Санкт-Петербургский Политехнический Университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ Lab6\_Z1

Дисциплина: Проектирование реконфигурируемых гибридных вычислительных систем

Тема: Optimizing for Latency

Выполнил студент гр. Шеметов С.А.

гр.5140901/21502

Руководитель, доцент Антонов А.П.

«\_\_» \_\_\_\_\_\_ 2023

Санкт-Петербург

2023

Оглавление

[1. Задание 4](#_Toc150454955)

[2. Исходный код функции 5](#_Toc150454956)

[3. Исходный код теста 5](#_Toc150454957)

[3.1. Моделирование 5](#_Toc150454958)

[4. Создание скрипта 7](#_Toc150454959)

[5. Исследование решений 8](#_Toc150454960)

[5.1. Решение 1 8](#_Toc150454961)

[5.1.1. Синтез 8](#_Toc150454962)

[5.1.2. C/RTL моделирование 10](#_Toc150454963)

[5.2. Решение 2 10](#_Toc150454964)

[5.2.1. Синтез 10](#_Toc150454965)

[5.2.2. C/RTL моделирование 12](#_Toc150454966)

[5.3. Решение 3 13](#_Toc150454967)

[5.3.1. Синтез 13](#_Toc150454968)

[5.3.2. C/RTL моделирование 14](#_Toc150454969)

[5.4. Решение 3\_1 15](#_Toc150454970)

[5.4.1. Синтез 15](#_Toc150454971)

[5.4.2. C/RTL моделирование 17](#_Toc150454972)

[5.5. Решение 3\_2 18](#_Toc150454973)

[5.5.1. Синтез 18](#_Toc150454974)

[5.5.2. C/RTL моделирование 20](#_Toc150454975)

[5.6. Решение 3\_3 21](#_Toc150454976)

[5.6.1. Синтез 21](#_Toc150454977)

[5.6.2. C/RTL моделирование 23](#_Toc150454978)

[5.7. Решение 4 24](#_Toc150454979)

[5.7.1. Синтез 24](#_Toc150454980)

[5.7.2. C/RTL моделирование 26](#_Toc150454981)

[6. Сравнение результатов решений 26](#_Toc150454982)

[6.1. Сравнение временных параметров, производительности и аппаратных затрат 26](#_Toc150454983)

[6.2. Анализ результатов 27](#_Toc150454984)

[7. Измерение времени выполнения 28](#_Toc150454985)

[7.1. Измерение времени выполнения на ПК 28](#_Toc150454986)

[7.2. Измерение времени выполнения на аппаратной реализации 32](#_Toc150454987)

[7.3. Сравнение результатов 33](#_Toc150454988)

[8. Исследовательское задание 34](#_Toc150454989)

[9. Вывод 36](#_Toc150454990)

1. Задание

* Создать на языке С++ функцию (lab6\_z1.cpp), выполняющую умножение двух массивов поэлементно.
* Создать на языке С++ тест (lab6\_z1\_test.cpp) для проверки работы функции.
* Создать скрипт автоматизирующий процесс создания проекта и работы с ним.
* Создать, провести исследование и сравнительный анализ семи решений.
  + Микросхема: xa7a12tcsg325-1q;
  + clock period 10; clock\_uncertainty 1;
  + Solution1:
    - Выключить конвейеризацию цикла.
  + Solution2:
    - Установить unroll, factor=2;
    - Выключить конвейеризацию цикла;
    - Сравнить с Solution1.
  + Solution3:
    - Установить unroll, factor=4;
    - Выключить конвейеризацию цикла;
    - Сравнить с Solution1, 2.
  + Solution3\_1:
    - Выбрать настройки array\_partition к Solution3;
    - Сравнить с Solution1, 2, 3.
  + Solution3\_2:
    - Выбрать настройки array\_reshape к Solution3;
    - Сравнить с Solution3\_1.
  + Solution3\_3:
    - Выбрать настройки array\_reshape к Solution3;
    - Сравнить с Solution3\_2.
  + Solution4:
    - Установить unroll, factor=16;
    - Сделать такие установки, чтобы чтение всех необходимых данных осуществлялось за один такт, все 16 умножений выполнялись параллельно, запись данных выполнялась за один такт;
    - Включить конвейеризацию цикла.
  + Синтез.
  + С/RTL cosimulation.
* Измерить время выполнения на ПК и на аппаратной реализации для Solution4.
* Выполнить изменение unroll factor так, чтобы максимально увеличить производительность и сравнить с Solution4 и реализацией на ПК.

2. Исходный код функции

Исходный код синтезируемой функции приведен на рисунке 1. Код заголовочного файла указан на рисунке 2.

*Изображение выглядит как Мультимедийное программное обеспечение, программное обеспечение, текст, Графическое программное обеспечение

Автоматически созданное описание*

*Рис.1. Исходный код функции* *lab6\_z1*

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

*Рис.2. Исходный код заголовочного файла lab6\_z1*

Функция принимает три вектора типа short и размером N: два входных и один выходной. Затем в цикле перемножаются i-ые элементы входных массивов и результат записывается в выходной массив.

3. Исходный код теста

Исходный код теста для проверки функции lab6\_z1, исправленный в соответствии с заданием, приведен на рисунке 3. Тест обеспечивает: запуск функции 3 раза; формирование входных массивов; очистку выходного массива между запусками функции; проверку правильности вычисленного результата и формирование признака успешного/неуспешного выполнения для каждого запуска функции.

3.1. Моделирование

Результаты моделирования исходного кода синтезируемой функции приведены на рисунке 4. Результаты моделирования показывают, что тест успешно пройден — так как условия теста выполняются (функция *main* возвращает 0) и результат расчета совпадает с ожидаемыми значениями.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

*Рис.3. Исходный код теста*

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание*

*Рис.4. Лог результата выполнения моделирования*

Для проверки правильности работы теста в него было внесено изменение в ожидаемый результат - expted\_A[j] = in\_B[j] \* in\_C[j] + 1. Результат моделирования приведён на рисунке 5. Он показывает, что тест отрабатывает ошибку корректно.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

*Рис.5. Лог результата при ошибке в тесте*

4. Создание скрипта

Созданный скрипт для автоматизации работы с проектом представлен на рисунке 6.

Данный скрипт создаёт или открывает проект, предварительно очищая его, если он уже существует, командой *open\_project -reset lab6\_z1*. Затем добавляется файл с исходным кодом *add\_files ./source/lab6\_z1.cpp* и устанавливается функция верхнего уровня в иерархии *set\_top lab6\_z1*. Добавляется тестовый файл - *add\_files -tb ./source/lab6\_z1\_test.cpp*. Затем по очереди создаются 7 решений. Для каждого из них задаётся одинаковая микросхема, период 10 нс и неопределённость 1 нс.

В первом решении используется только директива отключения конвейеризации. Во втором и третьем решении используется директива UNROLL с разными значениями factor. Директива UNROLL позволяет полностью или частично развернуть цикл. Частичное развертывание цикла позволяет указать коэффициент factor=N, чтобы создать N копий тела цикла и соответственно сократить количество итераций цикла. Решения sol3\_1-sol3\_3 основываются на решении sol3 и в них применяются ARRAY\_PARTITION и ARRAY\_RESHAPE для доступа к большему количеству данных за один такт и повышения пропускной способности. В решении sol4 используются директивы UNROLL и ARRAY\_RESHAPE, а также включена конвейеризация.

Запуск скрипта выполняется в командной строке Vitis HLS 2021.2 Command Prompt с помощью команды *Vitis\_hls -f lab6\_z1.tcl*.

**Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание**

*Рис.6. Скрипт для работы с проектом*

5. Исследование решений

5.1. Решение 1

5.1.1. Синтез

Решение sol1 не имеет директив для оптимизации и в нём отключена конвейеризация для цикла Mult. Временные параметры, производительность и аппаратные затраты решения представлены на рисунке 7.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

*Рис.7. Результаты синтеза для sol1*

Estimated timing для решения составило 4.302 нс. Latency и II – 161 такт и 162 такта соответственно. Аппаратные затраты - 18 триггеров и 62 таблиц.

Результаты планирования для sol1 представлены на рисунке 8.

Изображение выглядит как снимок экрана, Мультимедийное программное обеспечение, программное обеспечение, текст

Автоматически созданное описание

*Рис.8. Результаты планирования для sol1*

Как можно видеть на рисунке 8, одна итерация длится 5 тактов. На первом такте происходит считывание элементов b[i] и c[i], затем на следующих трёх тактах – умножение и на последнем такте итерации цикла – запись в a[i].

5.1.2. C/RTL моделирование

Временная диаграмма для sol1 представлена на рисунке 9.

Изображение выглядит как снимок экрана, текст, дисплей, программное обеспечение

Автоматически созданное описание

*Рис.9. Временная диаграмма C/RTL для sol1*

На рисунке 9 видно, что из массивов b и c одновременно считывается по одному элементу, они перемножаются между собой и результат записывается в массив a про фронту we0 и ce0.

5.2. Решение 2

5.2.1. Синтез

В решении sol2 используется директива unroll с параметром factor=2. Данная директива позволяет создать 2 дубликата цикла и выполнять вычисление параллельно. Результаты сравнения двух решений приведены на рисунке 10.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.10. Сравнение решений sol1 и sol2*

Сравнение данных на рисунке 10 показывает, что Estimated timing у обоих решений одинаковое. Однако Latency и II различаются в два раза. Для первого значения - 161 и 162 такта, для второго – 81 и 82 такта. Это связано с тем, что unroll сокращает число итераций и позволяет обрабатывать данные параллельно.

Аппаратные затраты немного различаются по числу FF триггеров и LUT таблиц перекодировок. Первое решение требует - 18 триггеров и 62 таблицы, второе - 22 триггера и 57 таблиц.

Результаты планирования для sol2 представлены на рисунке 11.

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание

*Рис.11. Результаты планирования для sol2*

При сравнении результатов планирования для sol1 и sol2 (рисунки 8 и 11) видно, что число тактов на итерацию цикла осталось одинаковым, но количество операций на одну итерацию удвоилось. В первом решении по умолчанию была выбрана однопортовая память, во втором же – двухпортова и unroll смог успешно распараллелить данные на 2 части.

5.2.2. C/RTL моделирование

Временная диаграмма для sol2 представлена на рисунке 12.

Изображение выглядит как электроника, снимок экрана, текст, Электронная техника

Автоматически созданное описание

*Рис.12. Временная диаграмма C/RTL для sol2*

На рисунке 12 видно, что из массивов b и c одновременно считывается по два элемента (i и i+1), происходит попарное перемножение и запись в советующий элемент массива a.

5.3. Решение 3

5.3.1. Синтез

Как и во втором решении в третьем используется директива unroll, только с другим значением factor=4. Результаты сравнения трёх решений приведены на рисунке 13.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.13. Сравнение решений sol1, sol2 и sol3*

Сравнение данных на рисунке 13 показывает, что Estimated timing у всех решений одинаковое. Однако у третьего решения относительно второго значения Latency и II сократились почти вдвое – 49 и 50 тактов. При этом аппаратные затраты третьего решения наибольшие - 35 триггеров и 175 таблиц.

Результаты планирования для sol3 представлены на рисунке 14.

Изображение выглядит как текст, снимок экрана, Графическое программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

*Рис.14. Результаты планирования для sol3*

При сравнении результатов планирования для трёх решений (рисунки 8, 11 и 14) видно, что sol3 возникает проблема с параллельным чтением. Хоть параметр factor задан равным 4, но за один такт невозможно считать 4 пары значений b и c. Эта проблема возникает в связи с тем, что решение имеет по умолчанию двухпортовую память и может считывать только 2 пары значений, самостоятельно увеличить производительность входного потока оно не может.

5.3.2. C/RTL моделирование

Временная диаграмма для sol3 представлена на рисунке 15.

Изображение выглядит как снимок экрана, текст, дисплей, программное обеспечение

Автоматически созданное описание

*Рис.15. Временная диаграмма C/RTL для sol3*

На временной диаграмме видно, что сначала на одном такте считываются первые две пары значений b и c, на следующем такте вторые две пары. После задержки на одном такте записывается сначала результат перемножения первых двух пар, а на следующем – вторых.

5.4. Решение 3\_1

5.4.1. Синтез

В решении sol3\_1 необходимо решить проблему, возникшую в предыдущем решении. Для того, чтобы все четыре пары данных для b и c считывались и обрабатывались одновременно ко всем массивам программы была применена директива array\_partition с параметрами factor=4 и cyclic. Значение factor указывает что массив будет разбит на 4 части. Параметр cyclic описывает как части будут заполняться – в данном случае циклично, т. е. первое значение будет в первой части, второе во второй и т. д. Результаты сравнения данного решения с решениями sol1-3 приведены на рисунке 16.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.16. Сравнение решений sol1, sol2, sol3 и sol3\_1*

Сравнение данных на рисунке 16 показывает, что Estimated timing у всех решений одинаковое. При этом решение sol3\_1 имеет наименьшие Latency и II (41 такт и 42 такта), а наименьшие аппаратные затраты как по числу триггеров (15 триггеров), так и по числу LUT (52 таблицы).

Результаты планирования для sol3\_1 представлены на рисунке 17.

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

*Рис.17. Результаты планирования для sol3\_1*

При сравнении результатов планирования для всех решений (рисунки 8, 11, 14 и 17) видно, что аналогично решению sol1-2, одна итерация цикла занимает 5 тактов. При этом происходит большее число операций чтения и в отличии от sol3 они все выполняются параллельно на одном такте.

5.4.2. C/RTL моделирование

Временная диаграмма для sol3\_1 представлена на рисунке 18.

Изображение выглядит как текст, электроника, снимок экрана, Электронная техника

Автоматически созданное описание

*Рис.18. Временная диаграмма C/RTL для sol3\_1*

При сравнении временных диаграмм sol3 и sol3\_1 (рисунки 15 и 18) видно, что в sol3\_1 все операции чтения из массивов b и с происходят одновременно и при этом между получением новой порции данных нет «пустых» тактов.

Для решения sol3\_1 при одинаковом Schedule Viewer возможны две разные временные диаграммы. Это связано с применяемыми директивами. В данном случае использовалась array\_partition с faсtor=4 и однопортовой памятью. Другим решением является использование array\_partition с faсtor=2 и двухпортовой памятью.

5.5. Решение 3\_2

5.5.1. Синтез

Решение sol3\_2 аналогично sol3\_1 решает проблему sol3 только с использованием директивы array\_reshape. Директива применяется с параметрами factor=2 и cyclic. Значение factor указывает что массив будет разбит на 2 части. Параметр cyclic описывает как будут формироваться слова – в данном случае циклично, т. е. первый элемент пойдёт в первое слово, второй – во второе, третий – в первое, четвёртый – во второе. Длина нового слова будет factor\*длина элемента. Результаты сравнения данного решения с sol3\_1 приведены на рисунке 19.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.19. Сравнение решений sol3\_1 и sol3\_2*

Сравнение данных на рисунке 19 показывает, что Estimated timing, Latency и II у двух решений совпадают. Аппаратные затраты незначительно отличаются и у sol3\_2 они немного больше. Для sol3\_1 – 15 триггеров и 52 таблицы, для sol3\_2 - 19 триггеров и 56 таблиц.

Результаты планирования для sol3\_2 представлены на рисунке 20.

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

*Рис.20. Результаты планирования для sol3\_2*

При сравнении результатов планирования для двух решений (рисунки 17 и 20) видно, число тактов на итерацию цикла для обоих решений совпадает и равно 5 тактам. Количество операций чтения для sol3\_2 меньше в два раза, однако длина считанных слов в 2 раза больше.

5.5.2. C/RTL моделирование

Временная диаграмма для sol3\_2 представлена на рисунке 21.

Изображение выглядит как текст, снимок экрана, электроника, дисплей

Автоматически созданное описание

*Рис.21. Временная диаграмма C/RTL для sol3\_2*

При сравнении временных диаграмм sol3\_1 и sol3\_2 (рисунки 18 и 21) видно, что в sol3\_2 число операций чтения сократилось вдвое, но при этом длина слова стала в два раза больше и равна 32 битам.

5.6. Решение 3\_3

5.6.1. Синтез

Решение sol3\_3 аналогично sol3\_2 решает проблему sol3 с использованием директивы array\_reshape. Директива применяется с параметрами factor=4 и cyclic. Результаты сравнения данного решения с sol3\_2 приведены на рисунке 22.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.22. Сравнение решений sol3\_2 и sol3\_3*

Сравнение данных на рисунке 2 показывает, что Estimated timing, Latency и II у двух решений совпадают. Аппаратные затраты незначительно отличаются и у sol3\_3 они немного меньше. Для sol3\_2 - 19 триггеров и 56 таблиц, для sol3\_3 – 15 триггеров и 52 таблицы.

Результаты планирования для sol3\_3 представлены на рисунке 23.

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание

*Рис.23. Результаты планирования для sol3\_3*

При сравнении результатов планирования для двух решений (рисунки 20 и 23) видно, число тактов на итерацию цикла для обоих решений совпадает и равно 5 тактам. Количество операций чтения для sol3\_3 меньше в два раза, однако длина считанных слов в 2 раза больше.

5.6.2. C/RTL моделирование

Временная диаграмма для sol3\_3 представлена на рисунке 24.

Изображение выглядит как снимок экрана, текст, программное обеспечение, дисплей

Автоматически созданное описание

*Рис.24. Временная диаграмма C/RTL для sol3\_3*

При сравнении временных диаграмм sol3\_2 и sol3\_3 (рисунки 21 и 24) видно, что в sol3\_3 число операций чтения сократилось вдвое, но при этом длина слова стала в два раза больше и равна 64 битам.

5.7. Решение 4

5.7.1. Синтез

В четвёртом решении необходимо было установить директиву unroll с factor=16 при этом сделать так, чтобы чтение всех необходимых данных осуществлялось за один такт, все 16 умножений выполнялись параллельно, запись данных выполнялась за один такт. Для этого использовалась директива array\_reshape с factor=16 и cyclic. Это позволит сделать размер слова равным 16\*длина элемента и считывать 16 элементов за итерацию цикла. Также в данном решении была включена конвейеризация. Результаты решения sol4 приведены на рисунке 25.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

*Рис.25. Результаты синтеза для sol4*

Estimated timing для решения составило 4.417 нс. Latency и II – 7 тактов и 8 тактов соответственно. Аппаратные затраты - 80 триггеров и 86 таблиц.

Результаты планирования для sol4 представлены на рисунке 26.

Изображение выглядит как снимок экрана, текст, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

*Рис.26. Результаты планирования для sol7*

Как можно видеть на рисунке 26, одна итерация длится 5 тактов. На первом такте происходит считывание слов длиной 16\*длина элемента массива, затем параллельно выполняются 16 операций умножения и на последнем такте запись результата.

5.7.2. C/RTL моделирование

Временная диаграмма для sol4 представлена на рисунке 27.

Изображение выглядит как снимок экрана, текст, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

*Рис.27. Временная диаграмма C/RTL для sol4*

На временной диаграмме для sol4 видно, что за один такт считывается 16 значений из массивов b и c. Они объединены в слово длиной 256 бит. Один запуск функции выполняет 2 таких чтения, так как всего в массивах N=32 элементов.

6. Сравнение результатов решений

6.1. Сравнение временных параметров, производительности и аппаратных затрат

В данной работе были реализованы 7 решений с разными значениями unroll и директивами оптимизации пропускной способности. Результаты сравнения приведены на рисунке 28.

Изображение выглядит как текст, снимок экрана, диаграмма

Автоматически созданное описание

*Рис.28. Сравнение всех решений*

6.2. Анализ результатов

На основе результатов сравнения решений была составлена электронная таблица и построен график, в котором для всех решений отображены: Latency, II и использованные ресурсы. Результаты представлены на рисунках 29–30.

Изображение выглядит как текст, число, Шрифт, линия

Автоматически созданное описание

*Рис.29. Электронная таблица для всех решений*

Изображение выглядит как График, линия, диаграмма, снимок экрана

Автоматически созданное описание

*Рис.30. График для всех решений*

С точки зрения производительности оптимальным решением является sol4, так как оно имеет наименьшие значения Latency и затрачивает меньше времени. Это обусловлено использованием директив unroll, array\_reshape и конвейеризации.

С точки зрения аппаратных затрат решения sol3\_1 и sol3\_3 являются оптимальным, так как они имеет наименьшее число триггеров и таблиц перекодировок.

7. Измерение времени выполнения

7.1. Измерение времени выполнения на ПК

Измерение времени выполнения синтезируемой функции было выполнено на ПК со следующими характеристиками:

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип процессора** | **Базовая частота работы** | **Максимальная частота работы** | **ОЗУ** |
| 12th Gen Intel(R) Core(TM) i5-12450H | 2 ГГц | 4,4 ГГц | 16 Гб |

На базе теста lab6\_z1\_test.cpp был создан отдельный, модернизированный, тест lab6\_z1\_testSW.cpp, который содержит операторы измерения времени выполнения и имеет большее число запусков равное 32. Код теста представлен на рисунке 31.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

*Рис.31. Код теста* *lab6\_z1\_testSW.cpp*

Для компиляции теста был установлен компилятор GCC версии 13.2.0. После этого была выполнена сама компиляции и запуск теста для трёх значений N – 4096. 16384, 32768 (рисунок 32). Результаты запуска представлены на рисунках 33–35.

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

*Рис.32. Выполнение компиляции теста*

Изображение выглядит как текст, снимок экрана, шаблон, монохромный

Автоматически созданное описание

*Рис.33. Результаты теста на ПК для N=4096*

Изображение выглядит как текст, снимок экрана, черно-белый, монохромный

Автоматически созданное описание

*Рис.34. Результаты теста на ПК для N=16384*

Изображение выглядит как текст, снимок экрана, черно-белый, монохромный

Автоматически созданное описание

*Рис.35. Результаты теста на ПК для N=*32768

7.2. Измерение времени выполнения на аппаратной реализации

Для измерения времени необходимо использовать решение sol4.

Синтез решения sol4 был осуществлён для трёх значений N – 4096. 16384, 32768. Результаты запуска представлены на рисунках 36–38.

Время выполнения = II \* Estimated period для каждого случая:

* для 4096: 262 такта \* 5.407 нс = 1416.634 нс
* для 16384: 1030 тактов\* 5.407 нс = 5569.21 нс
* для 32768: 2054 такта \* 5.407 нс = 11105.978 нс

Изображение выглядит как снимок экрана, Мультимедийное программное обеспечение, программное обеспечение, текст

Автоматически созданное описание

*Рис.36. Результаты теста на аппаратной реализации для N=4096*

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

*Рис.37. Результаты теста на аппаратной реализации для N=16384*

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

*Рис.38. Результаты теста на аппаратной реализации для N=32768*

7.3. Сравнение результатов

На основе результатов, полученных при измерениях времени выполнения на ПК и аппаратной реализации, была составлена электронная таблица и построен график, в котором отражено соотношение размера массивов данных задачи (N) ко времени выполнения. Результаты представлены на рисунках 39–40.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

*Рис.39. Электронная таблица для ПК и аппаратной реализации*

Изображение выглядит как линия, График, снимок экрана, диаграмма

Автоматически созданное описание

*Рис.40. График для ПК и аппаратной реализации*

На графике можно видеть, что результаты аппаратной реализации растут медленнее, чем на ПК. При этом аппаратная реализация выигрывает с точки зрения производительности в 4-6 раз. Это связанно с тем, что в аппаратной реализации выполнение цикла было распараллелено.

8. Исследовательское задание

В данном задании требуется увеличить unroll factor так, чтобы максимально увеличить производительность, учитывая возможности выбранной микросхемы (доступно максимум 40 умножителей). При этом все остальные директивы должны быть выставлены так, чтобы все использованные умножители работали параллельно.

Для этого в скрипт lab6\_z1.tcl было добавлено решение sol4\_1(рисунок 41), в котором значение unroll factor=32. Также для этого решения была использована директива array\_reshape с параметрами factor=32 и cyclic для каждого массива. Таким образом, в цикле за одну итерацию считывается по 32 значения b и c, параллельно перемножаются в 32 дубликатах цикла и записываются в a.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

*Рис.41. Скрипт для создания решения sol4\_1*

Результаты сравнения sol4 и sol4\_1 при N=32 представлены на рисунках 42–43. По ним видно, что Estimated timing, Latency и аппаратные затраты значительно снизились для решения sol4\_1.

Изображение выглядит как текст, число, снимок экрана, Шрифт

Автоматически созданное описание

*Рис.42. Электронная таблица для sol4 и sol4\_1*

Изображение выглядит как снимок экрана, линия, График, диаграмма

Автоматически созданное описание

*Рис.43. График для sol4 и sol4\_1*

Затем было выполнено сравнение времени выполнения на ПК и на двух аппаратных реализаций sol4 и sol4\_1 при больших значениях N. Результаты представлены на рисунках 44–45.

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

*Рис.44. Электронная таблица для ПК и двух аппаратных реализаций*

Изображение выглядит как снимок экрана, линия, График, скат

Автоматически созданное описание

*Рис.45. График для ПК и двух аппаратных реализаций*

На графике можно видеть, что результаты sol4\_1 являются наилучшими и превосходят решение sol4 в два раза.

9. Вывод

В результате выполнения данной лабораторной работы были получены навыки работы с директивой unroll и распараллеливанием циклов. Для этого была создана программа на языке C++, выполняющая перемножение поэлементное умножение двух массивов размером N.

Для создания проекта и работы с ним был создан TCL скрипт, содержащий 7 решений. В решениях использовались разные значения factor для директивы unroll, а также директивы array\_partition и array\_reshape для увеличения числа обрабатываемых элементов массива за одну итерацию цикла. Оптимальным с точки зрения производительности оказалось sol4 за счёт обработки 16 значений за одну итерацию цикла.

Также было выполнено измерение времени выполнения программы на ПК и аппаратной реализации для sol4. Результаты показали, что аппаратная реализация значительно выигрывает по производительности у ПК за счёт распараллеливания цикла.

Кроме того, было создано решение sol4\_1, которое имело unroll factor=32 и максимальную производительность для выбранной микросхемы. При сравнении времени выполнения ПК и двух аппаратных реализаций, sol4\_1 показало лучшие результаты и обошло sol4 в два раза.