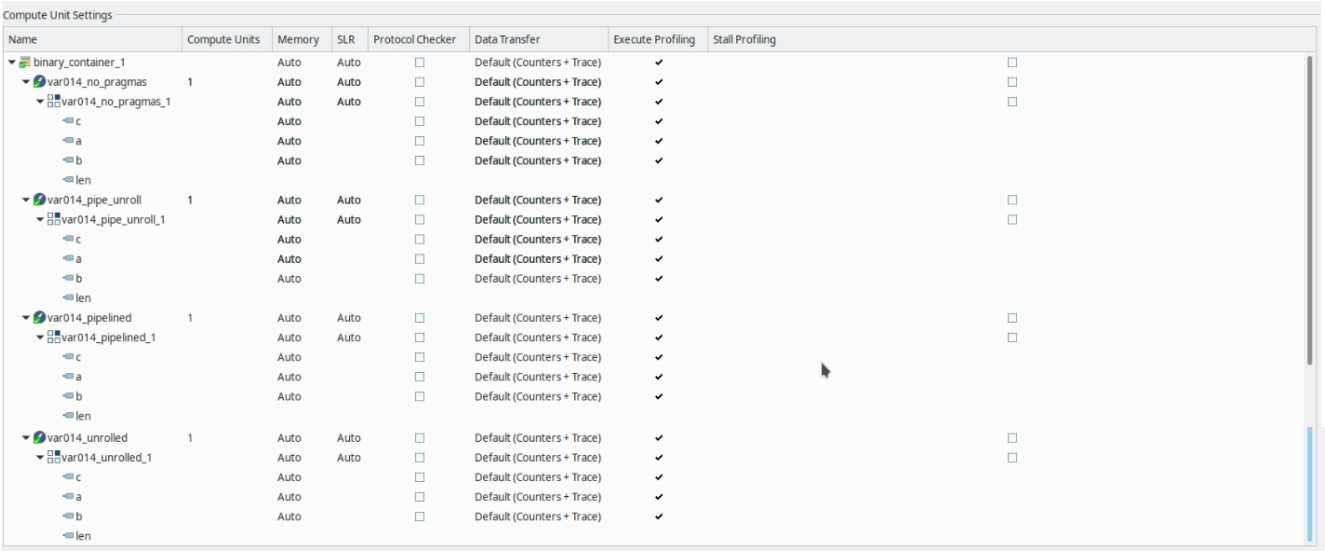



**Рисунок 1. Assistant View**

Результаты работы в режиме Emulation-HW представлены ниже:

```
|-------------------------+-------------------------|
| Kernel                  |    Wall-Clock Time (ns) |
|-------------------------+-------------------------|
| var014_no_pragmas       |              35013295665 |
|-------------------------+-------------------------|
| var014_unrolled         |              33015592670 |
|-------------------------+-------------------------|
| var014_pipelined        |              34021855413 |
|-------------------------+-------------------------|
| var014_pipe_unroll      |              36012919078 |
|-------------------------+-------------------------|
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
```

Для наблюдения транзакций на шинах и работы всех ускорительных ядек используется внутрисхемный отладчик IDE Vivado.

Диаграмма транзакций представлена на рисунке 2.

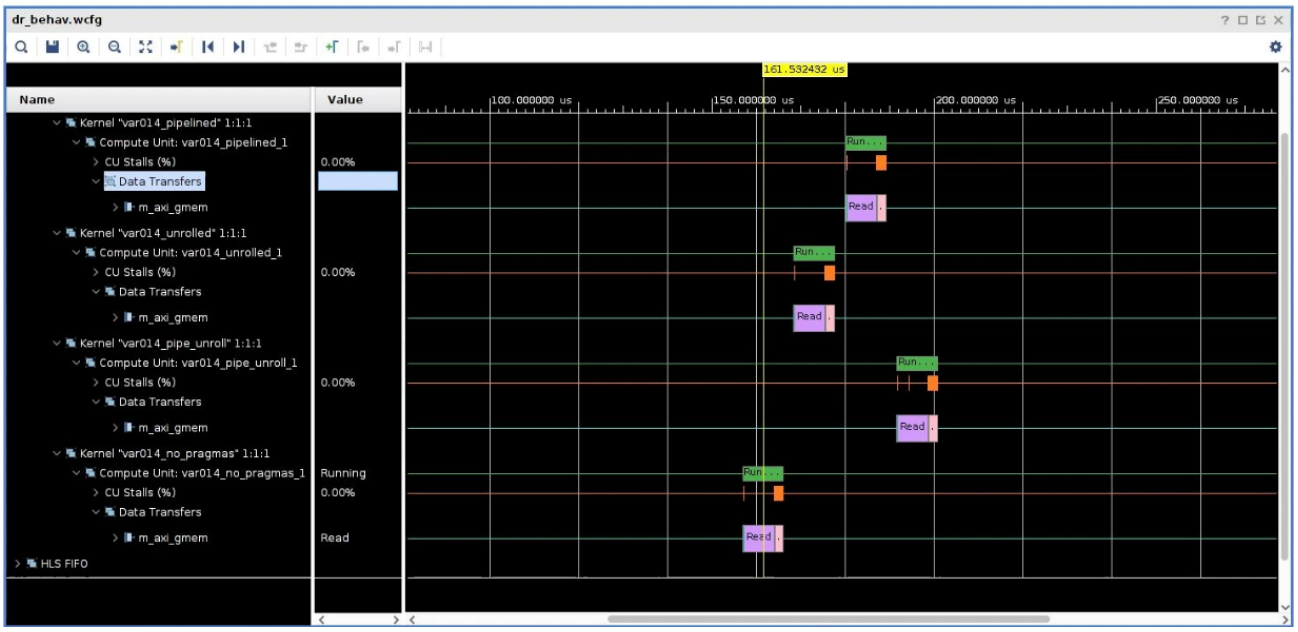



**Рисунок 2. Диаграмма транзакций**

# Hardware

Ниже представлены результаты работы в режиме HardWare

```
[Console output redirected to
file:/iu_home/iu7134/workspace/KostevD_lab2/Hardware/SystemDebugger_KostevD_lab2_s
ystem_KostevD_lab2.launch.log]
Found Platform
Platform Name: Xilinx
INFO: Reading
/iu_home/iu7134/workspace/KostevD_lab2_system_hw_link/Hardware/binary_container_1.
xclbin
Loading:
'/iu_home/iu7134/workspace/KostevD_lab2_system_hw_link/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       |                 5089634 |
|-------------------------+-------------------------|
| var014_unrolled         |                  396950 |
|-------------------------+-------------------------|
| var014_pipelined        |                 1691983 |
|-------------------------+-------------------------|
| var014_pipe_unroll      |                  597758 |
|-------------------------+-------------------------|
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.
```

На рисунках 3-7 представлены копии экранов вкладок “Summary”, “System Diagram”, “Platform Diagram”.

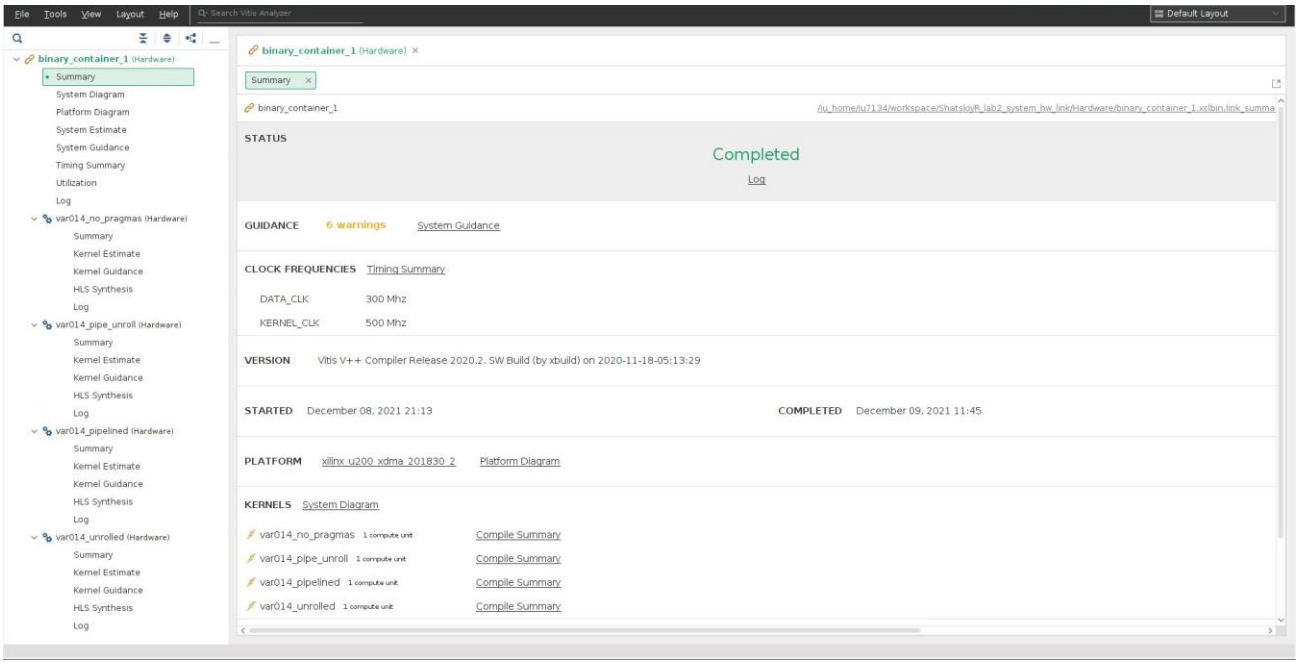



**Рисунок 3. Summary**

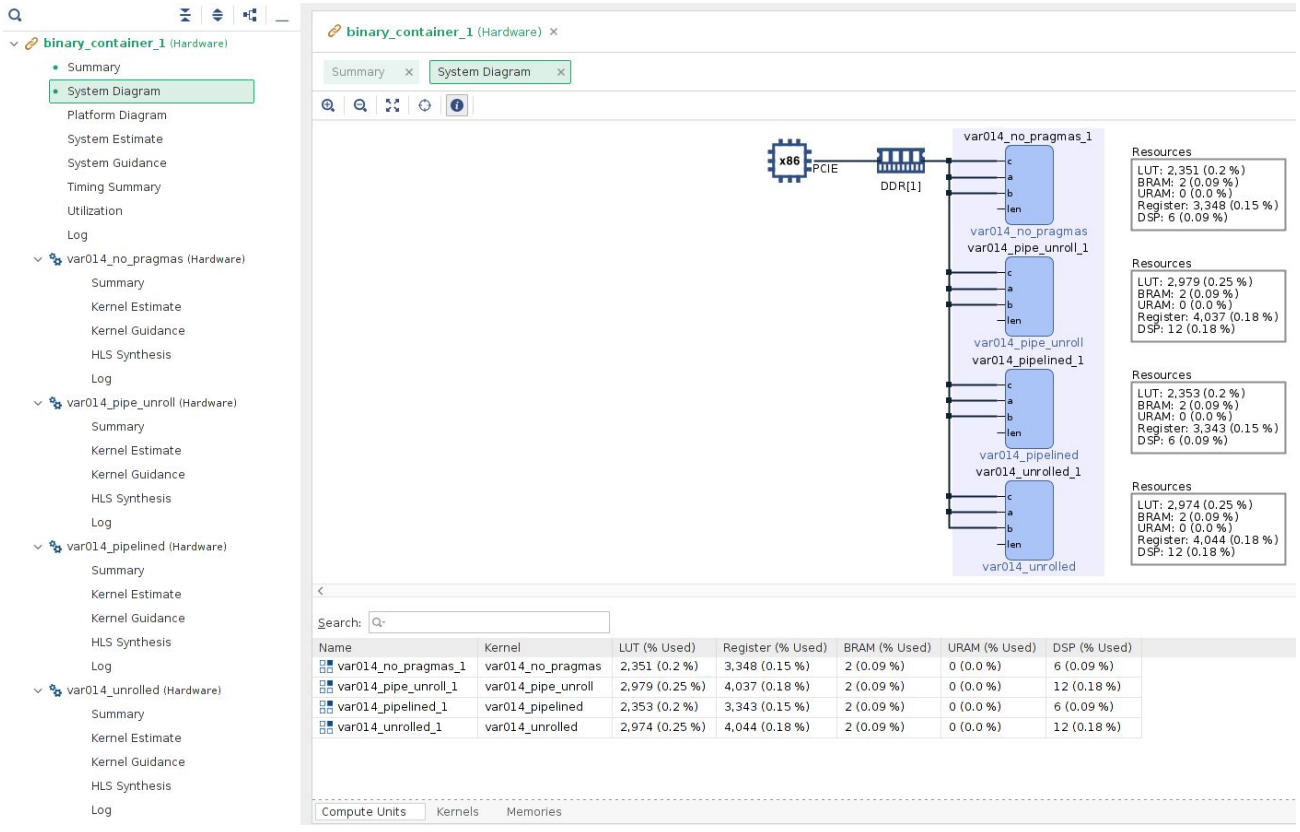



**Рисунок 4. System Diagram**

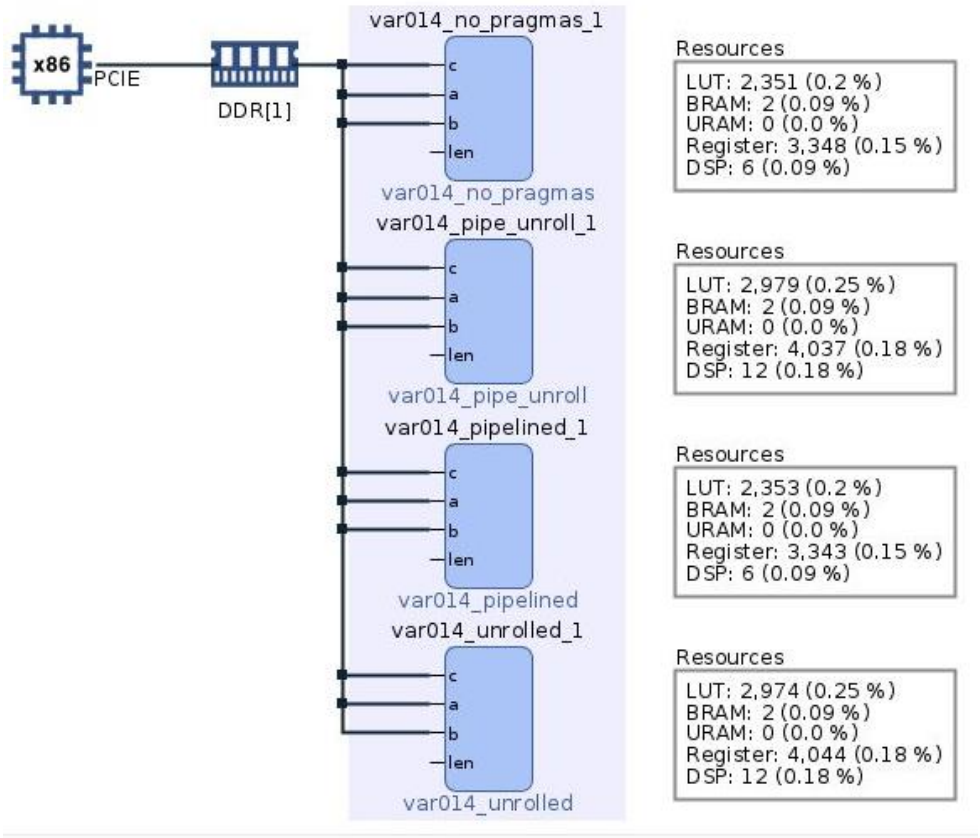



**Рисунок 5. System Diagram 2**

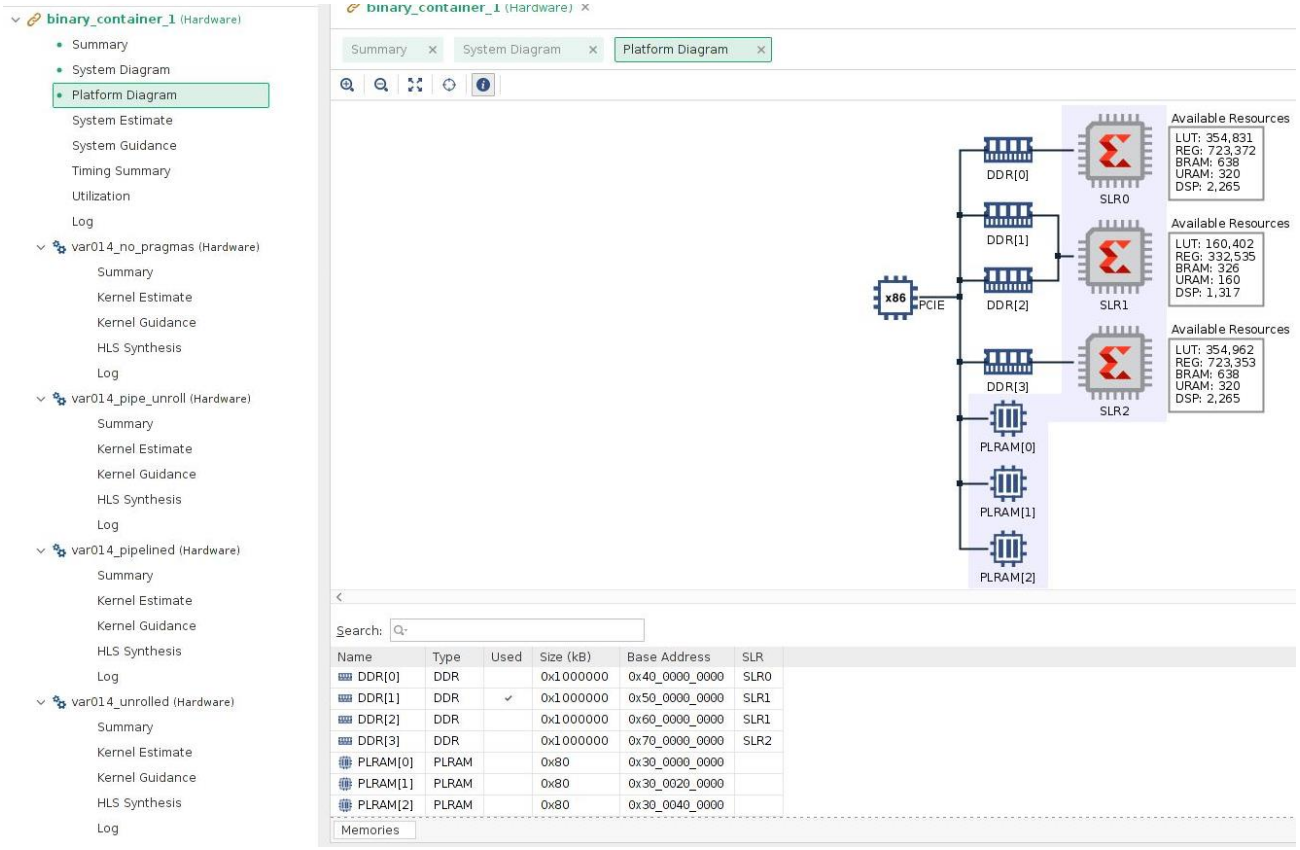



**Рисунок 6. Platform Diagram**

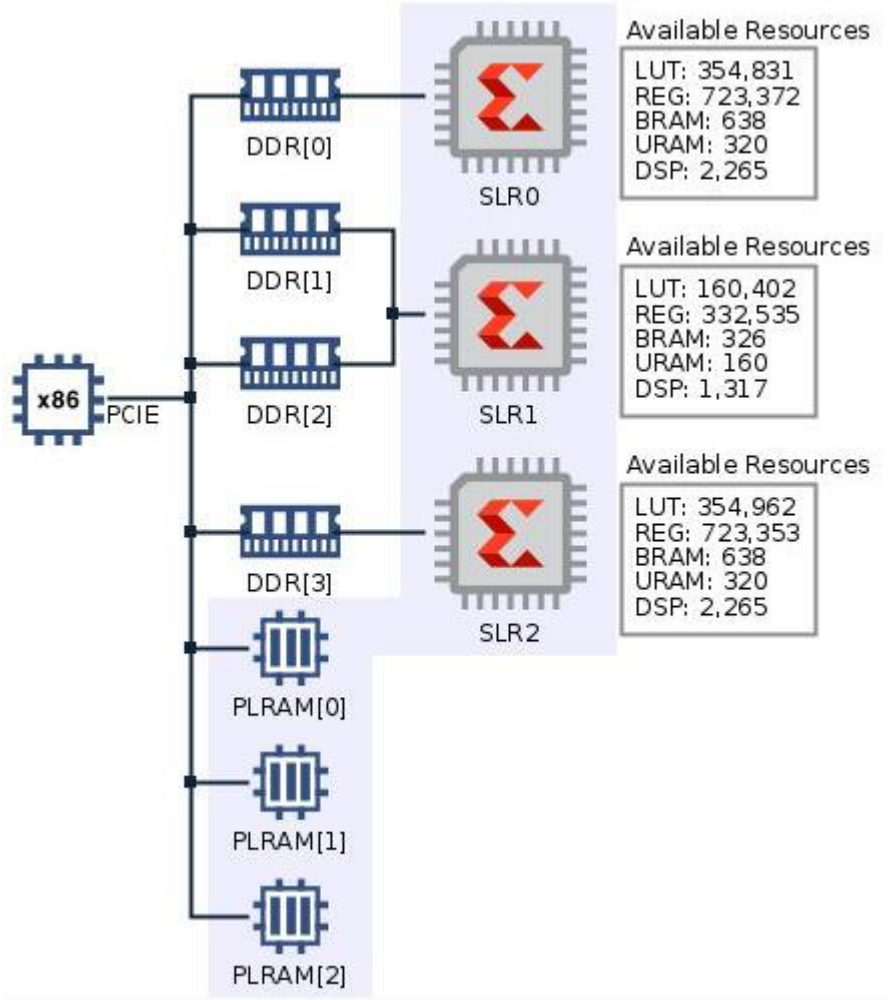
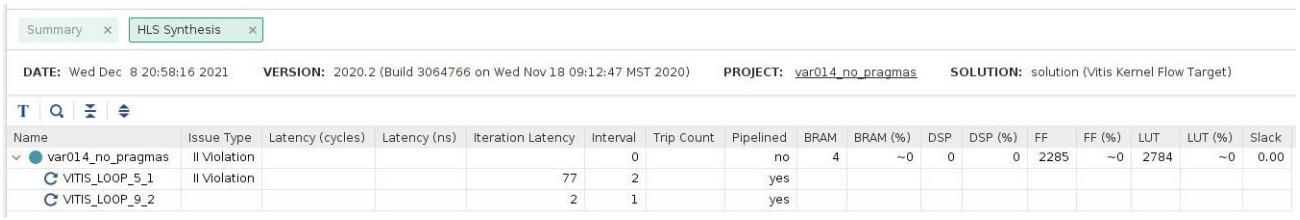



**Рисунок 7. Platfrom Diagram 2**

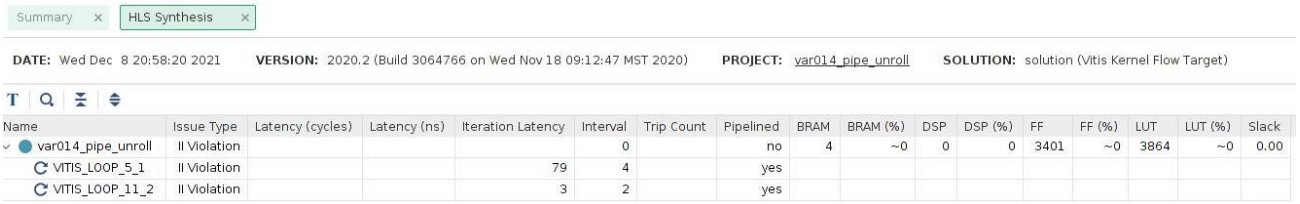
На рисунках 8-11 представлены копии экранов вкладок “HLS Synthesis” для  всех 4 ядер.


Summary × HLS Synthesis ×

DATE: Wed Dec 8 20:58:16 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: var014_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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| var014_no_pragmas | II Violation | | | | 0 | | no | 4 | ~0 | 0 | 0 | 2285 | ~0 | 2784 | ~0 | 0.00 |
| VITIS_LOOP_5_1 | II Violation | | | 77 | 2 | | yes | | | | | | | | | |
| VITIS_LOOP_9_2 | | | | 2 | 1 | | yes | | | | | | | | | |

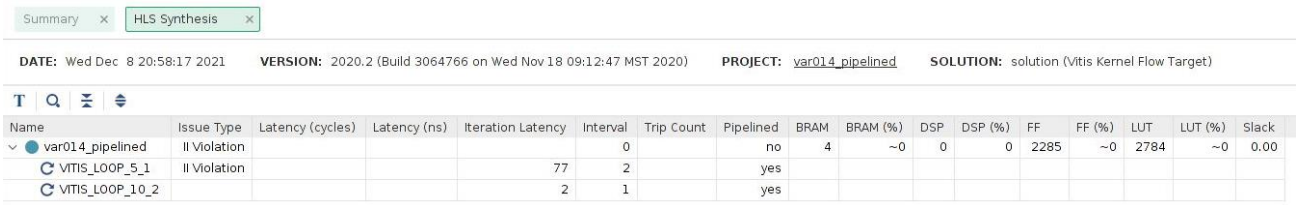
**Рисунок 8. HLS Synthesis – var014_no_pragmas**


Summary × HLS Synthesis ×

DATE: Wed Dec 8 20:58:20 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: var014_pipe_unroll 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| var014_pipe_unroll | II Violation | | | | 0 | | no | 4 | ~0 | 0 | 0 | 3401 | ~0 | 3864 | ~0 | 0.00 |
| VITIS_LOOP_5_1 | II Violation | | | 79 | 4 | | yes | | | | | | | | | |
| VITIS_LOOP_11_2 | II Violation | | | 3 | 2 | | yes | | | | | | | | | |

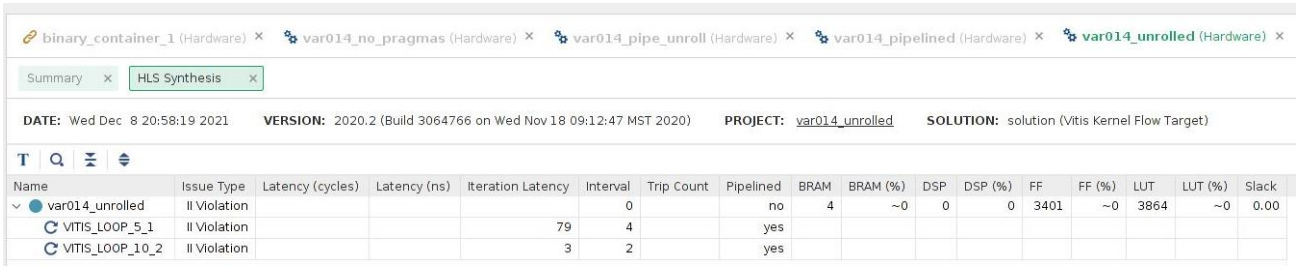
**Рисунок 9. HLS Synthesis – var014_pipe_unroll**


Summary × HLS Synthesis ×

DATE: Wed Dec 8 20:58:17 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: var014_pipelined 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| var014_pipelined | II Violation | | | | 0 | | no | 4 | ~0 | 0 | 0 | 2285 | ~0 | 2784 | ~0 | 0.00 |
| VITIS_LOOP_5_1 | II Violation | | | 77 | 2 | | yes | | | | | | | | | |
| VITIS_LOOP_10_2 | | | | 2 | 1 | | yes | | | | | | | | | |

**Рисунок 10. HLS Synthesis – var014_pipelined**


binary_container_1 (Hardware) × var014_no_pragmas (Hardware) × var014_pipe_unroll (Hardware) × var014_pipelined (Hardware) × var014_unrolled (Hardware) ×

Summary × HLS Synthesis ×

DATE: Wed Dec 8 20:58:19 2021 VERSION: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020) PROJECT: var014_unrolled 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| var014_unrolled | II Violation | | | | 0 | | no | 4 | ~0 | 0 | 0 | 3401 | ~0 | 3864 | ~0 | 0.00 |
| VITIS_LOOP_5_1 | II Violation | | | 79 | 4 | | yes | | | | | | | | | |
| VITIS_LOOP_10_2 | II Violation | | | 3 | 2 | | yes | | | | | | | | | |

**Рисунок 11. HLS Synthesis – var014_unrolled**

# Обоснование результатов тестов

В ходе лабораторной работы было проведено функциональное тестирование четырех ядер в 3 различных режимах.

Во всех режимах самым быстрым оказалось ядро, использующее разворачивание цикла, а вторым по эффективности оказался метод конвейеризации. Однако в режиме Hardware вторым по эффективности оказалось метод разворачиваения с конвейеризацией, который в остальных режимах был на 3 и 4 местах. Такой результат можно объяснить тем, что в разных режимах различаются способы симуляции тестируемых методов.

# Контрольные вопросы

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

   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. 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.

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

    - Конвейеризация циклов – организация конвейера выполнения команд при итерации цикла, занимающей больше 1 такта;
    - Разворачивание циклов - этот механизм может быть описан самим разработчиком вручную, если он просто повторит вычисления и сократит количество итераций цикла.
    - Потоковая обработка - позволяет сформировать конвейер из более крупных вычислительных блоков: нескольких функций или нескольких последовательных циклических конструкций.

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

    - Инициализация окружения
    - Создание буферов
    - Отображение буферов на устройство
    - Подготовках входных данных
    - Запуск ядер
    - Чтение данные из DDR памяти в буфер результата
    - Проверка результатов обработки данных
    - Освобождение памяти

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?

   В Vitis HLS можно использовать внутрисхемный отладчик IDE Vivado для наблюдения работы ускорительных ядер, Vivado Behavioural Simulation для выборов способа оптимизации, Assistant View для анализа результатов синтеза.

# Заключение

В ходе выполнения лабораторной работы я изучил методику и технологию синтеза аппаратных устройств ускорения вычислений по опсианиям на языках высокого уровня.