Санкт-Петербургский Политехнический Университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ Lab7\_Z1

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](#_Toc152097748)

[2. Исходный код функции 4](#_Toc152097749)

[3. Исходный код теста 4](#_Toc152097750)

[3.1. Моделирование 4](#_Toc152097751)

[4. Создание скрипта 6](#_Toc152097752)

[5. Исследование решений 7](#_Toc152097753)

[5.1. Решение 1 7](#_Toc152097754)

[5.1.1. Синтез 7](#_Toc152097755)

[5.2. Решение 1\_1 8](#_Toc152097756)

[5.2.1. Синтез 8](#_Toc152097757)

[5.3. Решение 1\_2 9](#_Toc152097758)

[5.3.1. Синтез 9](#_Toc152097759)

[5.4. Решение 2 11](#_Toc152097760)

[5.4.1. Синтез 11](#_Toc152097761)

[5.4.2. C/RTL моделирование 13](#_Toc152097762)

[6. Сравнение результатов решений 14](#_Toc152097763)

[6.1. Сравнение временных параметров, производительности и аппаратных затрат 14](#_Toc152097764)

[6.2. Анализ результатов 15](#_Toc152097765)

[7. Измерение времени выполнения 16](#_Toc152097766)

[7.1. Измерение времени выполнения на ПК 16](#_Toc152097767)

[7.2. Измерение времени выполнения на аппаратной реализации 18](#_Toc152097768)

[7.3. Сравнение результатов 18](#_Toc152097769)

[8. Вывод 19](#_Toc152097770)

1. Задание

* Создать на языке С++ функцию (lab7\_z1.cpp), выполняющую умножение двух массивов поэлементно. Тип данных int.
* Создать на языке С++ тест (lab7\_z1\_test.cpp) для проверки работы функции.
* Создать скрипт автоматизирующий процесс создания проекта и работы с ним.
* Создать, провести исследование и сравнительный анализ четырёх решений.
  + Микросхема: xc7a100tcsg324-2;
  + Solution1:
    - clock period 10; clock\_uncertainty 1;
    - Выключить конвейеризацию цикла.
  + Solution1\_1:
    - clock period 10; clock\_uncertainty 1;
    - Выключить конвейеризацию цикла;
    - Для переменной temp\_mult задать директиву BIND\_OP: op=mul, impl=dsp, latency=2.
  + Solution1\_2:
    - clock period 15; clock\_uncertainty 1;
    - Выключить конвейеризацию цикла;
    - Для переменной temp\_mult задать директиву BIND\_OP: op=mul, impl=fabric, latency=2.
  + Solution2:
    - clock period 10; clock\_uncertainty 1;
    - 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, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание*

*Рис.1. Исходный код функции* *lab7\_z1*

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

Автоматически созданное описание

*Рис.2. Исходный код заголовочного файла lab7\_z1*

Функция принимает три вектора типа int 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\_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, дисплей

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 проект, добавляет в него файл с исходным кодом и тестовый файл, а также устанавливает функцию верхнего уровня. Затем по очереди создаются 4 решений. Для каждого из них задаётся одинаковая микросхема и неопределённость 1 нс. Период для решения sol1\_2 указывается 15 нс, для остальных 10 нс.

В первом решении используется только директива отключения конвейеризации. В решениях sol1\_1 и sol1\_2 используется директива BIND\_OP с разными параметрами. BIND\_OP указывает, что для конкретной переменной операция (mul, add, div) должна быть сопоставлена ​​с конкретным ресурсом устройства для реализации (impl) в RTL. Опция latency позволяет задать задержку операции. В решении sol2 используются директивы UNROLL и ARRAY\_PARTITION для обеспечения балансировки производительности умножителей и чтения/записи данных, а также используется директива BIND\_OP и включена конвейеризация.

Запуск скрипта выполняется в командной строке Vitis HLS 2021.2 Command Prompt с помощью команды *Vitis\_hls -f lab7\_z1.tcl*.

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

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 снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

*Рис.7. Результаты синтеза для sol1*

Estimated timing для решения составило 6.880 нс. Для реализации потребовалось 3 модуля умножения DSP.

Результаты планирования для sol1 представлены на рисунке 8.

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

Автоматически созданное описание

*Рис.8. Результаты планирования для sol1*

Как можно видеть на рисунке 8, одна итерация длится 4 такта. На первых двух тактах происходит считывание элементов b[i] и c[i], затем на следующем – умножение и на последнем такте – запись в a[i]. Таким образом, сама операция умножения занимает 1 такт.

5.2. Решение 1\_1

5.2.1. Синтез

В решении sol1\_1 используется директива bind\_op с параметрами op=mul, impl=dsp, latency=2 для переменной temp\_mult, которой будет присваиваться результат выполнения операции. Эта директива позволяет связать запланированную операцию умножения с имеющимися модулями умножения DSP. Временные параметры, производительность и аппаратные затраты решения представлены на рисунке 9.

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

Автоматически созданное описание

*Рис.9. Результаты синтеза для sol1\_1*

Estimated timing для решения составило 6.105 нс. Для реализации потребовалось 3 модуля умножения DSP.

Всего в представленной микросхеме 240 умножителей. Максимальный unroll factor для цикла Mult, который можно использовать, чтобы использовались только DSP, равен 64, 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\_1 представлены на рисунке 10.

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

Автоматически созданное описание

*Рис.10. Результаты планирования для sol1\_1*

Как можно видеть на рисунке 10, как и в решении sol1 итерация цикла длится 4 такта, но из-за указанной в параметрах директивы задержки равной 2 такта, операция умножения теперь занимает 2 такта.

5.3. Решение 1\_2

5.3.1. Синтез

В решении sol1\_2 используется директива bind\_op с параметрами op=mul, impl=fabric, latency=2 для переменной temp\_mult, которой будет присваиваться результат выполнения операции. Эта директива позволяет связать запланированную операцию умножения с логическими элементами. Временные параметры, производительность и аппаратные затраты решения представлены на рисунке 11.

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

Автоматически созданное описание

*Рис.11. Результаты синтеза для sol1\_2*

Estimated timing для решения составило 12.912 нс. Для реализации потребовалось 0 модулей умножения DSP и 1102 логических элемента LUT.

Всего в представленной микросхеме 63400 LUT. Максимальный unroll factor для цикла Mult, который можно использовать, чтобы использовались только LUT, равен 32, так как при данном значении требуется число LUT, не превышающее максимальное значение микросхемы. При большем значении factor оно превысит число логических элементов.

Результаты планирования для sol1\_2 представлены на рисунке 12.

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

Автоматически созданное описание

*Рис.12. Результаты планирования для sol1\_2*

Как можно видеть на рисунке 12, результаты планирования схожи с решением sol1\_1 и итерация цикла длится 4 такта, операция умножения теперь занимает 2 такта.

5.4. Решение 2

5.4.1. Синтез

В решении sol2 используется директива unroll с factor=64 для того, чтобы использовались только DSP. Также вместе с unroll используется директива array\_partition factor=64 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 13.

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

Автоматически созданное описание

*Рис.13. Результаты синтеза для sol2*

Estimated timing для решения составило 7.215 нс. Для реализации потребовалось 192 модуля умножения DSP.

Результаты планирования для sol2 представлены на рисунках 14–15.

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

Автоматически созданное описание

*Рис.14. Результаты планирования для sol2*

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

Автоматически созданное описание*

*Рис.15. Результаты планирования для sol2*

Как можно видеть на рисунках 14–15, итерация цикла стала равна трём тактам. Операция умножения занимает 1 такт. Параллельно считываются 64 значения b и 64 значения c. Также 64 операции умножения выполняются параллельно и их результат записывается в 64 элемента массива a. В данном цикле была выполнена конвейеризация, что видно на рисунке 14. II составляет 5 тактов.

5.4.2. C/RTL моделирование

Временная диаграмма для sol2 представлена на рисунке 16.

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

Автоматически созданное описание

*Рис.16. Временная диаграмма C/RTL для sol2*

На временной диаграмме видно, что каждый из массивов b и c был разбит на 64 блока по два элемента. 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 4 решений с разными значениями bind\_op. Результаты сравнения приведены на рисунке 17.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

*Рис.17. 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 18–19.

Изображение выглядит как текст, число, снимок экрана, Шрифт

Автоматически созданное описание

*Рис.18. Электронная таблица для всех решений*

Изображение выглядит как линия, График, снимок экрана, диаграмма

Автоматически созданное описание

*Рис.19. 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\_z1\_test.cpp был создан отдельный, модернизированный, тест lab7\_z1\_testSW.cpp, который содержит операторы измерения времени выполнения и имеет большее число запусков равное 32. Код теста представлен на рисунке 20.

Изображение выглядит как текст, снимок экрана, дисплей

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, число, снимок экрана

Автоматически созданное описание

*Рис.20. Код теста* *lab7\_z1\_testSW.cpp*

Также в исходный код функции для реализации многопоточности была добавлена pragma omp со значением 8, что соответствует числу ядер на ПК (рисунок 21).

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

*Рис.21. Исходный код многопоточной функции lab7\_z1*

Для компиляции теста был установлен компилятор GCC версии 13.2.0 (рисунок 22). 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, Шрифт, снимок экрана, черный

Автоматически созданное описание

*Рис.22. Выполнение компиляции теста*

Результаты запуска для одного потока.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время. нс** | **Медиана, нс** |
| 8192 | 14562.5 | 36000 | 18562.5 |
| 16384 | 39343.8 | 95333.3 | 49812.5 |
| 32768 | 59937.5 | 97000 | 68562.5 |
| 65536 | 94281.2 | 268000 | 115438 |

Результаты запуска для 8 потоков.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Минимальное время, нс** | **Максимальное время, нс** | **Медиана, нс** |
| 8192 | 20406.2 | 41000 | 28437.5 |
| 16384 | 17437.5 | 26000 | 17937.5 |
| 32768 | 34272.7 | 36500 | 34655.2 |
| 65536 | 86566.7 | 156000 | 93250 |

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 | 132 | 8.232 | 1086.624 |
| 16384 | 260 | 8.232 | 2140.32 |
| 32768 | 516 | 8.232 | 4247.712 |
| 65536 | 1028 | 8.232 | 8462.496 |

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 23–24.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

*Рис.23. Электронная таблица для ПК (1 и 8 ядер) и аппаратной реализации*

Изображение выглядит как линия, График, снимок экрана, диаграмма

Автоматически созданное описание

*Рис.24. График для ПК (1 и 8 ядер) и аппаратной реализации*

На графике можно видеть, что результаты аппаратной реализации растут медленнее и она выигрывает по производительности относительно 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 однопоточная реализация на ПК выигрывает в производительности на небольших значениях, но проигрывает на больших (начиная с 16384) по сравнению с многопоточной.

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 данных int.

Для создания проекта и работы с ним был создан TCL скрипт, содержащий 4 решения. Первое решения является базовым, а в остальных используется директива bind\_op с разными параметрами impl и latency для операции умножения. 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 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 однопоточная реализация на ПК выигрывает в производительности на небольших значениях N, но проигрывает на больших (начиная с N=16384) по сравнению с многопоточной.