Санкт-Петербургский Политехнический Университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ Lab7\_Z2

Дисциплина: Проектирование реконфигурируемых гибридных вычислительных систем

Тема: Optimizing for Area and Logic

Выполнил студент гр. Шеметов С.А.

гр.5140901/21502

Руководитель, доцент Антонов А.П.

«\_\_» \_\_\_\_\_\_ 2023

Санкт-Петербург

2023

Оглавление

[1. Задание 3](#_Toc152281962)

[2. Исходный код функции 3](#_Toc152281963)

[3. Исходный код теста 4](#_Toc152281964)

[3.1. Моделирование 4](#_Toc152281965)

[4. Создание скрипта 6](#_Toc152281966)

[5. Исследование решений 7](#_Toc152281967)

[5.1. Решение 1 7](#_Toc152281968)

[5.1.1. Синтез 7](#_Toc152281969)

[5.2. Решение 2 8](#_Toc152281970)

[5.2.1. Синтез 8](#_Toc152281971)

[5.2.2. C/RTL моделирование 10](#_Toc152281972)

[6. Сравнение результатов решений 11](#_Toc152281973)

[6.1. Сравнение временных параметров, производительности и аппаратных затрат 11](#_Toc152281974)

[6.2. Анализ результатов 11](#_Toc152281975)

[7. Измерение времени выполнения 13](#_Toc152281976)

[7.1. Измерение времени выполнения на ПК 13](#_Toc152281977)

[7.2. Измерение времени выполнения на аппаратной реализации 15](#_Toc152281978)

[7.3. Сравнение результатов 15](#_Toc152281979)

[8. Вывод 16](#_Toc152281980)

1. Задание

* Создать на языке С++ функцию (lab7\_z2.cpp), выполняющую умножение двух массивов поэлементно. Тип данных double.
* Создать на языке С++ тест (lab7\_z2\_test.cpp) для проверки работы функции.
* Создать скрипт автоматизирующий процесс создания проекта и работы с ним.
* Создать, провести исследование и сравнительный анализ двух решений.
  + Микросхема: xc7a100tcsg324-2;
  + clock period 10; clock\_uncertainty 1;
  + Solution1:
    - Выключить конвейеризацию цикла;
    - Для переменной temp\_mult задать директиву BIND\_OP: op=dmul, impl=dsp.
  + Solution2:
    - Задать максимально возможный Unroll Factor цикла MULT (чтобы использовались только DSP);
    - Использовать Array Partition (или Array Reshape) нужного типа и фактора, обеспечивающего балансировку производительности умножителей и чтения/записи данных (использование одно портовой или двух портовой памяти на выбор);
    - Включить конвейеризацию цикла;
    - Для переменной temp\_mult задать директиву BIND\_OP: op=mul, impl=dsp, latency=1.
  + Синтез.
  + С/RTL cosimulation.
* Измерить время выполнения на ПК (однопоточная и многопоточная реализация) и на аппаратной реализации для Solution2.

2. Исходный код функции

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

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

Автоматически созданное описание*

*Рис.1. Исходный код функции* *lab7\_z2*

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

Автоматически созданное описание

*Рис.2. Исходный код заголовочного файла lab7\_z2*

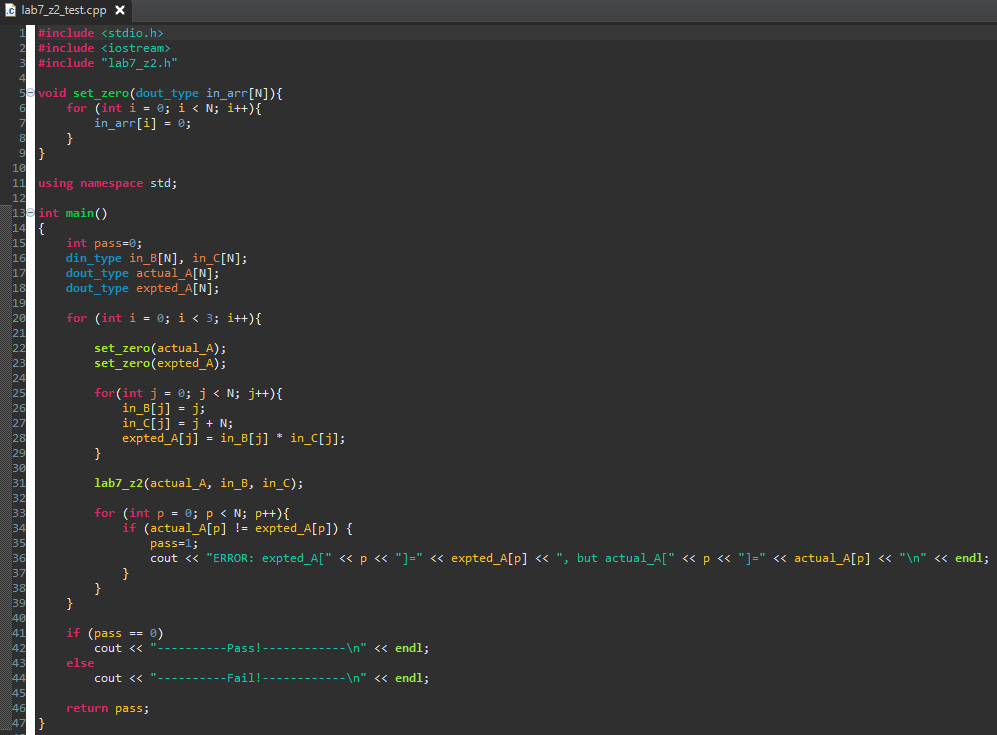
Функция принимает три вектора типа double и размера N=128: два входных и один выходной. Затем в цикле перемножаются элементы входных массивов и результат записывается в выходной массив.

3. Исходный код теста

Исходный код теста для проверки функции lab7\_z2, 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.



*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, Шрифт, программное обеспечение

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 проект, добавляет в него файл с исходным кодом и тестовый файл, а также устанавливает функцию верхнего уровня. Затем по очереди создаются 2 решений. Для каждого из них задаётся одинаковая микросхема, период 10 нс и неопределённость 1 нс.

В первом решении используются директивы отключения конвейеризации и BIND\_OP с параметрами dmul (умножение double чисел с плавающей запятой), dsp (связывание операции с DSP умножителями).

В решении sol2 используются директивы UNROLL и ARRAY\_PARTITION для обеспечения балансировки производительности умножителей и чтения/записи данных, а также используется директива BIND\_OP с параметрами mul (умножение целых чисел) и dsp (связывание операции с DSP умножителями) и включена конвейеризация.

Запуск скрипта выполняется в командной строке Vitis HLS 2021.2 Command Prompt с помощью команды *Vitis\_hls -f lab7\_z2.tcl*.

**Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание**

*Рис.6. Скрипт для работы с проектом*

5. Исследование решений

5.1. Решение 1

5.1.1. Синтез

В решении sol1 используется директива bind\_op с параметрами op=dmul, impl=dsp для переменной temp\_mult, которой будет присваиваться результат выполнения операции. Эта директива позволяет связать запланированную операцию умножения чисел double с ПЗ с имеющимися модулями умножения DSP. Временные параметры, производительность и аппаратные затраты решения представлены на рисунке 7.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, линия

Автоматически созданное описание

*Рис.7. Результаты синтеза для sol1*

Estimated timing для решения составило 8.567 нс. Для реализации потребовалось 11 модулей умножения DSP.

Всего в представленной микросхеме 240 умножителей. Максимальный unroll factor для цикла Mult, который можно использовать, чтобы использовались только DSP, равен 16. Так как при данном значении требуется число DSP, не превышающее максимальное значение микросхемы. При большем значении factor оно превысит число умножителей микросхемы.

Результаты планирования для sol1 представлены на рисунке 8.

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

Автоматически созданное описание

*Рис.8. Результаты планирования для sol1*

Как можно видеть на рисунке 8, в решении sol1 итерация цикла длится 8 тактов, операция умножения занимает 4 такта.

5.2. Решение 2

5.2.1. Синтез

В решении sol2 используется директива unroll с factor=16 для того, чтобы использовались только DSP. Также вместе с unroll используется директива array\_partition factor=16 и cyclic. Для такого решения по умолчанию будет применяться однопортовая память. Директива bind\_op используется с параметрами op=mul, impl=dsp, latency=1 для переменной temp\_mult, которой будет присваиваться результат выполнения операции. Включена конвейеризация. Временные параметры, производительность и аппаратные затраты решения представлены на рисунке 9.

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

Автоматически созданное описание

*Рис.9. Результаты синтеза для sol2*

Estimated timing для решения составило 8.567 нс. Для реализации потребовалось 176 модулей умножения DSP.

Результаты планирования для sol2 представлены на рисунках 10–11.

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

Автоматически созданное описание

*Рис.10. Результаты планирования для sol2*

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

Автоматически созданное описание*

*Рис.11. Результаты планирования для sol2*

Как можно видеть на рисунках 10–11, как и в sol1 итерация цикла занимает 8 тактов. Операция умножения занимает 4 такта. Параллельно считываются 16 значения b и 16 значения c. Также 16 операции умножения выполняются параллельно и их результат записывается в 16 элементов массива a. В данном цикле была выполнена конвейеризация, что видно на рисунке 10. II составляет 16 тактов.

5.2.2. C/RTL моделирование

Временная диаграмма для sol2 представлена на рисунке 12.

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

Автоматически созданное описание

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

На временной диаграмме видно, что каждый из массивов b и c был разбит на 16 блоков по 8 элементов. На каждом такте из блока считывается по одному элементу и выполняется их перемножение.

6. Сравнение результатов решений

6.1. Сравнение временных параметров, производительности и аппаратных затрат

В данной работе были реализованы 2 решения с разными параметрами bind\_op. Результаты сравнения приведены на рисунке 13.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.13. 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 14–15.

Изображение выглядит как текст, число, снимок экрана, Шрифт

Автоматически созданное описание

*Рис.14. Электронная таблица для всех решений*

Изображение выглядит как снимок экрана, линия, диаграмма, текст

Автоматически созданное описание

*Рис.15. График для всех решений*

С точки зрения производительности оптимальным решением является sol2, так как оно имеет наименьшие значения Latency и затрачивает меньше времени. Это обусловлено использованием директив bind\_op, unroll, array\_partition и конвейеризации.

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 lab7\_z2\_test.cpp был создан отдельный, модернизированный, тест lab7\_z2\_testSW.cpp, который содержит операторы измерения времени выполнения и имеет большее число запусков равное 32. Код теста представлен на рисунке 16.

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

Автоматически созданное описание

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

Автоматически созданное описание

*Рис.16. Код теста* *lab7\_z2\_testSW.cpp*

Также в исходный код функции для реализации многопоточности была добавлена pragma omp со значением 8, что соответствует числу ядер на ПК (рисунок 17).

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

Автоматически созданное описание

*Рис.17. Исходный код многопоточной функции lab7\_z2*

Для компиляции теста был установлен компилятор GCC версии 13.2.0 (рисунок 18). После этого была выполнена компиляции была выполнена на одном ядре и на 8. Запуск теста проводился для четырёх значений N – 8192, 16384, 32768, 65536.

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

*Рис.18. Выполнение компиляции теста*

Результаты запуска для одного потока.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время. нс** | **Медиана, нс** |
| 8192 | 18111.1 | 29000 | 21612.9 |
| 16384 | 24781.2 | 57000 | 29687.5 |
| 32768 | 38000 | 59705.9 | 54413.8 |
| 65536 | 84000 | 94600 | 89409.1 |

Результаты запуска для 8 потоков.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время, нс** | **Медиана, нс** |
| 8192 | 14500 | 39333.3 | 16500 |
| 16384 | 22812.5 | 43000 | 25437.5 |
| 32768 | 42093.8 | 60000 | 45647.1 |
| 65536 | 77250 | 135000 | 78846.2 |

7.2. Измерение времени выполнения на аппаратной реализации

Для измерения времени необходимо использовать решение sol2.

Синтез решения sol2 был осуществлён для четырёх значений N – 8192, 16384, 32768, 65536. Время выполнения = II \* Estimated period для каждого случая:

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **II, такт** | **Estimated period, нс** | **Время выполнения, нс** |
| 8192 | 521 | 8.567 | 4463.407 |
| 16384 | 1033 | 8.567 | 8849.711 |
| 32768 | 2057 | 8.567 | 17622.319 |
| 65536 | 4105 | 8.567 | 35167.535 |

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 19–20.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

*Рис.19. Электронная таблица для ПК (1 и 8 ядер) и аппаратной реализации*

Изображение выглядит как снимок экрана, График, линия, диаграмма

Автоматически созданное описание

*Рис.20. График для ПК (1 и 8 ядер) и аппаратной реализации*

На графике можно видеть, что результаты аппаратной реализации растут медленнее и значительно выигрывают по производительности у обеих реализаций на ПК. Это связанно с тем, что в аппаратной реализации выполнение цикла было распараллелено, а также использована конвейеризация и директива bind\_op. При этом многопоточная реализация незначительно выигрывает в производительности у однопоточной.

8. Вывод

В результате выполнения данной лабораторной работы были получены навыки работы с директивой bind\_op, которая позволяет связать запланированную операцию с имеющимися аппаратными ресурсами. Для этого была создана программа на языке C++, выполняющая поэлементное умножение двух массивов размером N и типом double.

Для создания проекта и работы с ним был создан TCL скрипт, содержащий 2 решения. В первом решении отключена конвейеризация и использована директива bind\_op, которая связывает операцию умножение чисел double с ПЗ с умножителями DSP. Во втором решении bind\_op связывает целочисленное умножение с умножителями DSP. Также в sol2 были использованы директивы unroll и array\_partition для распараллеливания цикла и конвейеризация. Оптимальным с точки зрения производительности оказалось sol2.

Также было выполнено измерение времени выполнения программы на для однопоточной и многопоточной реализации на ПК и аппаратной реализации для sol2. Результаты показали, что аппаратная реализация значительно выигрывает по производительности у ПК за счёт оптимизации. При этом многопоточной реализация на ПК незначительно выигрывает в производительности у однопоточной.