Санкт-Петербургский Политехнический Университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ

Дисциплина: Проектирование реконфигурируемых гибридных вычислительных систем

Вариант: 7

Выполнил студент гр. Шеметов С.А.

гр.5140901/21502

Руководитель, доцент Антонов А.П.

«\_\_» \_\_\_\_\_\_ 2023

Санкт-Петербург

2023

Оглавление

[Оглавление 2](#_Toc162202350)

[1. Вариант задания 3](#_Toc162202351)

[2. Этапы курсовой работы 3](#_Toc162202352)

[2.1. Этап 1 3](#_Toc162202353)

[2.2. Этап 2 3](#_Toc162202354)

[3. Исходный код функции 4](#_Toc162202355)

[4. Исходный код теста 5](#_Toc162202356)

[4.1. Моделирование 5](#_Toc162202357)

[5. Исследование вариантов аппаратной реализации 8](#_Toc162202358)

[6. Оптимизация функции 11](#_Toc162202359)

[6.1. Решение 2 11](#_Toc162202360)

[6.2. Решения 2\_1 и 2\_2 13](#_Toc162202361)

[6.3. Решение 2\_3 16](#_Toc162202362)

[6.4. Решение 2\_4 18](#_Toc162202363)

[6.5. Решение 2\_5 20](#_Toc162202364)

[6.6. Решение 2\_6 21](#_Toc162202365)

[6.7. Поиск максимального unroll factor для разных типов данных 23](#_Toc162202366)

[6.8. Реализация решений в TCL 25](#_Toc162202367)

[6.9. Сравнение решений 28](#_Toc162202368)

[7. Измерение времени выполнения 29](#_Toc162202369)

[7.1. Измерение времени выполнения на ПК 29](#_Toc162202370)

[7.2. Измерение времени выполнения на аппаратной реализации 33](#_Toc162202371)

[7.3. Сравнение результатов 33](#_Toc162202372)

[8. Исследовании производительности решения для разных типов данных 34](#_Toc162202373)

[8.1. Тип данных long long 35](#_Toc162202374)

[8.2. Тип данных float 36](#_Toc162202375)

[8.3. Тип данных double 38](#_Toc162202376)

[9. Вывод 40](#_Toc162202377)

[Приложение 1. Описание содержимого архива 42](#_Toc162202378)

1. Вариант задания

Надо найти вектора Х1 и Х2, из N элементов, каждый i элемент которых равен:

D[i] = b[i] 2 - 4 \* a[i] \* c[i]

Х1[i] = (- b[i] - sqrt (D[i])) / (2 \* a[i])

Х2[i] = (- b[i] + sqrt (D[i])) / (2 \* a[i])

* N при моделировании, оптимизации и Co-simulation – 8.
* N для исследования производительности – 8192, 16384, 32768, 65 536.
* Надо: провести исследования оптимизированного решения для типов данных int, long long, float, double.

2. Этапы курсовой работы

2.1. Этап 1

1. Разработка описания создаваемой функции на языке С++.

2. Разработка теста на языке С++.

3. Моделирование созданной функции.

4. Исследование вариантов аппаратной реализации функции для периода тактового сигнала 6, 10, 14 нс (uncertainty = 1нс). Построение в xls таблицы и графиков. Выбор оптимального (мин аппаратных затрат, мах производительности) решения (периода тактового сигнала) для дальнейших исследований.

* При данном исследовании следует выключить конвейеризацию всех циклов.

5. Использование известных Вам директив, актуальных для оптимизации вашей функции: цель – мах производительность, ограничения – доступные аппаратные ресурсы микросхемы xc7a200tsbv484-2.

* FIFO vs RAM для реализации памяти массивов.
* UNROLL, Array\_Partition, Array\_Reshape, Pipeline, DataFlow (с выбором FIFO vs PingPong), Inline, Loop Merge. Следует привести соображения по выбору тех или иных директив для оптимизации.
  + Все, даже неудачные, попытки оптимизации желательно оформить отдельными решениями и представить в пояснительной записке.
* Для сравнения результатов следует заполнить таблицу и привести графики (решение <-> аппаратные затраты и быстродействие).

2.2. Этап 2

1. Создание теста для анализа временных затрат при программной реализации функции (на Вашем ПК). Тест должен содержать не менее 32 запусков исследуемой функции.

2. Оценка производительности аппаратного решения.

3. Оценка производительности программной реализации на ПК.

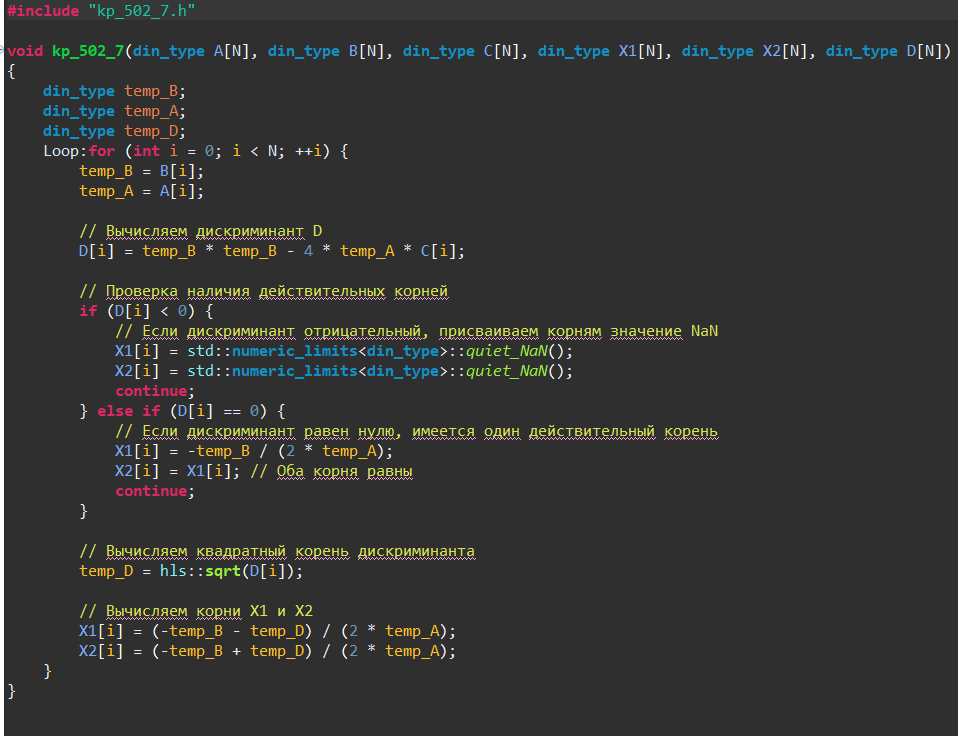
Для оценки выбираем медиану среди полученных при каждом из запусков функции 32 значений временных затрат.

* Надо реализовать:
  + одноядерное выполнение вашей функции на вашем ПК;
  + многоядерное выполнение.
* При исследовании производительности решения для типов данных int, long long, float, double исходные данные входных массивов должны быть псевдослучайными из всего диапазона соответствующих типов данных.

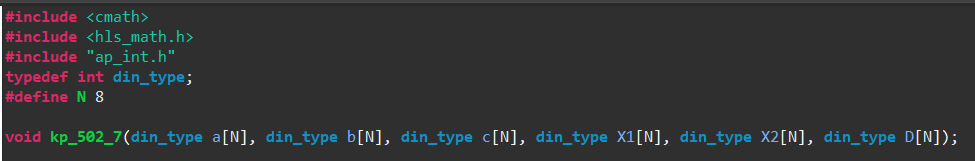
4. Сравнительный анализ программных и аппаратной реализаций (составить таблицу и привести графики).

3. Исходный код функции

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



*Рис.1. Исходный код функции kp\_502\_7*



*Рис.2. Исходный код заголовочного файла* *kp\_502\_7*

Функция принимает на вход 6 входных массивов и 3 выходных. Каждый массив имеет размер N и тип данных int. Функция выполняет вычисление значений массивов X1[i], X2[i], D[i] по формуле

D[i] = b[i] 2 - 4 \* a[i] \* c[i]

Х1[i] = (- b[i] - sqrt (D[i])) / (2 \* a[i])

Х2[i] = (- b[i] + sqrt (D[i])) / (2 \* a[i])

в цикле. Значение a, b и d используются в формуле несколько раз, поэтому чтобы сократить число считываний на одной итерации цикла, создаётся временные переменные temp.

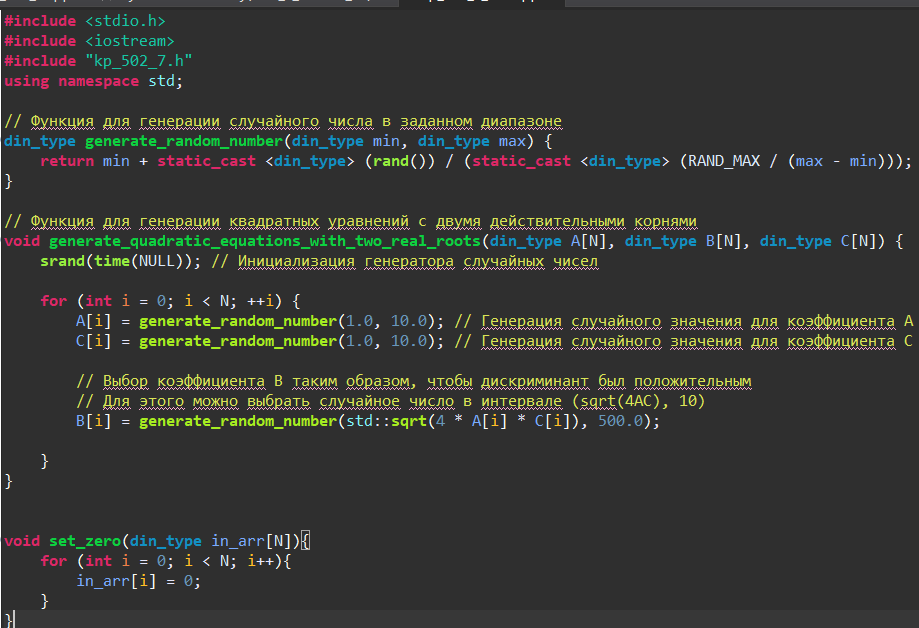
4. Исходный код теста

Исходный код теста для проверки функции kp\_502\_7 на рисунке 3. В тесте задаётся переменная pass, определяющая успешное/неуспешное выполнение теста; 4 входных массива; два выходных массива – ожидаемый и полученный. Затем функция в цикле запускается три раза. Перед каждым запуском выходные массивы обнуляются вспомогательной функцией set\_zero.

После этого во внутреннем цикле входные массивы заполняются случайными числами с помощью функции генерирующей квадратные уравнения, далее формируется ожидаемый результат. Запускается функция kp\_502\_7 и её значения сравниваются с ожидаемым результатом. Если значения расходятся, то переменной pass присваивается значение 1 и в консоль выводится информация об ошибке. По завершению формируется сообщение об успешном/неуспешном выполнении теста, исходя из значения pass.

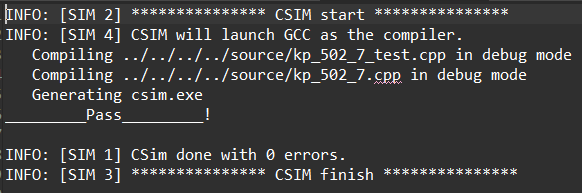
4.1. Моделирование

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.



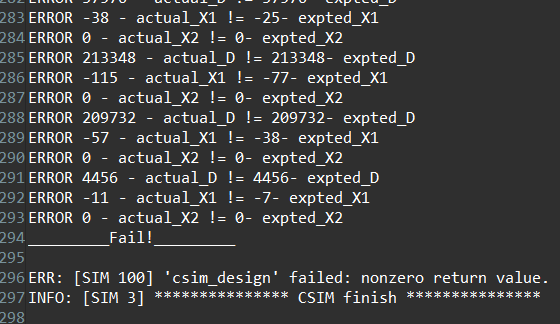


*Рис.3. Исходный код теста*



*Рис.4. Лог результата выполнения моделирования*

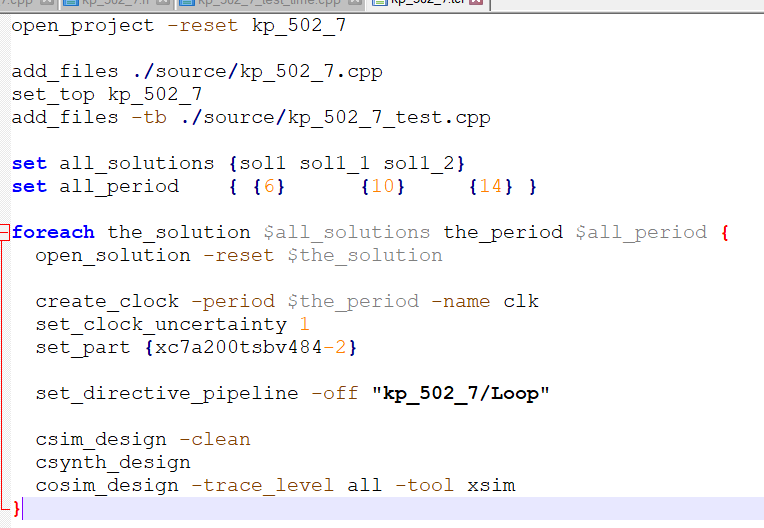
Для проверки правильности работы теста в него было внесено изменение в ожидаемый результат - expted\_X1[i] = (-B[i] - hls::sqrt(expted\_D[i])) / (3 \* A[i]). Результат моделирования приведён на рисунке 5. Он показывает, что тест отрабатывает ошибку корректно.



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

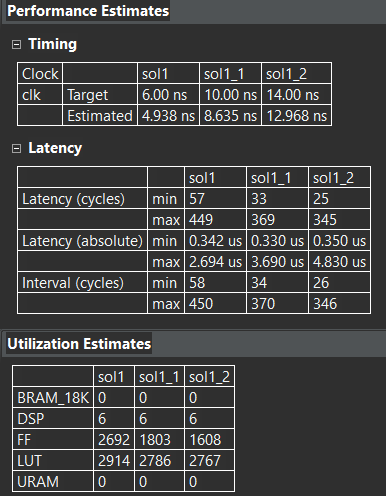
5. Исследование вариантов аппаратной реализации

Для исследования аппаратной реализации функции было реализованы три решения с периодами тактового сигнала 6, 10, 14. Скрипт для запуска решений представлен в файле *kp\_502\_7.tcl* (рисунок 6).



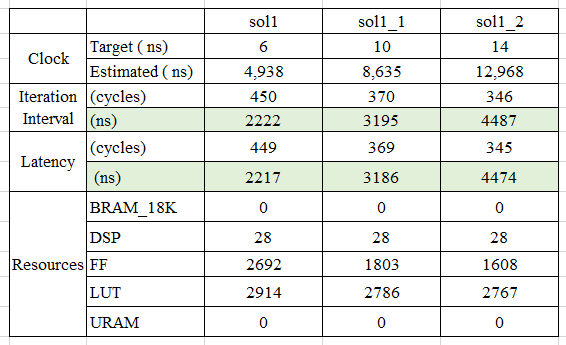
*Рис.6. Исследование вариантов аппаратной реализации (kp\_502\_7.tcl)*

Временные параметры, производительность и аппаратные затраты этих решений представлены на рисунке 7.

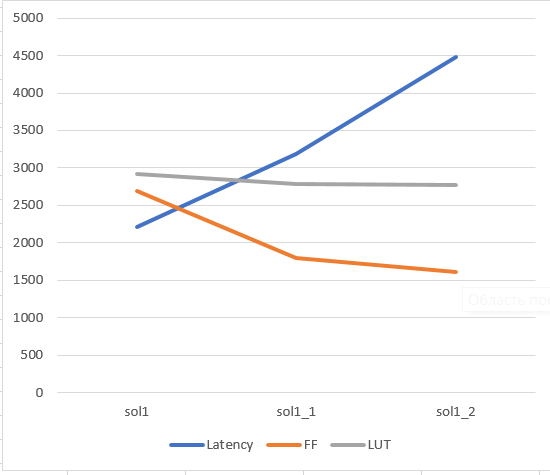


*Рис.7. Сравнение решений с разным периодом*

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 8–9.



*Рис.8. Электронная таблица для всех решений с разным периодом*



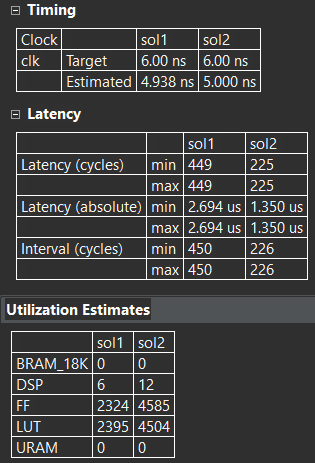
*Рис.9. График для всех решений с разным периодом*

Оптимальным решением с точки зрения максимальной производительности и минимальных аппаратных затрат является sol1. На его основе будут проводиться дальнейшие исследования.

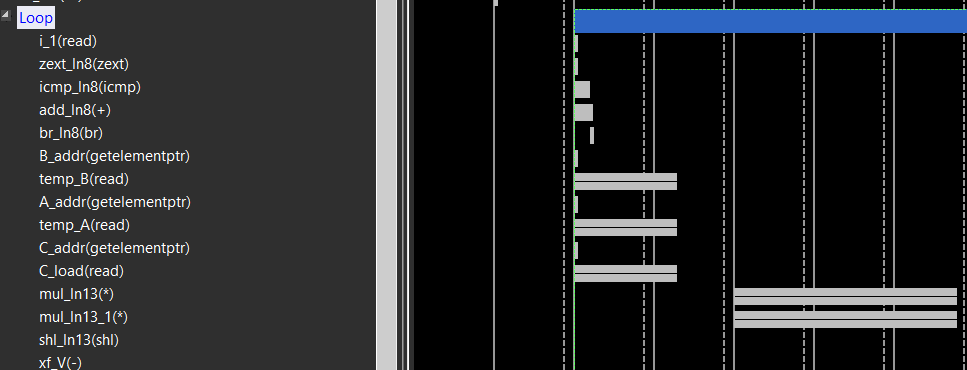
6. Оптимизация функции

6.1. Решение 2

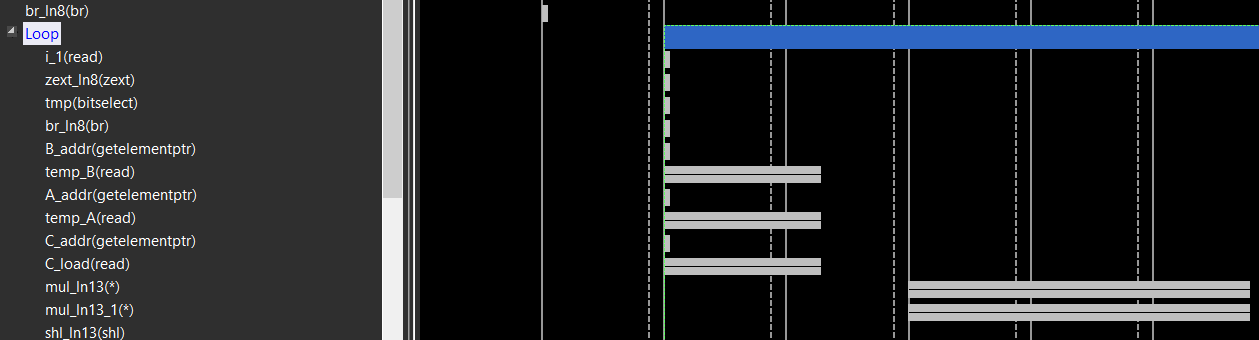
На основе sol1 создаётся решение sol2, в котором выполняется распараллеливание функции с помощью директивы unroll с factor=2. По умолчанию для данного решения используется двухпортовая память. Конвейеризация отключена. Результаты сравнения представлены на рисунке 10–12.



*Рис.10. Сравнение решений с применением unroll factor=2*



*Рис.11. Результаты планировщика для решения sol1 (базовое решение)*



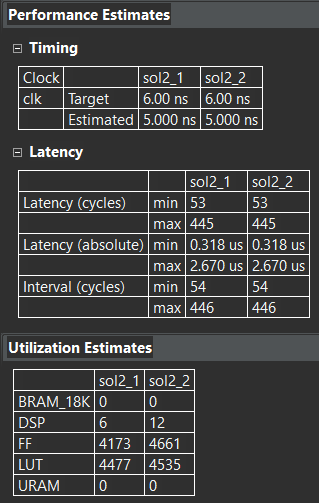
*Рис.12. Результаты планировщика для решения sol2 (unroll 2)*

На рисунках 11–12 видно, что число происходит дублирование функции и каждая операция выполняется дважды за счёт unroll factor=2.

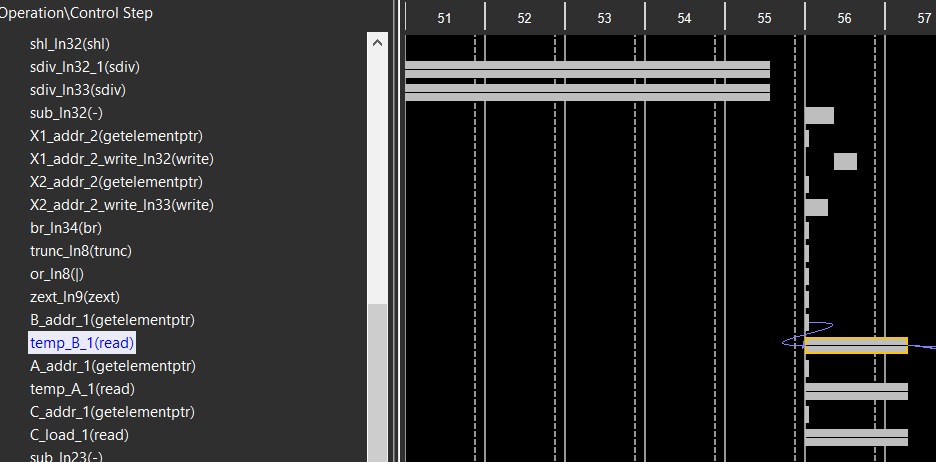
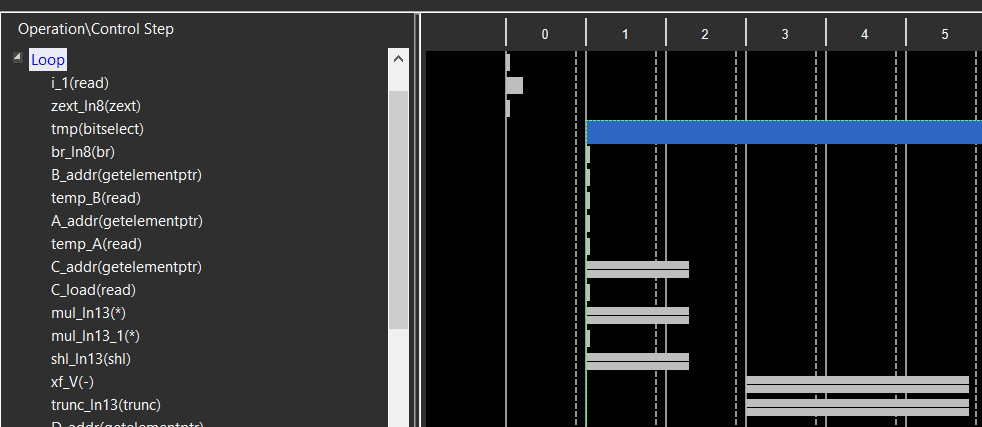
Использование распараллеливания с factor=2 вместе двухпортовой памятью, которая устанавливается по умолчанию, привело к улучшению производительности примерно в два раза. Но при этом аппаратные затраты выросли.

6.2. Решения 2\_1 и 2\_2

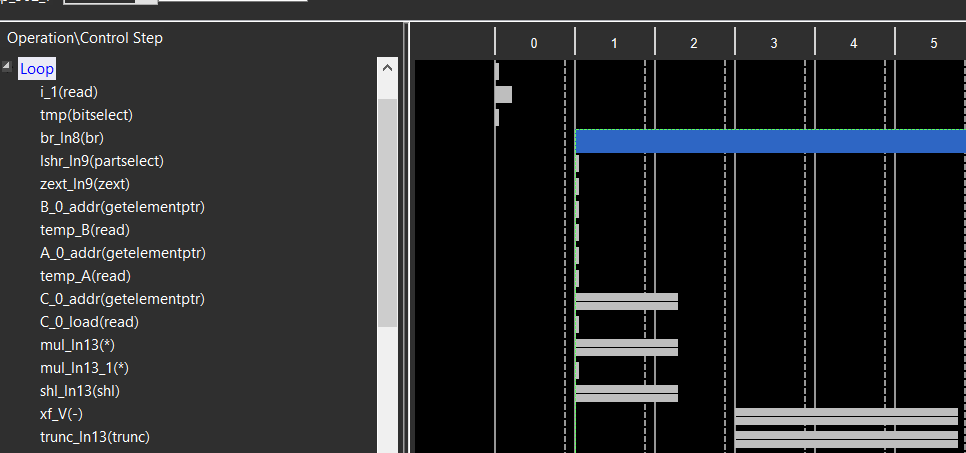
Теперь необходимо сравнить работу unroll с однопортовой и двухпортовой памятью. Для этого создаются два решения: в sol2\_1 явно задаётся двухпортовая память, а в sol2\_2 – однопортовая и директива array\_partition с factor=2 и type=cyclic. Результаты сравнения представлены на рисунке 13–15.

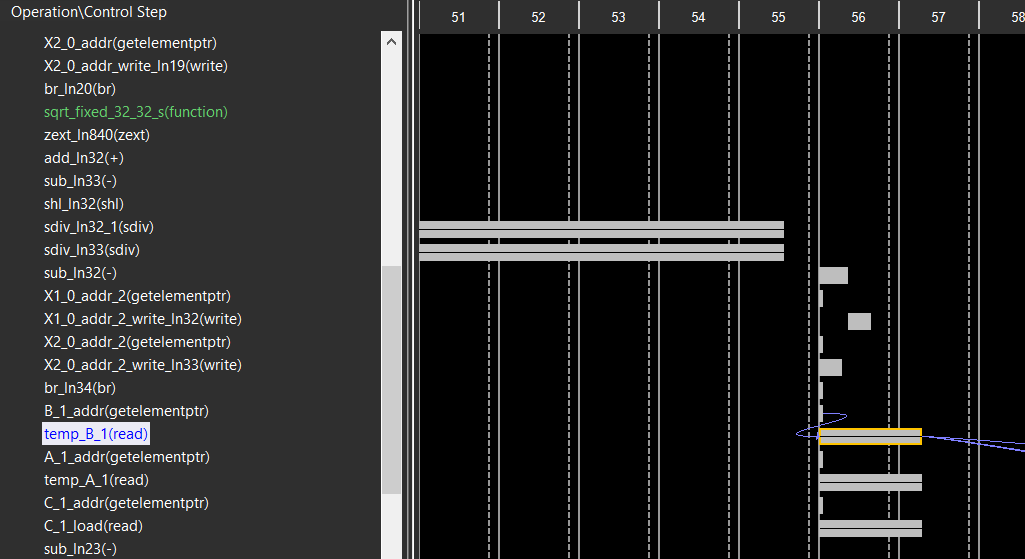


*Рис.13. Сравнение решений с разными типами памяти*



*Рис.14. Результаты планировщика для решения sol2\_1 (ram\_2p)*



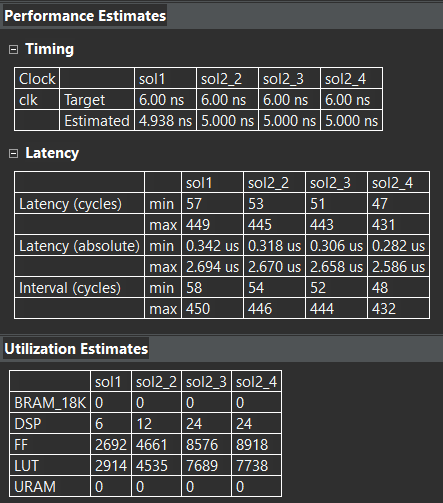


*Рис.15. Результаты планировщика для решения sol2\_2 (ram\_1p)*

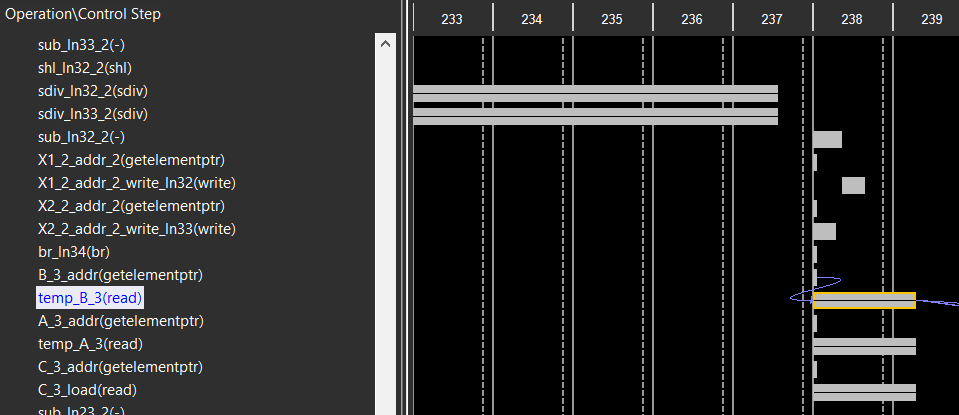
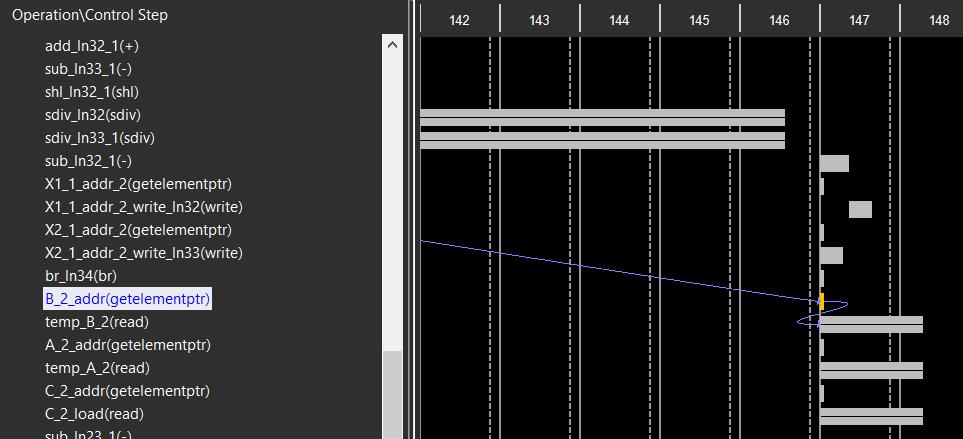
Как можно видеть на рисунке 13, оба решения имеют одинаковую производительность. Поэтому в последующих решениях будет использоваться однопортовая память.

6.3. Решение 2\_3

Для данного решения unroll factor увеличивается до 4, а также используется директива array\_partition для всех массивов с параметрами factor=4 и type=cyclic, чтобы организовать параллельное чтение данных. Конвейеризации отключена. Результаты сравнения представлены на рисунке 16–17.



*Рис.16. Сравнение решений с применением unroll factor=4 (array\_partition)*

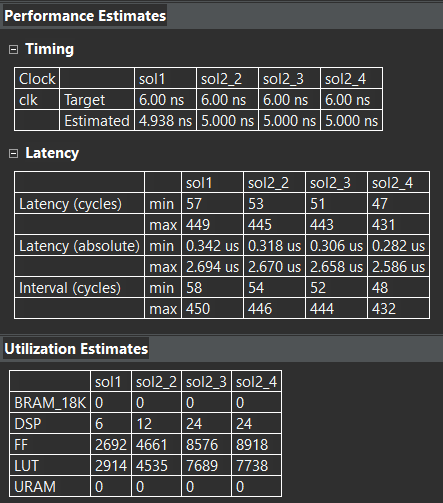


*Рис.17. Результаты планировщика для решения sol2\_3 (unroll 4, array\_partition)*

В данном решении число параллельно выполняемых операций выросло в двое относительно предыдущих решений, а соответственно и производительность выросла в два раза.

6.4. Решение 2\_4

Для данного решения unroll factor=4, а также вместо директивы array\_partition используется array\_reshape для всех массивов с параметрами factor=4 и type=cyclic. Конвейеризации отключена. Результаты сравнения представлены на рисунке 18–19.



*Рис.18. Сравнение решений с применением unroll factor=4 (array\_reshape)*

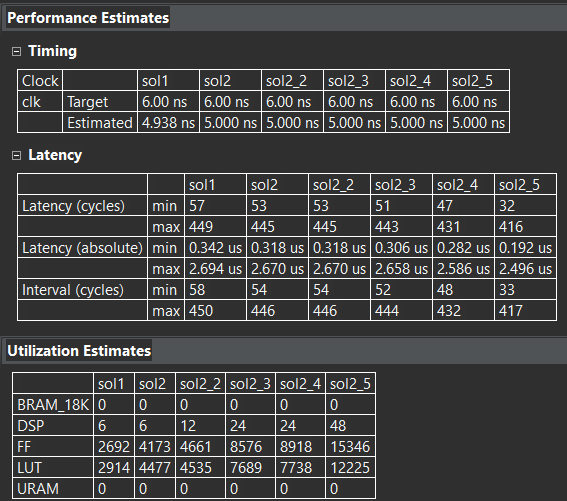


*Рис.19. Результаты планировщика для решения sol2\_4 (unroll 4, array\_reshape)*

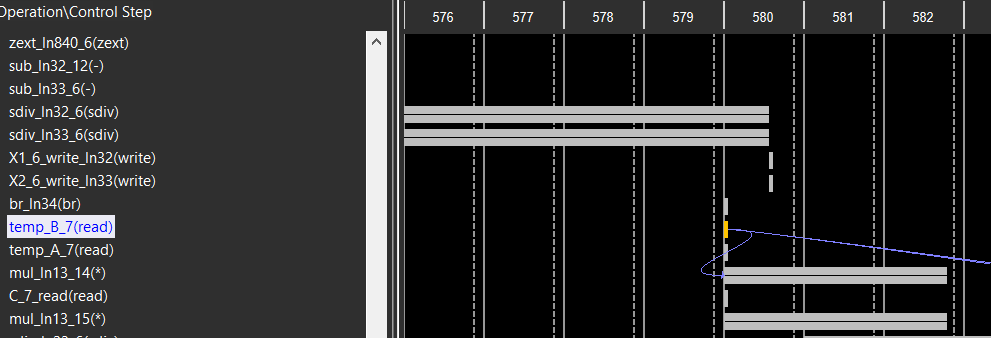
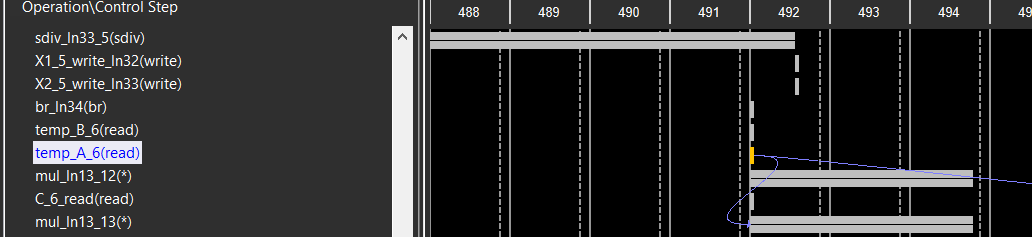
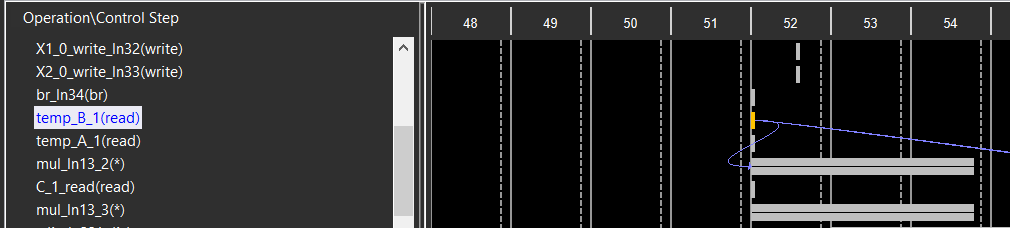
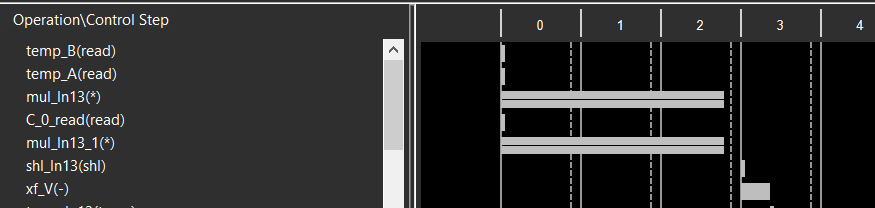
Как можно видеть на рисунке 18, применение array\_reshape директивы незначительно ухудшило показатели затраченных ресурсов. Число тактов Latency относительно sol2\_3 с array\_partition не изменилось, как и значение Estimated timing. Для дальнейших решений будет использоваться array\_partition.

6.5. Решение 2\_5

Для данного решения unroll factor увеличивается до 8, что равно максимальному числе элементов в каждом массиве, а также используется директива array\_partition для всех массивов с параметрами factor=8 и type=cyclic, чтобы организовать параллельное чтение данных. Конвейеризации отключена. Результаты сравнения представлены на рисунке 20–21.



*Рис.20. Сравнение решений с применением unroll factor=8 (array\_ partition)*

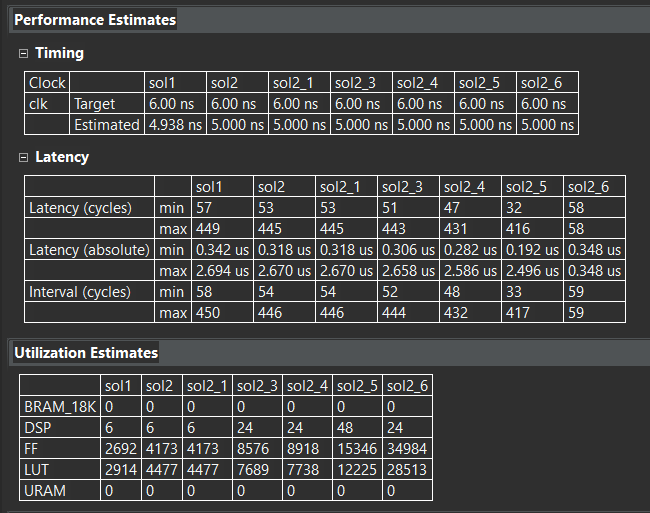


*Рис.21. Результаты планировщика для решения sol2\_5 (unroll 8, array\_partition)*

В данном решении производительность выросла относительно sol2\_3 и цикл выполняется за одну итерацию.

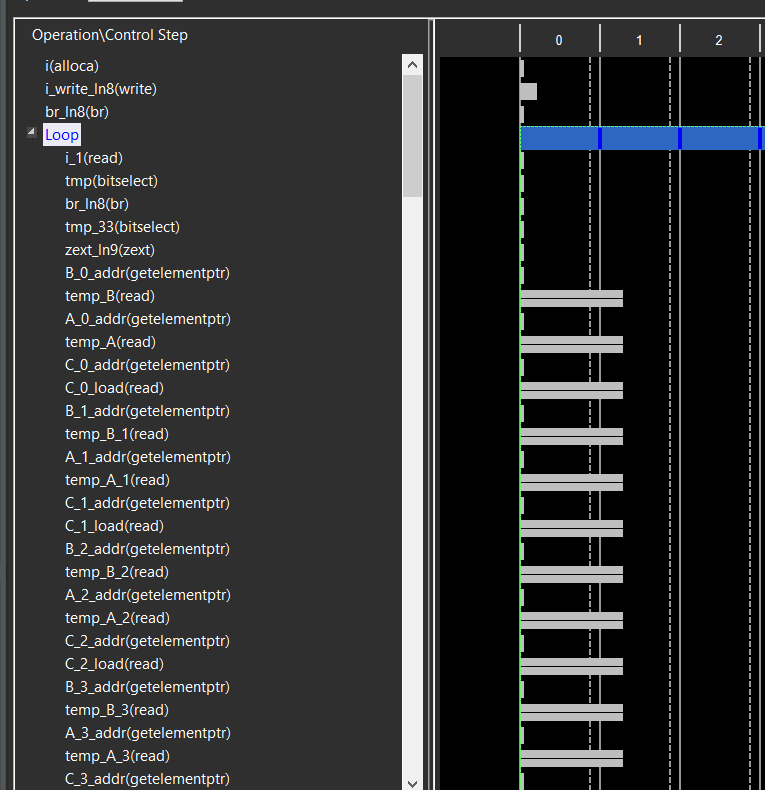
6.6. Решение 2\_6

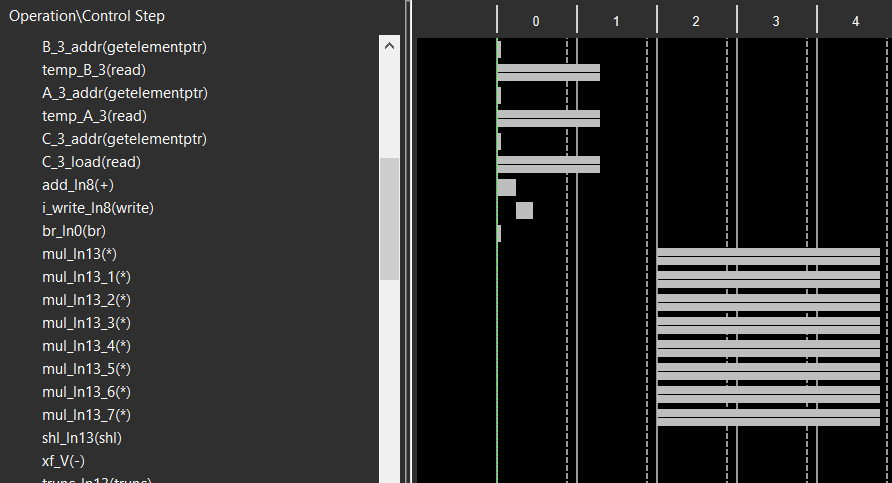
Данное решение основано на sol2\_5, однако в нём включена конвейеризация. Результаты сравнения представлены на рисунке 22.



*Рис.22. Сравнение решений с применением unroll factor=8 pipeline (array\_ partition)*

Применение конвейеризации не изменяет производительность, это может быть связанно с тем что весь цикл выполняется за одну итерацию (рисунок 23). Однако при дальнейшем исследовании на больших значениях N, конвейеризация требуется, для улучшения производительности.





*Рис.23. Результаты планировщика для решения sol2\_6 (unroll 8, array\_partition, pipeline)*

6.7. Поиск максимального unroll factor для разных типов данных

По результатам решений sol2-sol2\_6 видно, что с ростом unroll factor производительность возрастает, но также растут и аппаратные затраты. Необходимо определить, до какого значения можно увеличить значение unroll factor, чтобы задействовать все умножители использующейся микросхемы. На рисунке 24 видно, что микросхема имеет 740 умножителей и 134600 LUT.

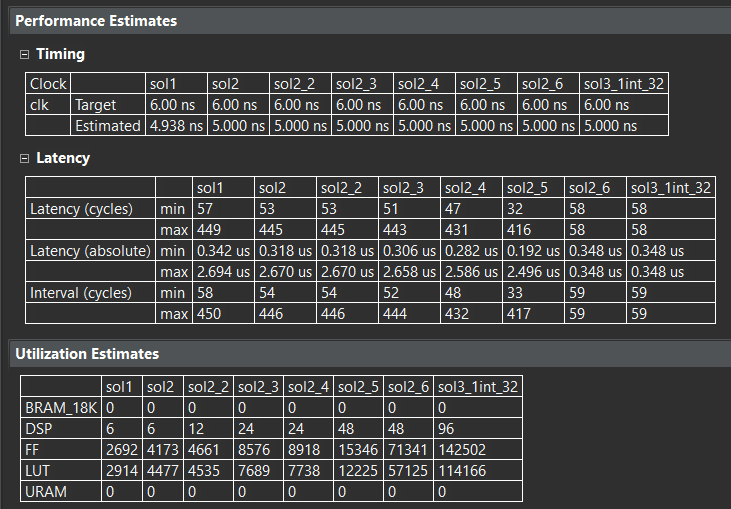
Изображение выглядит как снимок экрана, текст, линия, Мультимедийное программное обеспечение

Автоматически созданное описание

*Рис.24. Аппаратные ресурсы микросхемы*

Все предложенные для исследования типы данных будут иметь отличный максимальный unroll factor. Поэтому необходимо создать 4 решения sol3\_1-sol3\_4 в каждом из которых будет подобран unroll фактор для конкретного типа данных. Исследование проводится на больших размерах данных, так как изначальный размер массива равен N=8.

Решение для типа int – sol3\_1. Размер данных N=32. В результате исследования было установлено, что для массивов типа int максимальной производительности с учётом возможностей микросхемы достигает unroll factor=32. В таком случае DSP=96, а LUT=114166 (рисунок 25).



*Рис.25. Максимальный unroll factor для int*

Аналогичные исследования для других типах данных привели к следующим результатам. Размер массива N в исследовании соответствует unroll factor для каждого из типов.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **sol** | **Тип** | **Unroll** | **Estimated Timing** | **DSP** | **FF** | **LUT** |
| sol3\_1 | int | 32 | 5.000 | 96 | 68246 | 114166 |
| sol3\_2 | double | 16 | 4.888 | 448 | 42964 | 33314 |
| sol3\_3 | float | 64 | 4.998 | 640 | 73671 | 56837 |
| sol3\_4 | long long | 32 | 4.203 | 384 | 33254 | 22878 |

6.8. Реализация решений в TCL

В результате выполнения оптимизаций, был создан TCL скрипт *kp\_502\_7\_2.tcl*, представленный на рисунке 26 (старая нумерация 25). В данном скрипе представлены базовое решение с периодом 6 нс и решения, полученные в ходе оптимизации.

Решение sol1\_2 – базовое решение с периодом 6 нс и отключенной конвейеризацией.

Решение sol2 – выполнение первой оптимизации. Распараллеливание функции с помощью unroll factor=2. Конвейеризация отключена.

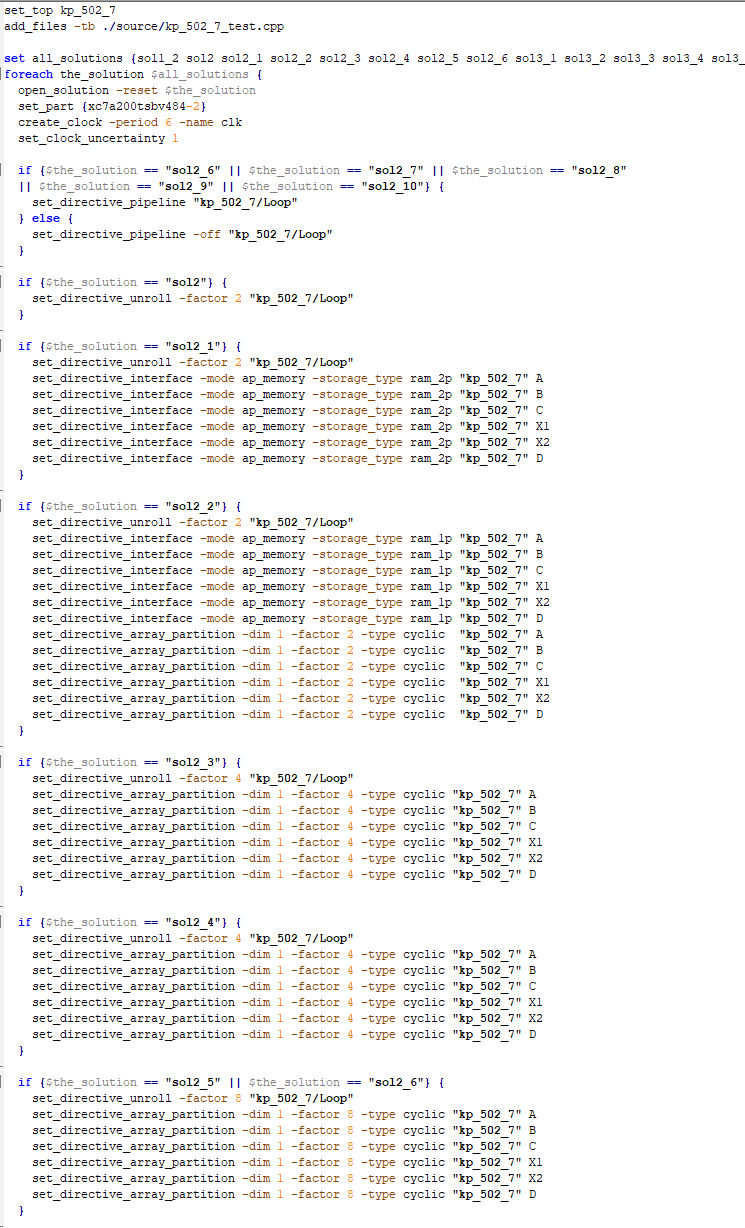
Решения sol2\_1-sol2\_2 – решения для сравнения влияния типа памяти на производительность. В sol2\_1 заданы unroll factor=2 и двухпортовая память. В решении sol2\_2 - unroll factor=2, array\_partition cyclic, factor=2 и однопортовая память. Конвейеризация отключена.

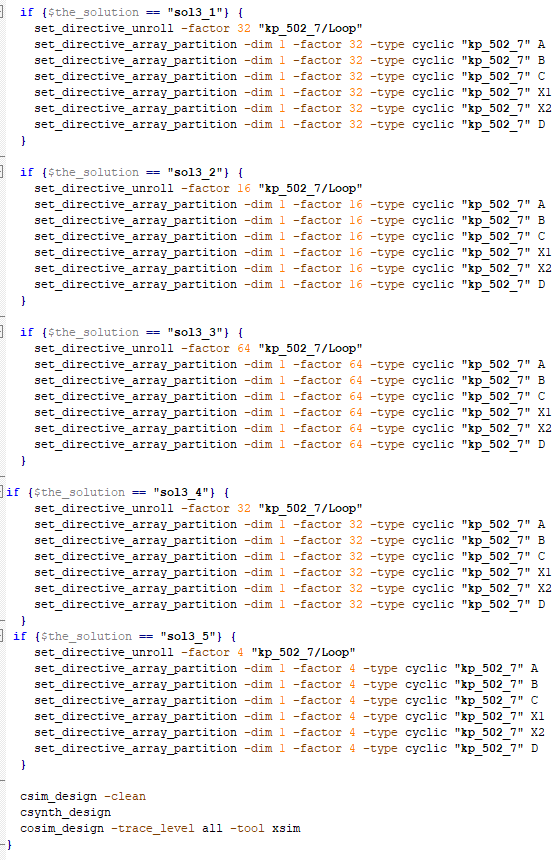
Решения sol2\_3-sol2\_4 – решение для сравнения влияния array\_partition и array\_reshape на производительность. В sol2\_3 - unroll factor=4, array\_partition cyclic, factor=4. В sol2\_4 - unroll factor=4, array\_reshape cyclic, factor=4. Конвейеризация отключена.

Решение sol2\_5 – повышение unroll factor до максимального значения, при размере массивов N=8, array\_partition cyclic, factor=8. Конвейеризация отключена.

Решение sol2\_6 – включение конвейеризации относительно sol2\_5, unroll factor=8, array\_partition cyclic, factor=8.

Решения sol3\_1-sol3\_4– решения с максимальным unroll factor для каждого из 4 типов данных, предложенных к исследованию. Исследование проводится на массивах больших размеров (значение N в h-файле увеличено). Значения unroll factor определены экспериментальным путём. Каждое из 4 решений соответствует своему типу данных: sol3\_1 – int, sol3\_2 – double, sol3\_3 – float, sol3\_4 – long long.

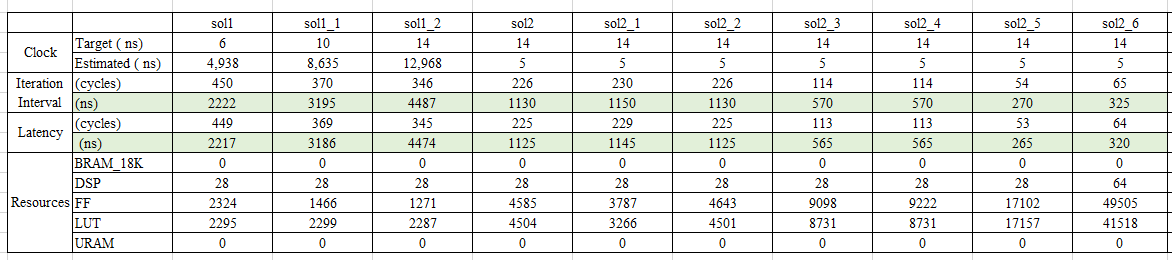




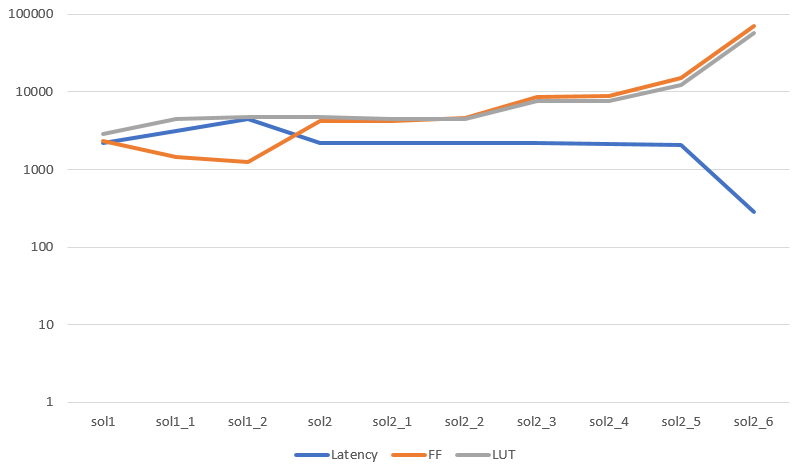
*Рис.26. Поиск оптимального решения (kp\_502\_7\_2.tcl)*

6.9. Сравнение решений

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 27–28.



*Рис.27. Электронная таблица для всех решений с разным unroll*



*Рис.28. График для всех решений с разным unroll*

С точки зрения производительности оптимальным решением являются sol2\_5, так как оно имеет наименьшие значения Latency и затрачивает меньше времени. Это обусловлено использованием директив оптимизации. Однако дальнейшие решение sol 3\_1 может быть более выигрышным в том случае, если значение N будет увеличено и не будет равно числу элементов массива, тогда будет применена конвейеризация.

7. Измерение времени выполнения

7.1. Измерение времени выполнения на ПК

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

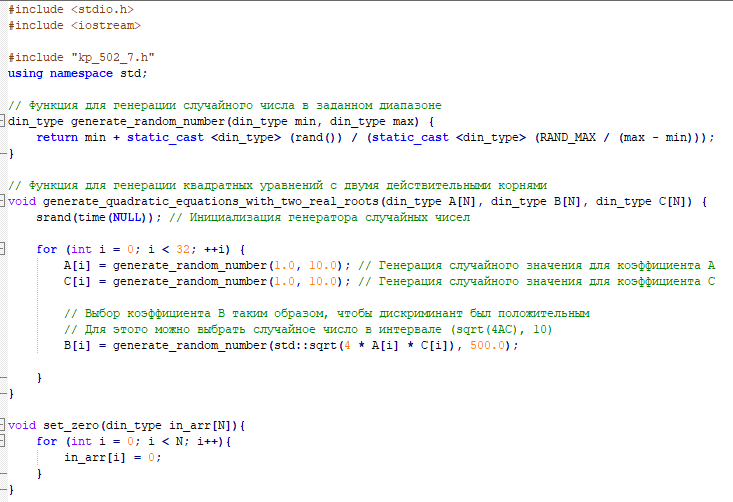
|  |  |  |  |
| --- | --- | --- | --- |
| **Тип процессора** | **Базовая частота работы** | **Максимальная частота работы** | **ОЗУ** |
| AMD Ryzen 5 3600X | 3,8 ГГц | 4,4 ГГц | 16 Гб |

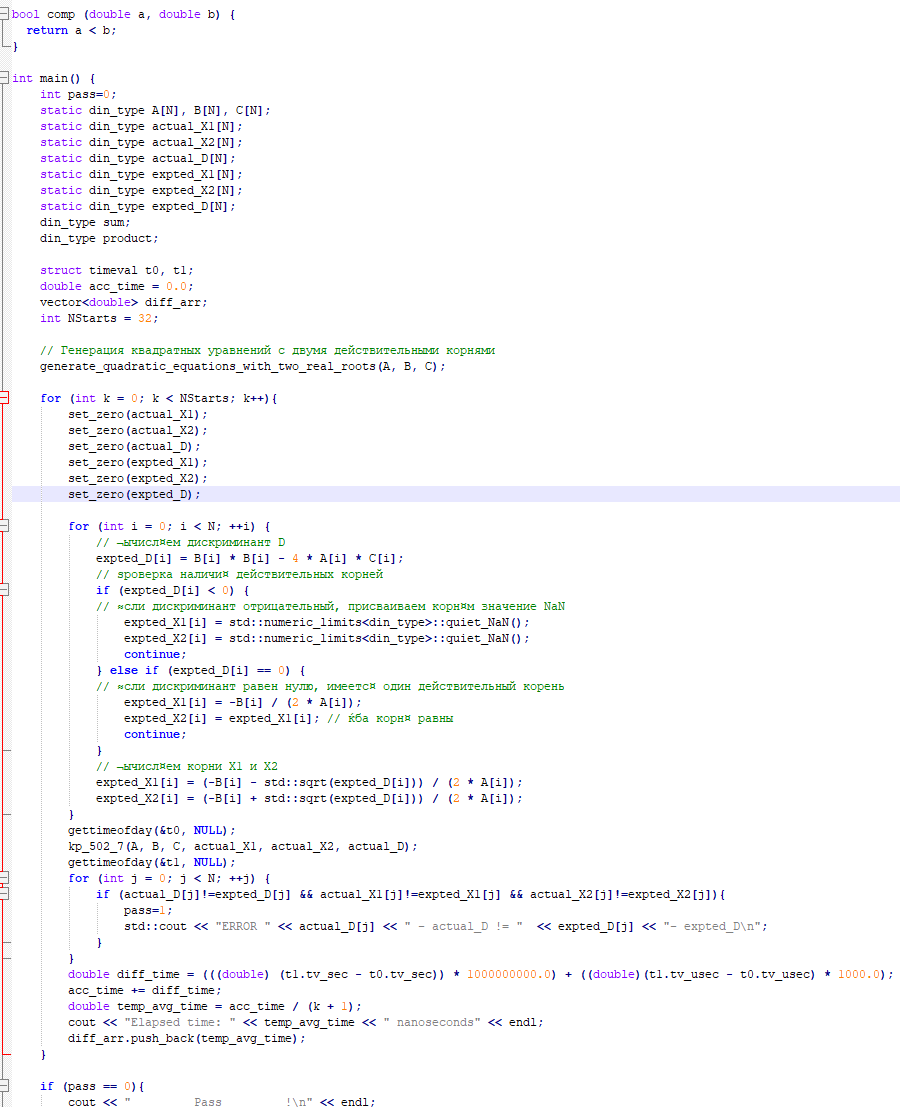
На базе исходного теста был создан отдельный, модернизированный, тест kp\_502\_7\_test\_time.cpp, который содержит операторы измерения времени выполнения и имеет большее число запусков равное 32. Код теста представлен на рисунке 29.

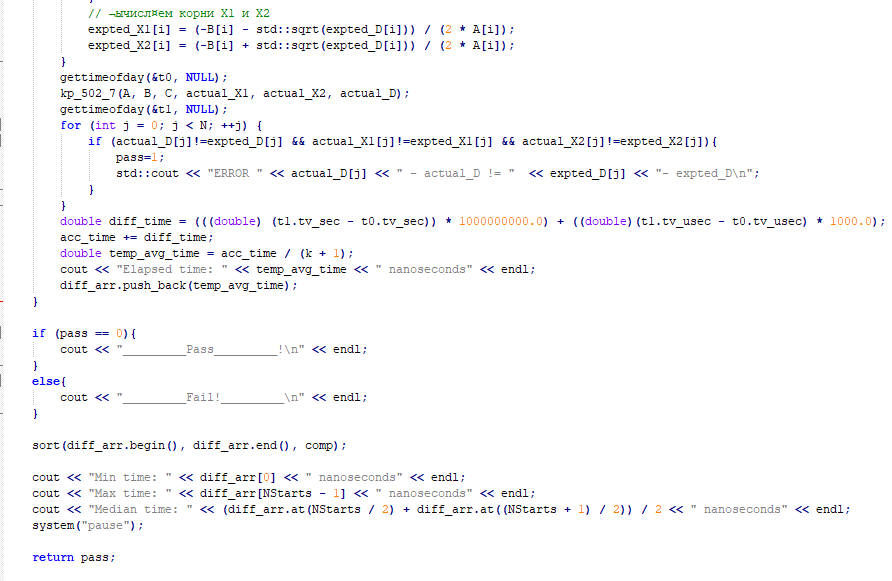
В нём выделяется память для входных массивов, задаются переменные для измерения времени и число запусков функции равное 32.

Затем в цикле аналогично тесту для аппаратной реализации очищаются выходные массивы, случайным образом заполняются входные массивы и формируется ожидаемый результат. После этого выполняется запуск функции и замер времени её выполнения. Полученный результат сравнивается с ожидаемым. Время выполнения итерации переводится в нс и сохраняется в векторе.

После всех запусков функции полученные временные результаты сортируются и на их основе вычисляются минимальное, максимальное и медианное время выполнения.



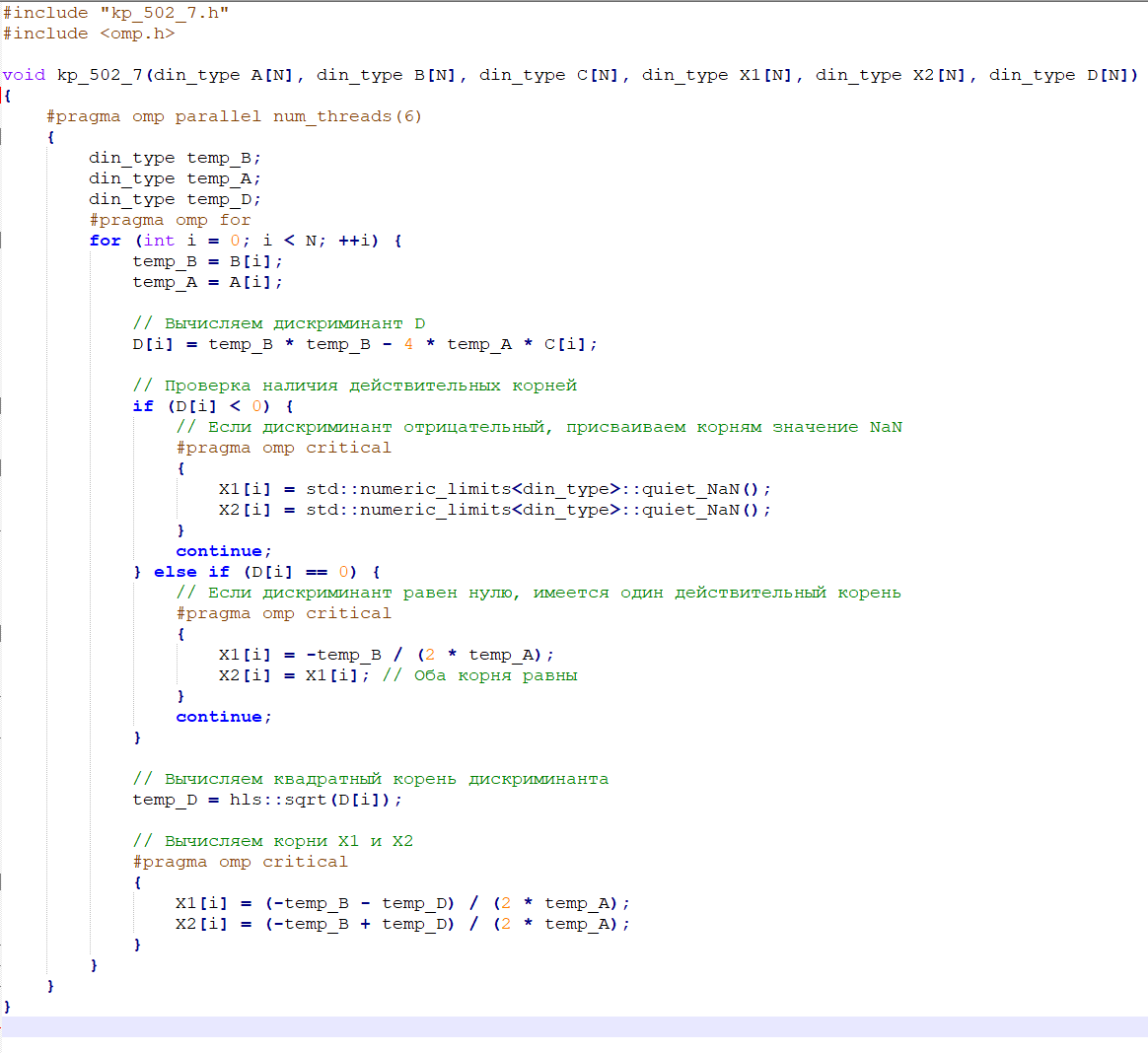




*Рис.29. Код теста* *kp\_502\_7\_testSW.cpp*

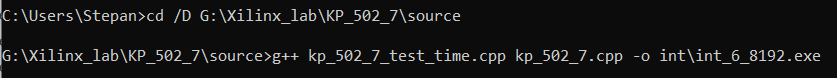
Исходная функция на рисунке 1, представляет собой однопоточную реализацию на ПК. Для того, чтобы создать многопоточную реализацию, необходимо задействовать библиотеку <omp.h>, которая позволит использовать pragma и распараллелить цикл между заданным числом потоков.

Для этого следует изменить исходную функцию и в исходный код для реализации многопоточности добавить pragma omp со значением num\_threads=8, что соответствует числу ядер на ПК (рисунок 30). При однопоточном исследовании, а также при запуске аппаратной реализации эту pragma и библиотеку следует закомментировать.



*Рис.30. Исходный код многопоточной функции kp\_502\_7*

Для компиляции теста был установлен компилятор GCC версии 13.2.0. После этого была выполнена компиляции на одном ядре и на 8 для четырёх значений N – 8192, 16384, 32768, 65536. По результатам компиляции были получены исполняемы exe файлы (рисунок 31).



*Рис.31. Выполнение компиляции теста*

Результаты запуска для одного потока.

|  |  |
| --- | --- |
| **N** | **Медиана, нс** |
| 8192 | 148272 |
| 16384 | 674271 |
| 32768 | 1071027 |
| 65536 | 2169928 |

Результаты запуска для 6 потоков.

|  |  |
| --- | --- |
| **N** | **Медиана, нс** |
| 8192 | 82140 |
| 16384 | 146008 |
| 32768 | 262890 |
| 65536 | 461875 |

7.2. Измерение времени выполнения на аппаратной реализации

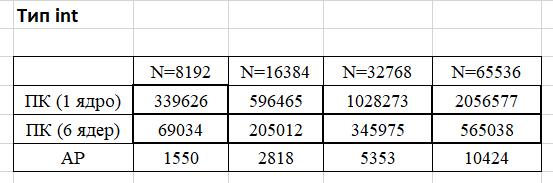
Для измерения времени используется аппаратная реализация sol3\_1. Данная реализация имеет следующие директивы: unroll factor=32, array\_partition cyclic, factor=32, конвейеризация включена. Решение является оптимальным для типа данных int.

Синтез решения sol3\_1был осуществлён для четырёх значений N – 8192, 16384, 32768, 65536. Время выполнения = II \* Estimated period для каждого случая:

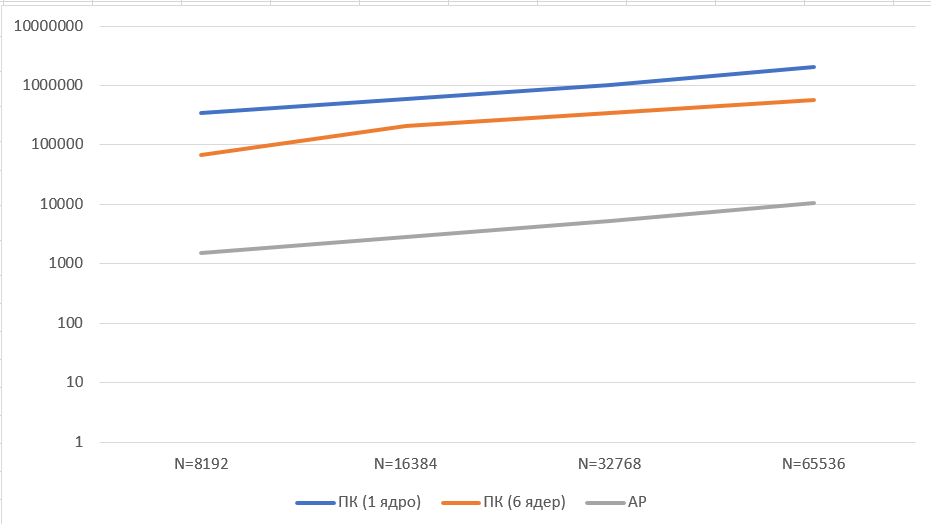
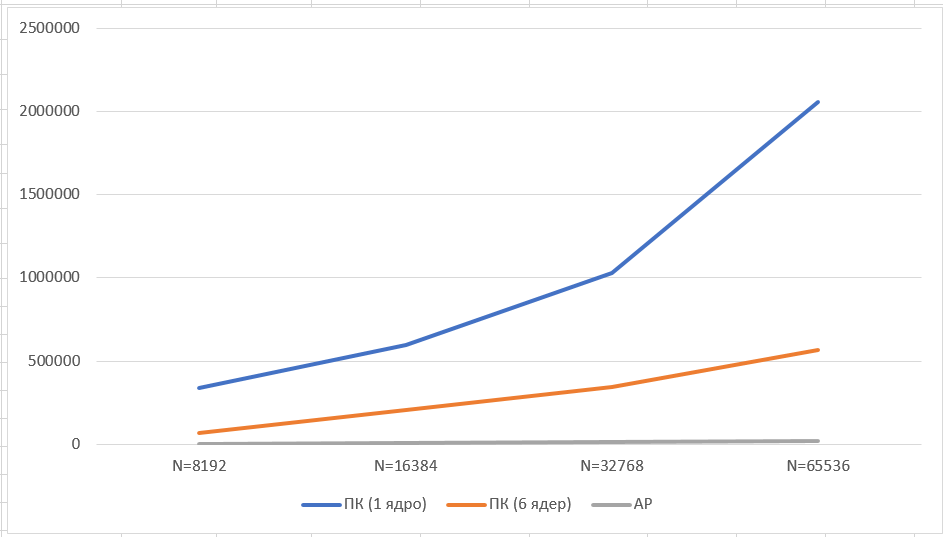
|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **II, такт** | **Estimated period, нс** | **Время выполнения, нс** |
| 8192 | 369 | 5.000 | 1845 |
| 16384 | 611 | 5.000 | 3055 |
| 32768 | 1142 | 5.000 | 5710 |
| 65536 | 2155 | 5.000 | 10775 |

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 32–33.



*Рис.32. Электронная таблица для ПК (1 и 8 ядер) и аппаратной реализации (тип int)*



*Рис.33. График для ПК (1 и 6 ядер) и аппаратной реализации (тип int)*

На графики видно, что аппаратная реализация значительно выигрывает по производительности у обеих реализаций на ПК за счёт распараллеливания и конвейеризации. При этом многопоточная реализации на ПК даже на малых значениях N выигрывает однопоточную реализацию.

8. Исследовании производительности решения для разных типов данных

Необходимо выполнить аналогичные измерения времени на ПК и аппаратной реализации для других типов данных.

8.1. Тип данных long long

Результаты запуска для одного потока.

|  |  |
| --- | --- |
| **N** | **Медиана, нс** |
| 8192 | 351330 |
| 16384 | 836475 |
| 32768 | 1664620 |
| 65536 | 3234606 |

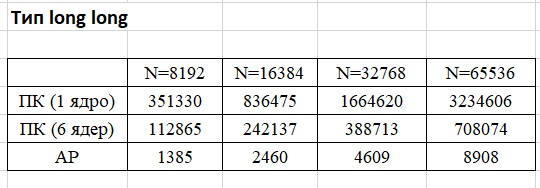
Результаты запуска для 6 потоков.

|  |  |
| --- | --- |
| **N** | **Медиана, нс** |
| 8192 | 112865 |
| 16384 | 242137 |
| 32768 | 388713 |
| 65536 | 708074 |

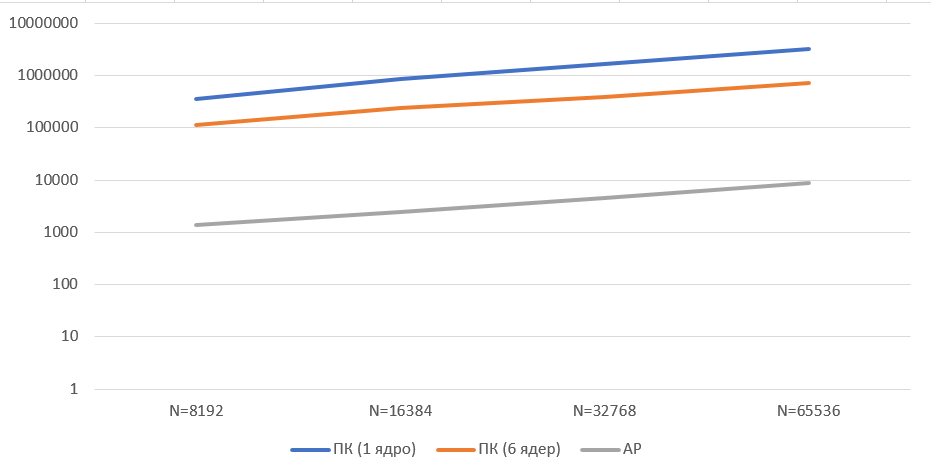
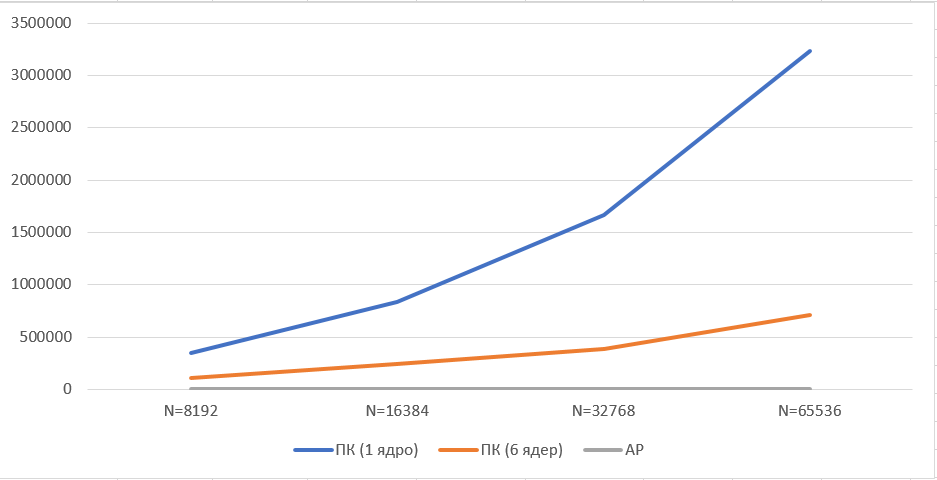
Результаты синтеза аппаратного решения sol3\_4. Данная реализация имеет следующие директивы: unroll factor=32, array\_partition cyclic, factor=32, конвейеризация включена. Решение является оптимальным для типа данных long long.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **II, такт** | **Estimated period, нс** | **Время выполнения, нс** |
| 8192 | 330 | 4,198 | 1385 |
| 16384 | 586 | 4,198 | 2460 |
| 32768 | 1098 | 4,198 | 4609 |
| 65536 | 2122 | 4,198 | 8908 |

Сравнение результатов long long (рисунки 34–35).



*Рис.34. Электронная таблица для ПК (1 и 6 ядер) и аппаратной реализации (тип long long)*



*Рис.35. График для ПК (1 и 6 ядер) и аппаратной реализации (тип long long)*

На графики видно, что аппаратная реализация значительно выигрывает по производительности у обеих реализаций на ПК за счёт распараллеливания и конвейеризации. При этом многопоточная реализации на ПК даже на малых значениях N выигрывает однопоточную реализацию.

8.2. Тип данных float

Результаты запуска для одного потока.

|  |  |
| --- | --- |
| **N** | **Медиана, нс** |
| 8192 | 117824 |
| 16384 | 268330 |
| 32768 | 403475 |
| 65536 | 1013603 |

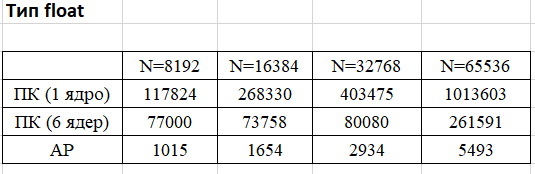
Результаты запуска для 6 потоков.

|  |  |
| --- | --- |
| **N** | **Медиана, нс** |
| 8192 | 77000 |
| 16384 | 73758 |
| 32768 | 80080 |
| 65536 | 261591 |

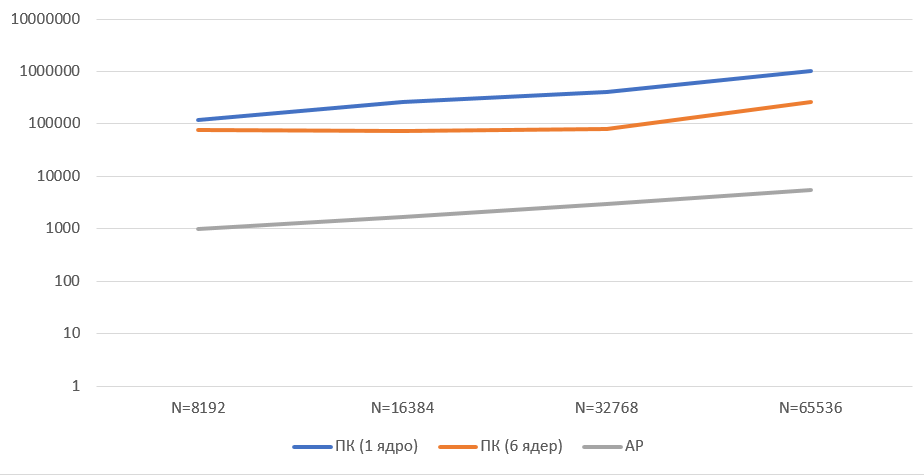
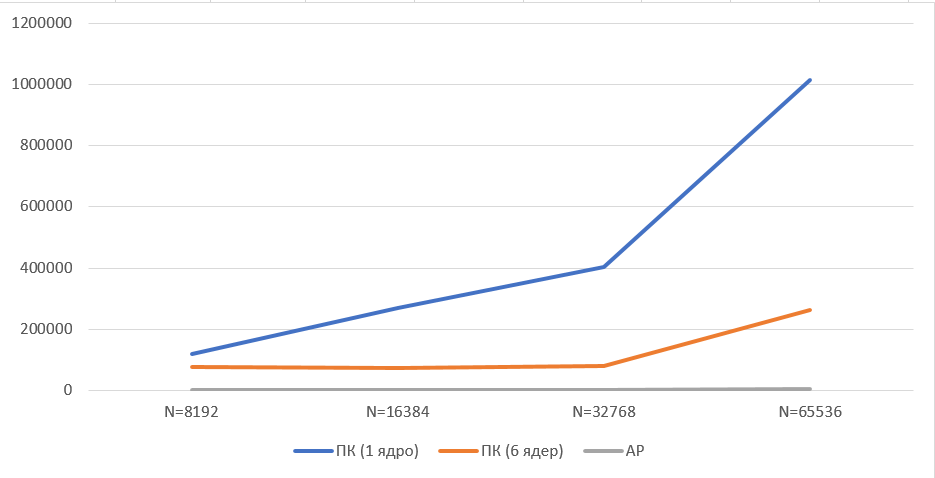
Результаты синтеза аппаратного решения sol3\_3. Данная реализация имеет следующие директивы: unroll factor=64, array\_partition cyclic, factor=64, конвейеризация включена. Решение является оптимальным для типа данных float.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **II, такт** | **Estimated period, нс** | **Время выполнения, нс** |
| 8192 | 203 | 4,998 | 1015 |
| 16384 | 331 | 4,998 | 1654 |
| 32768 | 587 | 4,998 | 2934 |
| 65536 | 1099 | 4,998 | 5493 |

Сравнение результатов float (рисунки 36–37).



*Рис.36. Электронная таблица для ПК (1 и 6 ядер) и аппаратной реализации (тип float)*



*Рис.37. График для ПК (1 и 6 ядер) и аппаратной реализации (тип float)*

На графики видно, что аппаратная реализация значительно выигрывает по производительности у обеих реализаций на ПК за счёт распараллеливания и конвейеризации. При этом многопоточная реализации на ПК даже на малых значениях N выигрывает однопоточную реализацию.

8.3. Тип данных double

Результаты запуска для одного потока.

|  |  |
| --- | --- |
| **N** | **Медиана, нс** |
| 8192 | 204758 |
| 16384 | 538837 |
| 32768 | 1031040 |
| 65536 | 2040575 |

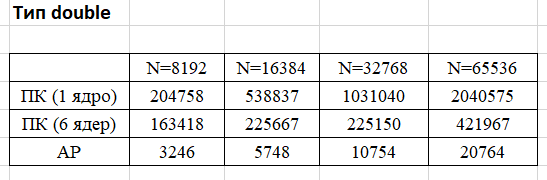
Результаты запуска для 6 потоков.

|  |  |
| --- | --- |
| **N** | **Медиана, нс** |
| 8192 | 163418 |
| 16384 | 225667 |
| 32768 | 225150 |
| 65536 | 421967 |

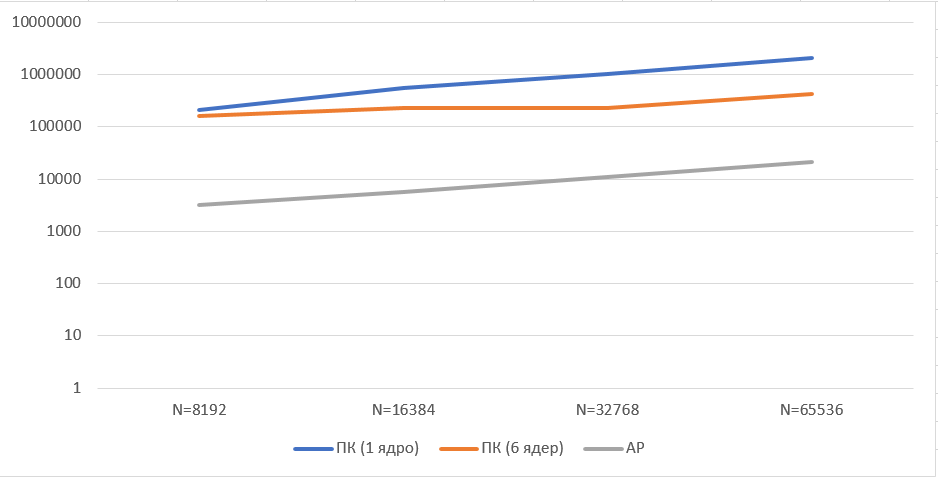
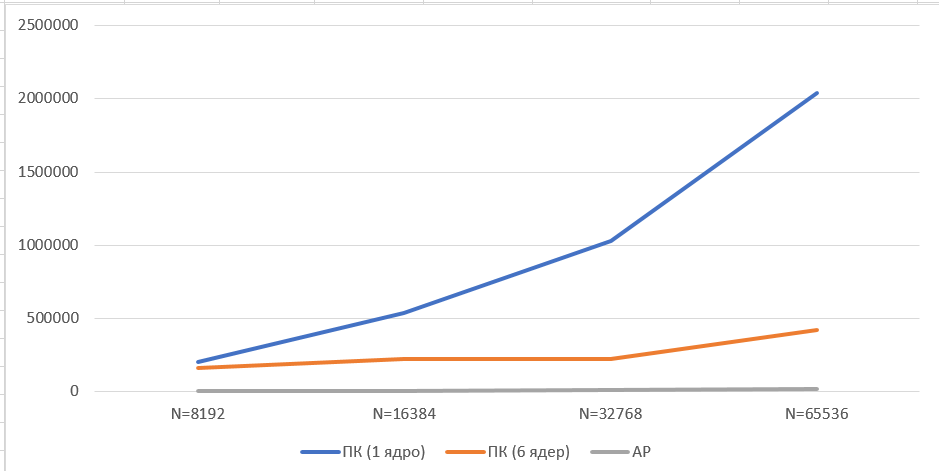
Результаты синтеза аппаратного решения sol3\_2. Данная реализация имеет следующие директивы: unroll factor=16, array\_partition cyclic, factor=16, конвейеризация включена. Решение является оптимальным для типа данных double.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **II, такт** | **Estimated period, нс** | **Время выполнения, нс** |
| 8192 | 664 | 4,888 | 3246 |
| 16384 | 1176 | 4,888 | 5748 |
| 32768 | 2200 | 4,888 | 10754 |
| 65536 | 4248 | 4,888 | 20764 |

Сравнение результатов float (рисунки 38–39).



*Рис.38. Электронная таблица для ПК (1 и 6 ядер) и аппаратной реализации (тип double)*



*Рис.39. График для ПК (1 и 6 ядер) и аппаратной реализации (тип double)*

На графике можно видеть, что при типе данных double аппаратная реализация так же, как и на других типах данных выигрывает, но производительность ухудшается быстрее. При этом различие между многопоточной и однопоточной реализацией не очень большое, но с ростом N разрыв увеличивается.

9. Вывод

В результате выполнения данной курсовой работы были закреплены знания по оптимизации аппаратных реализаций программ. Для этого на языке C++ была написана программа, для поиска элементов массива r, которые равны значению многочлена.

На этой программе было выполнено исследование вариантов аппаратной реализации с разными периодами тактового сигнала. Наилучшее значение с точки зрения производительности и аппаратных затрат было решение с периодом 6 нс.

На основе данного решения была выполнена оптимизации функции. Для этого использовались директивы Unroll, Array\_Partition, Array\_Reshape, Pipeline и RAM. Для N=8 лучшие результаты показало решение с unroll factor=8 и конвейеризацией. Также был выявлен максимальный unroll factor, который позволял получить максимальную производительность для 4 типов данных. Этот unroll factor использовался при замере времени.

Далее был выполнен анализа временных затрат при программной и аппаратной реализациях функции. Были исследованы разные размеры массивов и типы данных. В результате было выявлено, что для всех четырёх рассматриваемых типов аппаратная реализация является лучшей.

Приложение 1. Описание содержимого архива

Архив проекта содержит следующие элементы.

Папка doc – папка, которая содержит файлы, предназначенные для описания проекта. Здесь находится kp\_502\_7.xlsx – файл с графиками и таблицами, построенными на основе анализа решений, и ПРГВС\_КР\_21502\_Шеметов.docx – отчет по курсовой работе в редактируемом формате.

Папка kp\_502\_7 – папка проекта. Содержит все созданные с помощью tcl скриптов решения (sol1-sol1\_2 и sol2-sol3\_4).

Папка source – папка, которая содержит все исходные коды. Файл с исходным кодом функции kp\_502\_7.cpp, заголовочный файл kp\_502\_7.h с типами данных массивов и размерностью, тестовый файл для аппаратной реализации kp\_502\_7\_test.cpp, тестовый файл для реализации на ПК kp\_502\_7\_test\_time.cpp. Также source содержит внутреннюю папки с exe, в которых находятся исполняемые файлы для всех вариаций решений в формате exe. Они распределены по папкам в соответствии с исследуемым типом данных: int, long, float, double. Пример исполняемого файла: int\_1\_8192.exe – тип данных int, число потоков 1, размер N=8192.

Также архив содержит 2 tcl скрипта: kp\_502\_7.tcl и kp\_502\_7\_2.tcl. Скрипт kp\_502\_7.tcl предназначен для создания и исследования решений с разным периодом (решения sol1-sol1\_2). Скрипт kp\_502\_7\_2.tcl предназначен для создания и исследования оптимизирующих исходную функцию (решения sol2-sol3\_5).