СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

БПФ, FFT – быстрое преобразование Фурье

ДПФ – дискретное преобразование Фурье

ПЛИС, FPGA - программируемая логическая интегральная схема

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Сопроцессор, coprocessor – устройство, предназначенное для расширения возможностей выполнения операций.

Оптимизация – процесс или способ модификации системы для улучшения её эффективности по заданному параметру.

Производительность – точность и скорость, с которой система выполняет свои задачи в определенных условиях.

Конвейер – способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности.

Аппаратный модуль – компонент аппаратуры, предназначенный для выполнения определенных функций или операций, является отдельным физическим блоком или подсистемой, который может быть встроен в общую систему или использоваться как отдельное устройство.

Процессор общего назначения – микропроцессор, спроектированный для выполнения широкого спектра задач и операций.

ВВЕДЕНИЕ

В современном мире обработка сигналов играет важную роль во многих областях, таких как связь, медицина, аудио- и видеообработка, астрономия, радарные системы, и другие. В связи с этим существует постоянная потребность в разработке эффективных систем обработки сигналов, способных обеспечить высокую производительность, точность вычислений и гибкость в адаптации к различным задачам.

Одним из ключевых компонентов в системах обработки сигналов является процессор. Процессоры общего назначения, такие как процессоры семейства RISC-V, предоставляют широкий набор инструкций и возможностей для обработки данных. Однако, для эффективной обработки сигналов могут быть полезны специализированные аппаратные модули, способные обеспечить ускоренные вычисления для определенных алгоритмов.

Целью данной работы является проектирование аппаратно-программной системы обработки сигналов на основе процессора с архитектурой RISC-V. В основе системы лежит использование процессора RISC-V в сочетании с дополнительными аппаратными модулями, специализированными для быстрого преобразования Фурье – одного из фундаментальных алгоритмов обработки сигналов. В задачи работы входят:

1. Разработать программную реализацию алгоритма быстрого преобразования Фурье для архитектуры RISC-V.

2. Реализовать аппаратное ускорение алгоритма быстрого преобразования Фурье на основе интерфейсов системной шины и/или интерфейса расширения систем команд.

3. Выполнить сравнительный анализ полученных реализаций по критериям производительности, площади и энергопотребления, выбрать наилучшие варианты реализации.

С появления алгоритма быстрого преобразования Фурье было создано большое количество материалов по оптимизации его работы. Данная работа опирается на следующие материалы:

Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation, 19(90), 297-301. – Эта работа представляет оригинальную статью, в которой алгоритм БПФ был впервые описан и предложен [1].

Brigham, E. O. (1974). The fast Fourier transform. Prentice-Hall. – Эта книга представляет введение в теорию и применение БПФ. В ней рассматриваются различные алгоритмы БПФ и их свойства [2].

Oppenheim, A. V., & Schafer, R. W. (2010). Discrete-time signal processing. Pearson Education. – В этой книге представлены основы обработки сигналов, включая преобразование Фурье и БПФ, с акцентом на их применение в цифровой обработке сигналов [3].

Rader, C. M. (1968). Discrete Fourier transforms when the number of data samples is prime. Proceedings of the IEEE, 56(6), 1107-1108. – В этой статье представлен алгоритм для вычисления БПФ, когда число данных является простым числом [4].

Van Loan, C. F. (1992). Computational frameworks for the fast Fourier transform. Society for Industrial and Applied Mathematics. – В этой книге представлены различные вычислительные методы для БПФ, включая алгоритм Кули-Тьюки и другие варианты [5].

Frigo, M., & Johnson, S. G. (1998). FFTW: An adaptive software architecture for the FFT. Proceedings of the IEEE, 93(2), 216-231. – Авторы представляют библиотеку FFTW (Fastest Fourier Transform in the West), которая предоставляет оптимизированную реализацию БПФ с использованием различных методов, включая различные варианты алгоритма Кули-Тьюки [6].

Chen L. et al. (2007) Optimizing the fast fourier transform on a multi-core architecture IEEE International Parallel and Distributed Processing Symposium – Авторы представляют оптимизированный алгоритм БПФ, который использует векторизацию и параллелизм для эффективного выполнения последовательности преобразований на многоядерных системах [7].

Тема ускорения БПФ аппаратными модулями рассматривалась мало. Существует статья Zheng Z., Zhu X., Qian H. Design and Implementation of Arbitrary Point FFT Based on RISC-V SoC, описывающая работу алгоритма БПФ на процессоре RISC-V с уменьшенным набором команд [8]. И работа Н. Н. Иванова, Н. А. Галанина, Д. В. Моисеев Особенности реализации алгоритма БПФ на ПЛИС типа FPGA в которой рассматривается реализация схемы, способной обрабатывать данные без использования процессора общего назначения [9].

Для работы с возможностями расширения процессора RISC-V использовались материалы ActiveCore Laboratory work manual Using Sigma MCU in FPGA designs [10] и Using UDM bus transactor in FPGA designs авторства А. Антонова [11]. А также The RISC-V Instruction Set Manual Volume I: Unprivileged ISA для справки по доступным командам архитектуры [12].

Ускорение вычислений и повышение их энергоэффективности позволит снизить затраты на обработку больших объёмов данных и увеличит время работы устройств, работающих от батареи. Полученные в ходе работы данные позволяют сделать выводы о сложностях и ограничениях в разработке и скорости работы аппаратных ускорителей вычислений.

Глава 1. Обзор предметной области

1.1 Алгоритмы преобразования Фурье

Быстрое преобразование Фурье (БПФ) — это алгоритм, который позволяет вычислять дискретное преобразование Фурье (ДПФ) с меньшей сложностью для последовательности значений, такой как временной ряд или сигнал. ДПФ преобразует последовательность значений в частотное представление, разложив ее на составляющие синусоидальные и косинусоидальные компоненты разных частот.

Дискретное преобразование Фурье вычисляется с помощью алгоритма прямого преобразования Фурье, который имеет сложность O(N^2), где N - количество значений во входной последовательности. Однако быстрое преобразование Фурье является алгоритмом, который может вычислить ДПФ значительно быстрее, с асимптотической сложностью O(N log N).

Основная идея БПФ заключается в использовании свойств периодичности и симметрии преобразования Фурье. Алгоритм разбивает входную последовательность на подпоследовательности более короткой длины, а затем комбинирует результаты их преобразования, чтобы получить окончательный результат.

Наиболее известный алгоритм БПФ - алгоритм Кули-Тьюки, который использует подход “разделяй и властвуй”. Алгоритм рекурсивно разделяет входную последовательность пополам до достижения базового случая, когда последовательность имеет длину 1. Затем результаты преобразования комбинируются, чтобы получить итоговый результат. Алгоритм Кули-Тьюки использует симметричность и периодичность ДПФ, что позволяет уменьшить количество операций, необходимых для вычисления преобразования. Однако это накладывает свои ограничения - алгоритм в чистом виде принимает только такие входные последовательности, длина которых равна степени двойки. В противном случае потребуется дополнительная обработка и подготовка

данных, которая может сказаться на точности результата или модификации алгоритма.

В результате БПФ позволяет получить спектральное представление сигнала, которое отображает важные характеристики сигнала в частотной области. Это представление может быть использовано для анализа, фильтрации, сжатия и многих других операций над сигналами, что делает БПФ одним из важных инструментов в обработке сигналов и связанных областях.

Алгоритм Кули-Тьюки был предложен в 1965 году и с тех пор получил множество оптимизаций и модификаций. Некоторые из них включают:

1. Рекурсивный Кули-Тьюки: Оптимизация заключается в применении рекурсивного подхода, чтобы уменьшить число операций. Алгоритм разделяет последовательность на две части, вычисляет их БПФ рекурсивно и затем комбинирует результаты.
2. Порядок бит-инверсии: БПФ требует, чтобы входные данные были упорядочены в определенном порядке. Оптимизация по порядку бит-инверсии в алгоритме Кули-Тьюки позволяет избежать перестановок элементов и упрощает процесс комбинирования результатов.
3. Таблицы предварительных вычислений: для ускорения вычислений, в алгоритме Кули-Тьюки можно использовать таблицы предварительно вычисленных значений для определенных углов, что позволяет избежать повторных вычислений и ускоряет процесс.
4. Оптимизация для малых размеров: когда размер входной последовательности мал, например, меньше 16, можно использовать прямые методы вычисления преобразования Фурье, которые могут быть более эффективными, чем БПФ.
5. Векторизация и параллелизм: с учетом современных архитектур процессоров, БПФ может быть оптимизирован путем использования векторных инструкций и параллельных вычислений, что позволяет эффективно использовать мощность многопоточных систем.

Для понимания работы алгоритма и его оптимизаций следует сначала обратиться к выражению дискретного преобразования Фурье:

$$S(k) = \sum_{n=0}^{N-1} s(n) e^{-j\frac{2\pi}{N}nk}, \qquad k = 0 \dots N-1.$$

Уравнение ставит N отсчетам сигнала s(n), n=0…N-1 соответствие N комплексных спектральных отсчетов S(k), k=0…N-1. Один спектральный отсчёт вычисляется путём применения операций комплексного сложения и умножения N раз. Таким образом для вычисления N спектральных точек потребуется N^2 операций умножения и сложения, а сложность алгоритма составляет O(N^2).

Сложность алгоритма квадратично растёт от размера входного сигнала, поэтому переход к вычислению того же алгоритма дважды, но на $\frac{N}{2}$ точках даст существенный выигрыш по скорости вычисления. Число операций уменьшится в два раза, но потребуются дополнительные операции разделения и объединения входной и выходной последовательностей соответственно. Также последовательность длины $\frac{N}{2}$ может быть разделена на две последовательности длины $\frac{N}{4}$, они могут быть разделены на последовательности длины $\frac{N}{8}$ и так далее, пока последовательность может быть разделена. В этом случае последовательность длины $N = 2^L$ последовательность может быть разделена L раз. Пример разделения для последовательности длины $N = 8, L = 3$ приведён на рисунке 1.

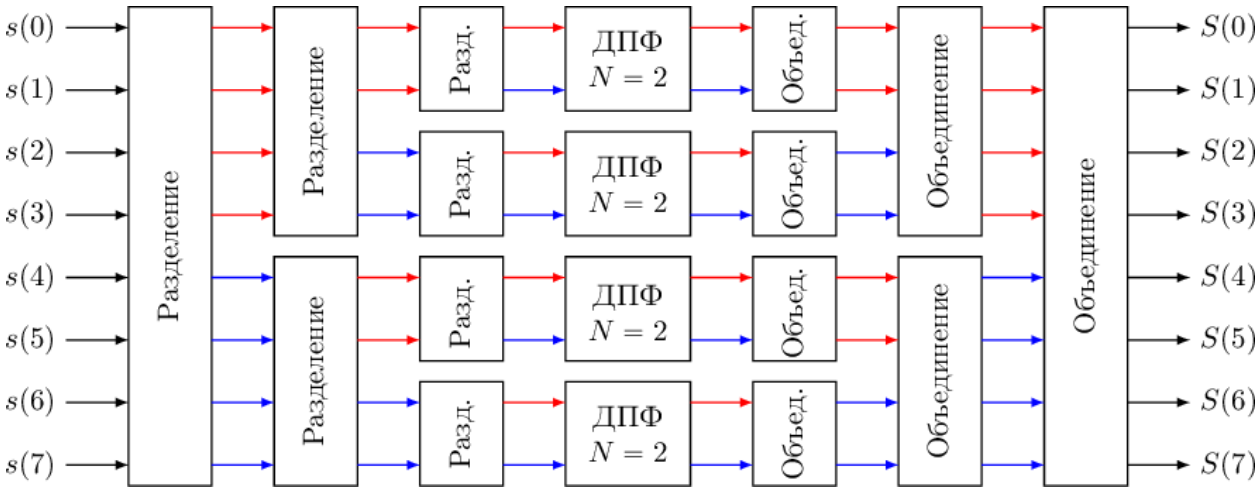

Рисунок 1 – Разделение последовательности длины 8

Алгоритмы с длиной входной последовательности $N = 2^L$ называются алгоритмами быстрого преобразования Фурье по основанию 2. Они получили широкое распространение благодаря удобству их применения.

1.1.1 Быстрое преобразование Фурье с прореживанием по времени

БПФ бывает двух видов в зависимости от особенностей применяемых операций разделения и объединения – с прореживанием по времени и с прореживанием по частоте. Рассмотрим сначала БПФ с прореживанием по времени с основанием 2. Для этого возьмём оригинальное уравнение БПФ:

$$S(k) = \sum_{n=0}^{N-1} s(n) e^{-j\frac{2\pi}{N}nk}, \; k = 0 \ldots N-1.$$

Выделим из уравнения часть, называемую поворотным коэффициентом:

$$W_N^k = e^{-j\frac{2\pi}{N}nk}.$$

Таким образом уравнение упрощается до:

$$S(k) = \sum_{n=0}^{N-1} s(n) W_N^{nk}, \qquad k = 0 \ldots N-1.$$

Для разделения входной последовательности s(n), n=0…N-1 с помощью прореживания по времени на две последовательности $s_0(m)$ и $s_1(m)$, m=0…$\frac{N}{2} - 1$ необходимо обеспечить следующие равенства: $s_0(m) = s(2 \cdot m)$ и

$s_1(m) = s(2 \cdot m + 1)$. Таким образом, чётные и нечётные элементы попадают в $s_0(m)$ и $s_1(m)$ соответственно. На рисунке 2 приведено прореживание по времени для последовательности длины $N = 8$.

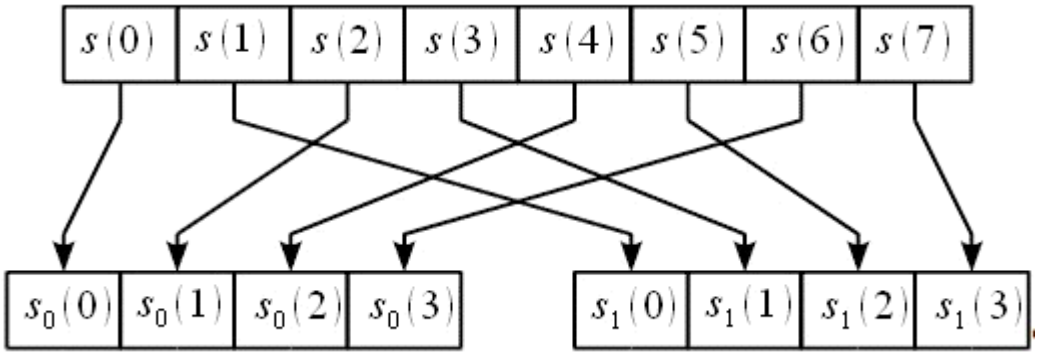

Рисунок 2 – Прореживание по времени последовательности длины 8

Таким образом уравнение дискретного преобразования Фурье принимает следующий вид:

$$S(k) = \sum_{m=0}^{\frac{N}{2}-1} s(2m) W_N^{2mk} + W_N^k \sum_{m=0}^{\frac{N}{2}-1} s(2m+1) W_N^{2mk}, \quad k = 0 \dots N-1.$$

С учётом $W_N^{2mk} = W_{\frac{N}{2}}^{mk}$ и с переходом к $k = 0 \dots \frac{N}{2} - 1$, уравнение упрощается до:

$$S(k) = S_0(k) + W_N^k S_1(k), \quad k = 0 \dots \frac{N}{2} - 1. \qquad (1)$$

При этом в конечной последовательности остаётся только $\frac{N}{2}$ элементов. Для получения элементов с индексами $k + \frac{N}{2}$ преобразования поворотных коэффициентов дают следующее уравнение:

$$S\left(k + \frac{N}{2}\right) = S_0(k) - W_N^k S_1(k), \quad k = 0 \dots \frac{N}{2} - 1. \qquad (2)$$

Уравнения (1) и (2) описывают операцию объединения прореживания по времени. На рисунке 3 представлен граф “бабочка”, показывающий данную операцию.

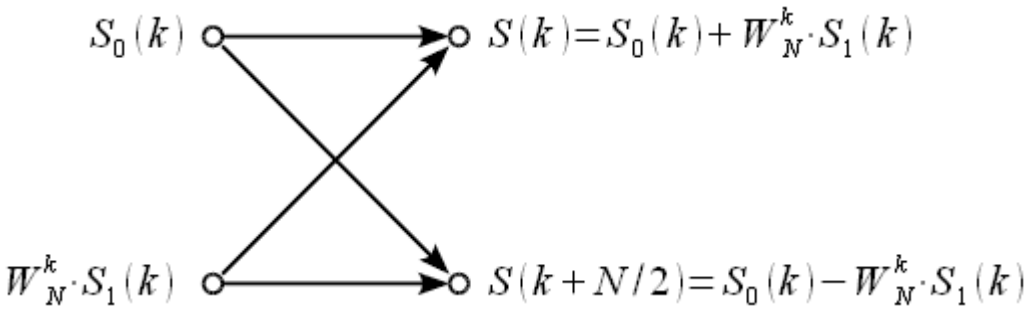

Рисунок 3 – Граф “бабочка” прореживания по времени

Подобные операции разделения-объединения также можно выполнить на $S_0(k)$ и $S_1(k)$, и получить в каждом из случаев последовательности длины $\frac{N}{4}$. Такое разделение можно производить до тех пор, пока последовательность возможно разделить пополам. При этом перестановку индексов можно упростить двоично-инверсной перестановкой. Можно заметить, что при разделении входной последовательности до пар значений, индекс входного элемента в двоичной форме переписанный побитово в обратном порядке будет равен индексу элемента после разделения.

1.1.2 Быстрое преобразование Фурье с прореживанием по частоте

Если в случае прореживания по времени разделение осуществлялось по чётности-нечётности элементов входной последовательности, прореживание по частоте разделяет входной сигнал на первую и вторую половину:

$$\mathrm{s}_0(\mathrm{m}) = \mathrm{s}(\mathrm{m}),\ \mathrm{s}_1(\mathrm{m}) = \mathrm{s}\left(\mathrm{m} + \frac{N}{2}\right),\ m = 0 \ldots \frac{N}{2} - 1.$$

В таком случае дискретное преобразование Фурье принимает вид:

$$S(k) = \sum_{m=0}^{\frac{N}{2}-1} \left[s(m) W_N^{mk} + s\left(m + \frac{N}{2}\right) W_N^{\left(m+\frac{N}{2}\right)k} \right]$$

$$= \sum_{m=0}^{\frac{N}{2}-1} \left[s(m) + s\left(m + \frac{N}{2}\right) W_N^{\frac{N}{2}k} \right] W_N^{mk}, \qquad k = 0 \ldots N-1.$$

Рассмотрим выражение для отсчётов с чётными индексами:

$$S(2p) = \sum_{m=0}^{\frac{N}{2}-1} \left[s(m) + s\left(m + \frac{N}{2}\right) \right] W_{\frac{N}{2}}^{mp}, \qquad p = 0 \ldots \frac{N}{2} - 1.$$

А для нечётных индексов 2p + 1 выражение принимает вид:

$$S(2p+1) = \sum_{m=0}^{\frac{N}{2}-1} \left[s(m) - s\left(m + \frac{N}{2}\right) \right] W_N^m W_{\frac{N}{2}}^{mp}, \qquad p = 0 \ldots \frac{N}{2} - 1.$$

При этом сигналы $s(m) + s\left(m + \frac{N}{2}\right)$ и $s(m) - s\left(m + \frac{N}{2}\right)$ обычно обозначаются как $x_0(m)$ и $x_1(m)$ соответственно. Отличается прореживание по частоте от прореживания по времени порядком перестановок и домножением на поворотный коэффициент после, а не до вычитания. Сигналы $x_0(m)$ и $x_1(m)$ также можно рекурсивно разделять пополам, пока их длина позволяет, как и в случае с прореживанием по времени.

1.1.3 Оптимизации алгоритма

Далее будет рассматриваться алгоритм с прореживанием по времени, поскольку большой разницы между алгоритмами нет и большинство оптимизаций изложенных далее могут быть применены к обеим версиям алгоритма.

Для начала следует рассмотреть наиболее простую версию алгоритма с прореживанием по времени, выполненную с помощью рекурсии, блок-схема которого приведена на рисунке 4. В данном случае W – функция расчёта поворотного коэффициента, s(n) – входная последовательность, N – длина входной последовательности, а S – результат. Функция принимает на вход последовательность длины $N = 2^L$, и, если её длина больше 2, производит разделение входной последовательности и вызывает сама себя, для вычисления БПФ на каждой из половин, после чего объединяет результаты.

Если же длина равна 2, меньше она быть не может, поскольку длина – степень двойки, то происходит выход из рекурсии с расчётом дискретного преобразования Фурье на двух точках.

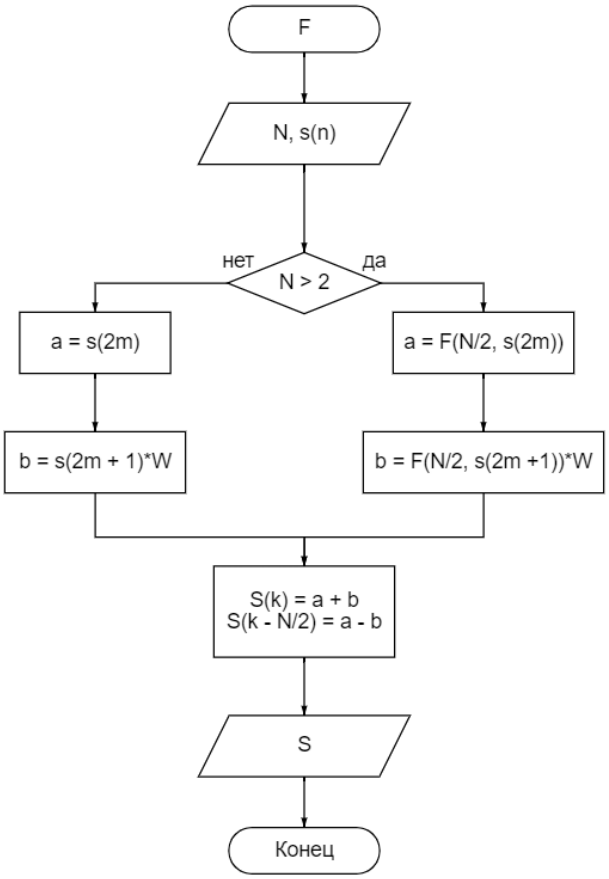

Рисунок 4 – Блок-схема алгоритма БПФ

Одна из оптимизаций не затрагивает сам алгоритм, но позволяет существенно сэкономить время потратив дополнительную память. Функция W, рассчитывающая поворотный коэффициент для определённого индекса и размера последовательности может быть заменена массивом, хранящим поворотные коэффициенты для каждого из индексов. В таком случае появляется ограничение по длине входной последовательности. Она должна быть фиксированным числом, чтобы было возможно заранее рассчитать коэффициенты.

Также мы можем избавиться от рекурсии, не занимая стек вызовами функцией самой себя. Это решение наиболее актуально для простых процессоров и контроллеров, где память для стека сильно ограничена или отсутствует вовсе. В таком случае будет очень удобно воспользоваться бит-инверсией, поскольку больше не требуется делить входную последовательность на две для вызова рекурсии, а нужно получить индексы элементов входной последовательности в результате выполнения всех перестановок.

Само по себе использование бит-инверсии так же является оптимизацией, поскольку уменьшает сложность операции разделения последовательности с O(Nlog(N)) до O(N), так как каждый входной элемент теперь переставляется на нужное место за одну операцию.

Другой возможной оптимизацией может быть более ранний выход из рекурсии, например, при N=16. Это связано с тем, что в некоторых случаях прямое вычисление дискретного преобразования Фурье может быть быстрее БПФ на малых длинах входных последовательностей, за счёт отсутствия стека вызовов и операций разделения и объединения последовательностей. Этот подход стоит упоминания, но поскольку он больше подходит для рекурсивного решения и не позволяет использовать бит-инверсию, из-за того, что не доходит до конца перестановок, в данной работе он применяться не будет.

Также существуют методы распараллеливания процесса вычисления БПФ, однако, в данной работе они так же не будут затрагиваться, поскольку работа сконцентрирована на аппаратном ускорении наиболее ресурсоёмких операций. Если в каждом ядре многоядерной системы будет доступен собственный ускоритель, можно ожидать пропорционального прироста по сравнению с одноядерной системой в задачах, не связанных непосредственно с управлением параллельной работой алгоритма.

1.2 Архитектура процессора

Для выполнения данной работы была выбрана архитектура RISC-V, поскольку она является открытой и достаточно широко распространена, что упрощает поиск материалов и пособий по работе с ней.

Разработка архитектуры RISC-V началась в 2010 году в Университете Калифорнии в Беркли под руководством профессора Дэвида Паттерсона и его коллег. Идея была в создании открытой архитектуры, которая может быть использована для обучения, исследования и коммерческой разработки процессоров.

С течением времени RISC-V приобретала все большую популярность и привлекла внимание широкого круга компаний и организаций. В настоящее время RISC-V International, некоммерческая организация, занимается поддержкой и развитием архитектуры RISC-V. Она объединяет более 100 организаций-членов, включая такие крупные компании, как Google, Nvidia, Western Digital, Alibaba, Samsung и другие. Устройства с архитектурой RISC-V активно продаются и уверенно занимают своё место на рынке микропроцессоров. Их можно свободно купить на различных интернет-площадках, и они привлекают всё больше разработчиков своей высокой производительностью по отношению к цене.

Среди основных плюсов архитектуры часто выделяют следующие:

1) Открытость: RISC-V является открытой архитектурой с открытым исходным кодом. Это означает, что любой может свободный доступ к спецификации и может использовать ее для разработки своих собственных процессоров без необходимости получать лицензию или платить лицензионные сборы. Это обеспечивает свободу выбора, стимулирует инновации и позволяет разработчикам настраивать архитектуру под свои потребности.

2) Модульность: архитектура RISC-V построена на модульном принципе, что позволяет выбирать и комбинировать различные наборы инструкций в зависимости от конкретных требований приложений. Это особенно полезно для разработки встраиваемых систем, где ресурсы могут быть ограничены и не все инструкции могут быть необходимы. Модульность также упрощает разработку и верификацию аппаратуры.
3) Простота и эффективность: архитектура RISC-V предоставляет минимальный набор основных инструкций, которые покрывают широкий спектр вычислительных задач. Более простая инструкционная схема и набор регистров делают процессоры RISC-V легкими для реализации и более энергоэффективными.
4) Поддержка для специализированных расширений: RISC-V предоставляет механизмы для добавления специализированных расширений инструкций. Это позволяет разработчикам добавлять новые инструкции, оптимизированные для конкретных приложений или областей, таких как обработка сигналов, машинное обучение или криптография. Расширения также облегчают портирование существующего программного обеспечения на процессоры RISC-V.
5) Переносимость и совместимость: RISC-V архитектура является независимой от конкретной реализации, что делает ее переносимой между различными производителями и платформами. Это означает, что программное обеспечение, разработанное для одного процессора RISC-V, может работать на других совместимых процессорах без изменений. Это упрощает разработку экосистемы и инструментария вокруг архитектуры.

Из минусов же выделяют следующие:

1) Ограниченный набор инструкций: одним из основных принципов RISC-V является простота и минимальный набор инструкций,

который иногда может быть и минусом. В некоторых случаях это может означать, что определённые сложные операции или оптимизации могут быть затруднены. Некоторые приложения или алгоритмы могут потребовать большего числа инструкций или специализированных расширений, которые не входят в базовый набор инструкций RISC-V.

2) Отсутствие стандартизированных расширений: хотя RISC-V предоставляет механизмы для добавления расширений инструкций, сами эти расширения не являются стандартизированными. Это может создать проблемы совместимости между различными реализациями процессоров RISC-V. Разработчики могут столкнуться с несовместимостью между расширениями или несоответствием между различными реализациями архитектуры.
3) Отсутствие широкой поддержки и экосистемы: в сравнении с более распространёнными архитектурами, такими как x86 или ARM, экосистема вокруг RISC-V все еще находится в стадии развития. Это может означать ограниченный выбор готовых компонентов, инструментов разработки и оптимизированного программного обеспечения. Некоторые разработчики могут столкнуться с трудностями в поддержке и оптимизации своего программного обеспечения для RISC-V.
4) Недостаток знаний и опыта: поскольку RISC-V является относительно новой архитектурой, ее использование может потребовать обучения и формирования новых привычек для разработчиков, которые раньше работали с другими архитектурами, такими как x86 или ARM. Недостаток знаний и опыта может повлиять на эффективность разработки и оптимизации систем на основе RISC-V.

Базовым набором инструкций считается RV32I, включающий 40 основных команд. Также существуют RV32E для небольших микроконтроллеров, 64-битный вариант RV64I и 128-битный RV128I. Стандартные расширения включают следующие наборы инструкций:

RV32M: Расширение для умножения и деления целых чисел (Multiply/Divide).

RV32A: Расширение для атомарных операций (Atomic).

RV32F: Расширение для операций с плавающей запятой одинарной точности (Floating-Point).

RV64F: Расширение для операций с плавающей запятой двойной точности (Floating-Point).

RV32D: Расширение для операций с плавающей запятой двойной точности (Floating-Point).

RV32C: Расширение для сжатия инструкций (Compressed).

RV32G: Сочетание расширений RV32I, RV32M и RV32A, включающее основные целочисленные инструкции, операции умножения/деления и атомарные операции.

RV64G: Сочетание расширений RV64I, RV64M и RV64A, включающее основные целочисленные инструкции, операции умножения/деления и атомарные операции.

RV32GC: Сочетание расширений RV32G и RV32C, включающее основные целочисленные инструкции, операции умножения/деления, атомарные операции и сжатие инструкций.

RISC-V также поддерживает расширения для векторных операций, такие как расширения V (Vector), которые добавляют инструкции и регистры для работы с векторами данных.

Во многом данная работа оглядывается на статью «Design and Implementation of Arbitrary Point FFT Based on RISC-V SoC», в которой используется набор инструкций RV32IM для компиляции кода, однако процессор поддерживает только 20 инструкций и не использует специальных модулей для ускорения.

Поскольку цель данной работы заключается именно в расширении функционала процессора общего назначения, было выбрано ядро с поддержкой полного набора инструкций RV32IM Aquaris из фреймворка ActiveCore. Sigma MCU, включающее ядро Aquaris, обладает интерфейсом расширений XIF, который позволяет подключать внешние модули. Так как он идентичен шине UDM, совместимые с ним модули могут быть интегрированы в это ядро и наоборот, модули для этого ядра могут быть перенесены на другие ядра с поддержкой UDM. Также в Sigma MCU присутствует интерфейс подключения сопроцессоров для дополнительных специализированных инструкций. Диаграмма устройства процессора приведена на рисунке 5.

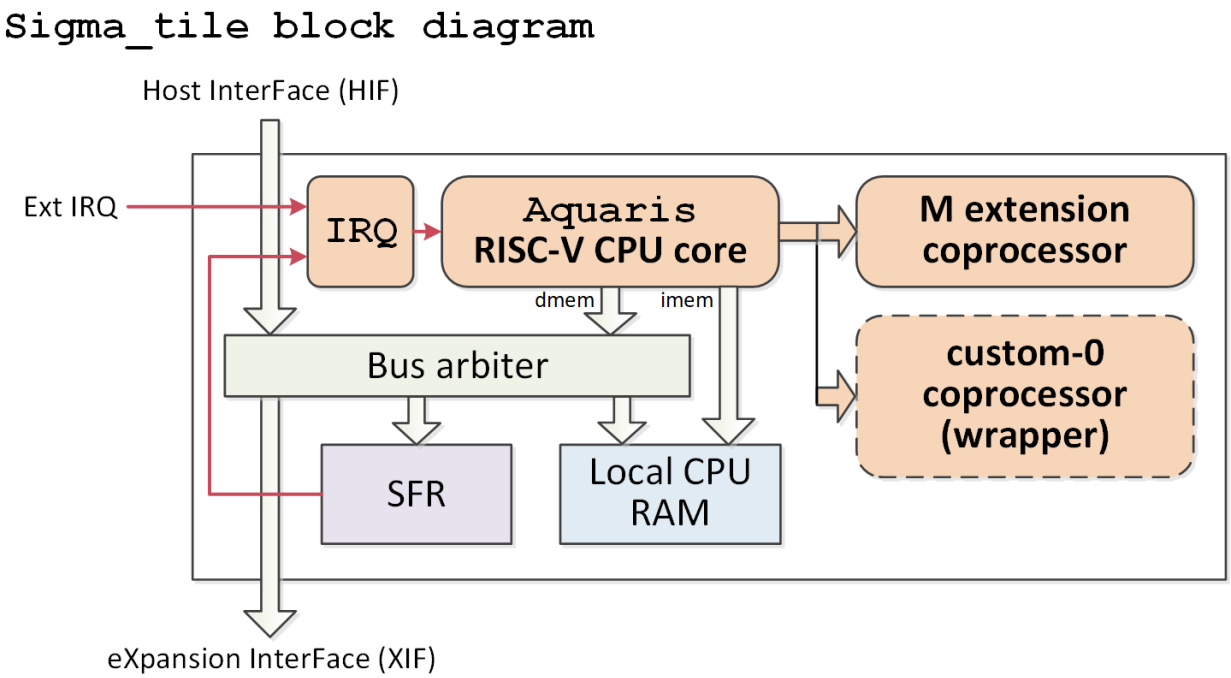

Рисунок 5 – Диаграмма устройства процессора

Подобная структура упрощает разработку расширений и позволяет применять различные подходы к их исполнению. Следует отметить, что интерфейс расширений позволяет более свободно реализовывать функционал в сравнении с добавлением сопроцессора. Благодаря меньшей привязанности к ядру процессора такие расширения могут работать с большими временными задержками по сравнению с сопроцессором, и реализовывать конвейерную обработку большей длины, поскольку не требуется жёсткой синхронизации с основным процессорным конвейером. Сопроцессоры же в свою очередь предназначены для реализации быстро работающих команд и занимают меньшую площадь на кристалле за счёт относительной простоты.

Глава 2. Разработка

2.1 Разработка программной реализации

Разработка программного решения произведена на языке C, поскольку компилятор gcc позволяет скомпилировать код в ассемблер RISC-V. Это позволяет протестировать разработанный алгоритм как на обычном компьютере, так и в симуляции процессора RISC-V.

Для начала следует рассмотреть итоговую версию алгоритма. Она использует оптимизации бит-инверсии и предрассчитанных коэффициентов, а также отказ от рекурсии. Блок схема итогового алгоритма для данной работы приведена на рисунке 6.

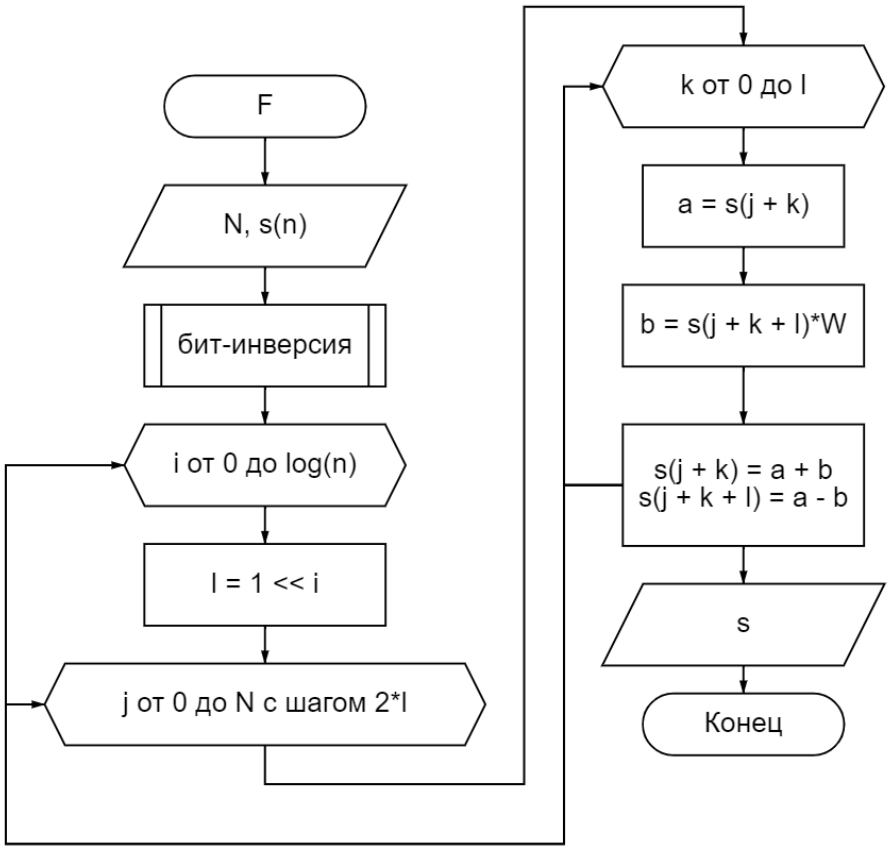

Рисунок 6 – Блок-схема алгоритма БПФ с оптимизациями

Входная последовательность та же, что и в оригинальном алгоритме, с тем же требованием, что её длина $N = 2^L$, только на этот раз она фиксировано

равна 1024, поскольку в этом алгоритме W – не функция, а массив коэффициентов, для предварительного расчёта которых нужна известная длина входной последовательности. Первым шагом идёт перестановка коэффициентов входной последовательности бит-инверсией. Таким образом можно пропустить этап разделения и перейти непосредственно к наиболее ресурсоёмкому этапу – комплексное умножение и объединение. Несмотря на то, что эта часть алгоритма вложена в три цикла, его сложность O(Nlog(N)), поскольку внешний цикл отсчитывает то, что раньше было бы глубиной рекурсии, и повторяется log(N) раз, в нашем случае 10. Число повторов следующих двух циклов равно $\frac{N}{2l} \cdot l = \frac{N}{2}$. Они нужны для формирования правильного порядка итерации по входному массиву. Также следует заметить, что в данном алгоритме массив входной последовательности является и массивом результата.

Помимо этого, в программе реализованы операции комплексного умножения, сложения и вычитания. Операция комплексного умножения реализована четырьмя операциями умножения и двумя операциями сложения/вычитания. Операция бит-инверсии реализована побитовым «и» нулевого разряда начального индекса со сдвигом результата влево, а входного значения вправо. Листинг исходного кода находится в Приложении 1.

2.2 Разработка аппаратных ускорителей

Как понятно из алгоритма, больше всего в ускорении нуждается операция комплексного умножения. Если позволяют ресурсы, можно также ускорить операции сложения и вычитания и объединить таким образом все команды в цикле в одну операцию. В чистом виде, без оптимизаций компилятора, команда комплексного умножения работает за 18 тактов, из которых 8 – загрузка операндов, 4 – умножение, 2 – сложение или вычитание и ещё 4 – сохранение результата. Операции комплексного сложения и вычитания работают за 10 тактов, из них 4 – загрузка, 2 – сложение или вычитание и 4 – сохранение результата.

2.2.1 Разработка сопроцессора

Сопроцессор в процессоре Aquaris позволяет загружать сразу два операнда и выполнять над ними действие за один такт. Это позволило выполнить часть вычислительных операций параллельно, снизив количество тактов с 10 до 4. Из-за того, что сопроцессор позволяет вернуть за один такт только одно значение, всего на получение результата уйдет 6 тактов. Теоретически, количество значений, которые получаются в результате вычислений, совпадает с количеством таков, необходимых для их вычисления, но поскольку в первые два такта результирующие значения ещё не будут готовы, потребуются два дополнительных такта на считывание. Этого можно избежать, если в эти такты считывания начать передавать значения для вычисления следующего ДПФ, однако, это потребует дополнительных вычислений новых индексов данных и их загрузки. Прирост производительности в таком случае ещё возможен, но такие изменения существенно повлияют на структуру алгоритма и усложнят использование возможных автоматических оптимизаций кода специализированным компилятором, предназначенным для использования сопроцессора. Схема

интеграции разработанного модуля с процессором приведена на рисунке 7, а микроархитектурная диаграмма на рисунке 8.

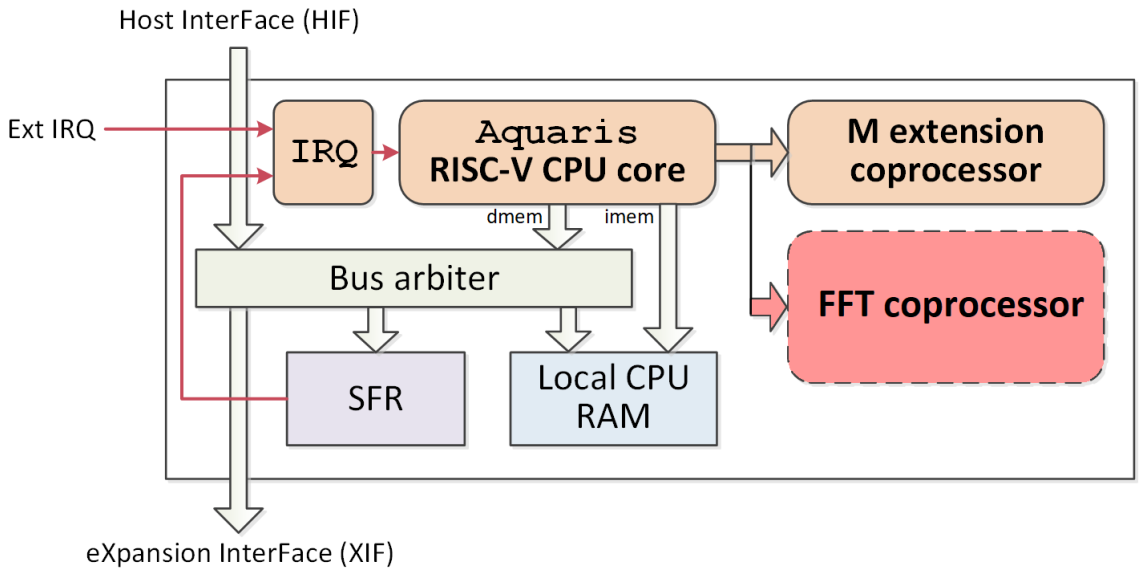

Рисунок 7 – Сопроцессор в схеме процессора

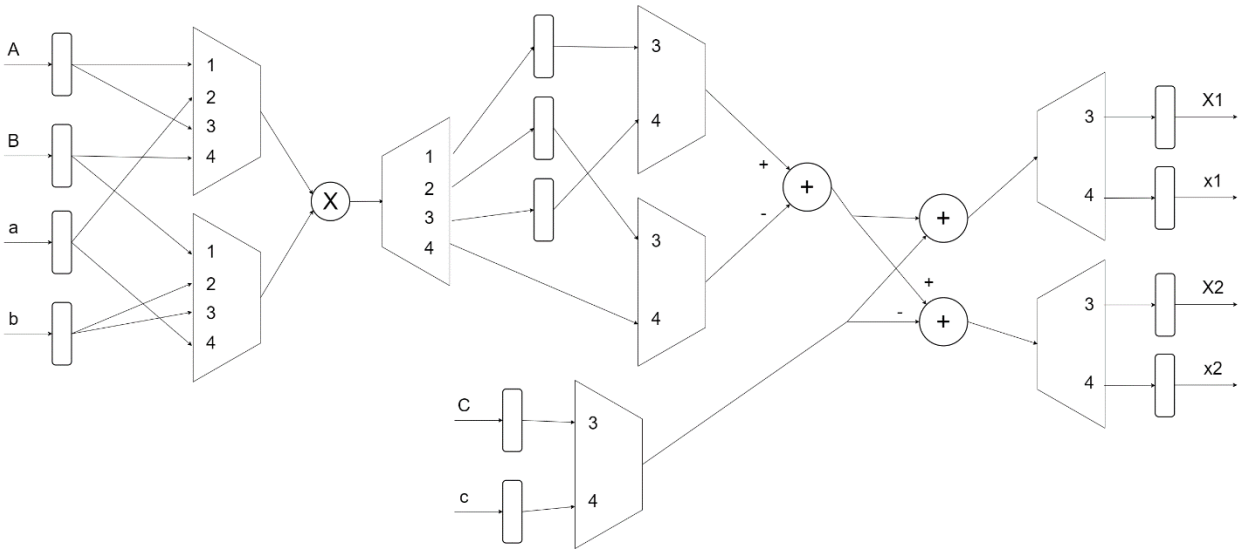

Рисунок 8 – Микроархитектурная диаграмма

Большими буквами обозначены вещественные части чисел, маленькими – мнимые. Вычисления происходят в 4 стадии:

1. Сопроцессор получает вещественные части чисел A и B и умножает их, сохраняя промежуточный результат. В данном случае одно из чисел –

поворотный коэффициент, а второе – одна из точек входной последовательности.

2. Сопроцессор получает мнимые части чисел A и B, обозначенные маленькими буквами, и умножает их, также сохраняя промежуточный результат.
3. Сопроцессор получает обе части числа C, производит умножение A и b, сохраняет результат, параллельно с этим вычисляет вещественные части обеих точек результата X1 и X2 возвращает вещественную часть X1.
4. Сопроцессор умножает B и a, возвращает вещественную часть X2 и вычисляет оставшиеся мнимые части.

Далее следуют ещё два такта чтения мнимых частей результатов из регистров сопроцессора.

Для использования сопроцессора требуется внести корректировки в программный код. Подробно этот процесс описан в руководстве фреймворка. Все вызовы функций умножения, сложения и вычитания комплексных чисел и их сохранения в программе:

```
mul(x[j + k + l], W, &product);
add(x[j + k], product, &up);
sub(x[j + k], product, &down);
x[j + k] = up;
x[j + k + l] = down;
```

Должны быть заменены на шесть вызовов функций, оборачивающих ассемблерные инструкции:

```
custom0_instr_wrapper(0x8, x[j + k + l].real, W[n*k/2/l].real);
custom0_instr_wrapper(0x9, x[j + k + l].img, W[n*k/2/l].img);
x[j + k].real = custom0_instr_wrapper(0xa, x[j + k].real, x[j + k].img);
x[j + k + l].img = custom0_instr_wrapper(0xb, x[j + k + l].real, W[n*k/2/l].img);
```

```
x[j + k].img = custom0_instr_wrapper(0x8, x[j + k + l].real, W[n*k/2/l].img);
x[j + k + l].real = custom0_instr_wrapper(0x9, x[j + k + l].real, W[n*k/2/l].img);
```

Функция custom0_instr_wrapper из руководства [10] получила только одно изменение – она теперь принимает в качестве аргумента ещё и код инструкции.

Разработанный сопроцессор ускоряет выполнение каждого внутреннего цикла, снижая количество тактов с 30 до 26, что означает прирост производительности в 13%. Такого же прироста можно было бы добиться оптимизацией вычисления индексов поворотного коэффициента и точки входной последовательности для комплексного умножения, поскольку даже с оптимизациями компилятора их вычисление использует 6 команд, а с объединением некоторых операций в сопроцессоре можно было бы ускорить этот процесс до двух команд. Реализация же последовательного выполнения и операции комплексного умножения и операций комплексного сложения и вычитания с помощью одной команды в условиях ограничений сопроцессора невозможна из-за большой задержки, даже с использованием менее экономного подхода с применением четырёх умножителей для комплексного умножения вместо трёх, и необходимости получить как минимум четыре операнда из памяти. Код сопроцессора находится в Приложении 2.

2.2.2 Разработка модуля расширения

Как уже упоминалось, модуль присоединяется к шине XIF и меньше привязан к процессору в вопросе синхронизации с основным конвейером. Это даёт возможность существенно ускорить работу специфичных алгоритмов за счёт построения аппаратной реализации без оглядки на ограничения по длине конвейера и чтение из памяти.

Этот модуль реализовывает конвейерное вычисление комплексного умножения и следующих за ним в алгоритме операций комплексного сложения и вычитания. Длина конвейера разработанного модуля равна трём, на первой стадии происходит параллельное умножение четырёх пар операндов. На второй стадии попарно складываются две результаты умножения и в итоге получаются вещественная и мнимая части результата. На третьей стадии результат параллельно складывается с третьим комплексным числом и вычитается из него. На выходе мы имеем четыре числа – вещественную и мнимую части результата преобразования Фурье на двух точках, то есть два комплексных числа. При этом важен порядок загрузки входных данных в модуль. Сначала загружается одна из входных точек и поворотный коэффициент, что занимает 4 такта, затем за два такта загружаются вещественная и мнимая части второй входной точки, тогда к моменту, когда они будут загружены, вторая стадия конвейера будет закончена и можно будет сразу приступать к третьей, и следующим же тактом считывать итоговый результат, что займёт ещё 4 такта. Схема модуля приведена на рисунке 9, а интеграция с процессором на рисунке 10.

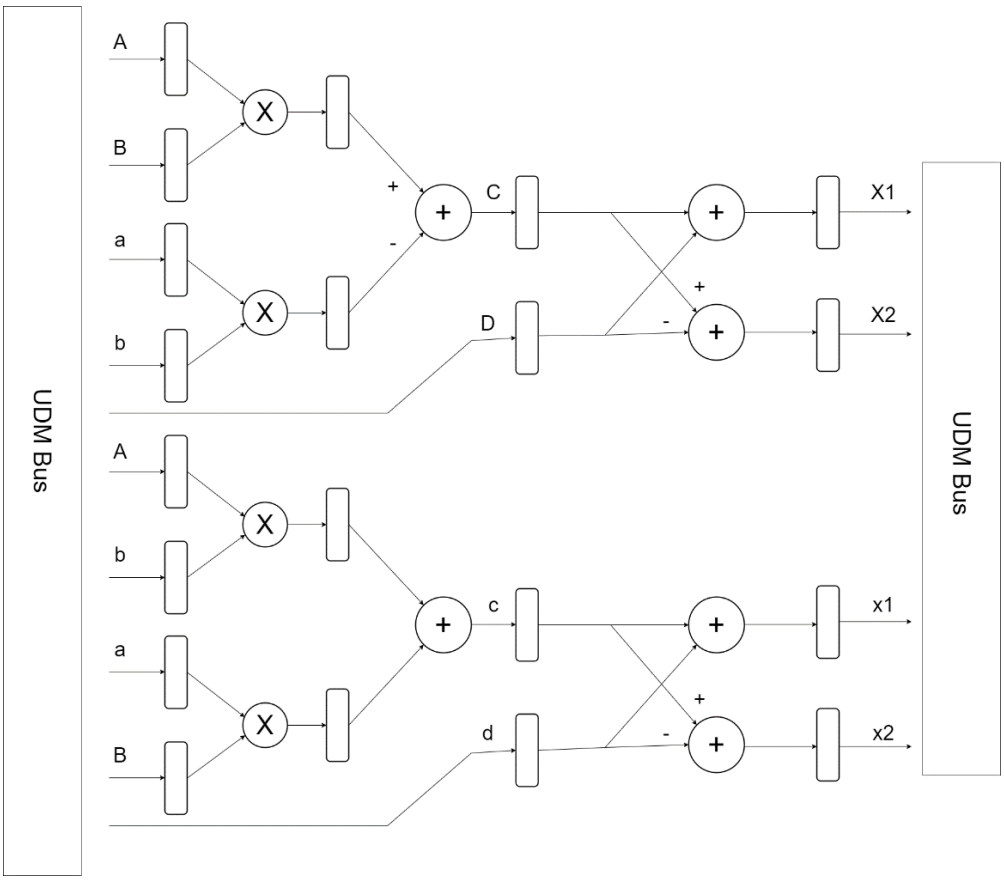

Рисунок 9 – Схема модуля ускорения на шине расширений

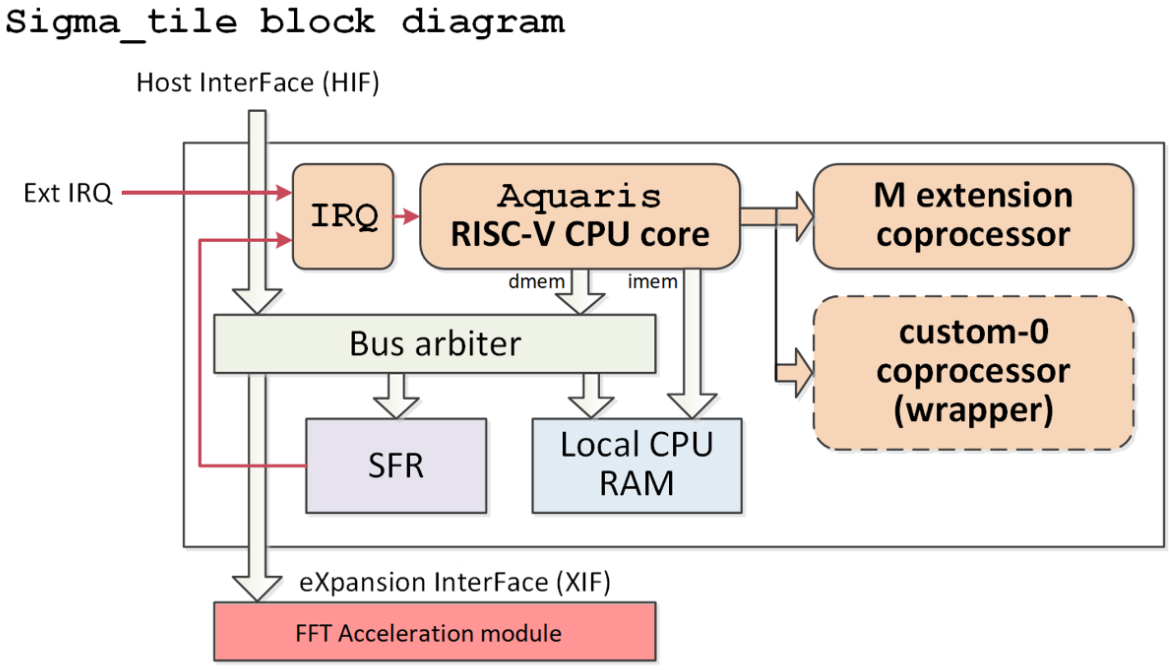

Рисунок 10 – Схема процессора с модулем расширения

Нужно заметить, что 10 тактов уходит на загрузку и выгрузку данных, при этом 7 из них не происходит никаких вычислений. Это без учёта сохранения и загрузки данных в самом процессоре, которые занимают ещё столько же. Частично обойти эти ограничения можно добавлением сопроцессора для расчёта индексов поворотных коэффициентов и точек входной последовательности. Это может быть применено вместе с данным модулем расширения на системной шине. Наибольший эффект могло бы дать расширение шины или увеличение её тактовой частоты для загрузки двух или более операндов за такт. Другим решением могло бы быть уменьшение разрядности вычислений с 32 до 16 бит, что так же увеличило бы количество передаваемых чисел, но существенно уменьшило бы точность.

Глава 3. Методика тестирования

Подробное описание подготовки для работы с процессором, настроек окружения, порядка загрузки тестовых сценариев и другого есть в руководствах по работе с ним. Здесь же будут разобраны методы подтверждения корректности работы модулей и замеров времени их работы, а также затраченных ресурсов ПЛИС. Тестирование и разработка осуществлялись в среде Vivado 2019.1, программная реализация разрабатывалась в Visual Studio Code, компиляция осуществлялась с помощью gcc и RISC-V GNU toolchain, а для удобного чтения ассемблера использовался онлайн-ресурс Compiler Explorer.

Фреймворк предоставляет для разработанных программ готовый makefile и простую интеграцию разработанной программы в тестовый сценарий. Поскольку процессор не предоставлятт обширных возможностей для вывода данных, был использован регистр LED, как для замеров производительности, так и для вывода контрольных данных. До начала вычислений быстрого преобразования Фурье, но после окончания работы бит-иверсии индексов, в этот регистр записывается некое начальное значение, после окончания вычислений оно заменяется конечным значением, что отражается на временной диаграмме и позволяет определить время работы алгоритма. Пример диаграммы приведён на рисунке 11.

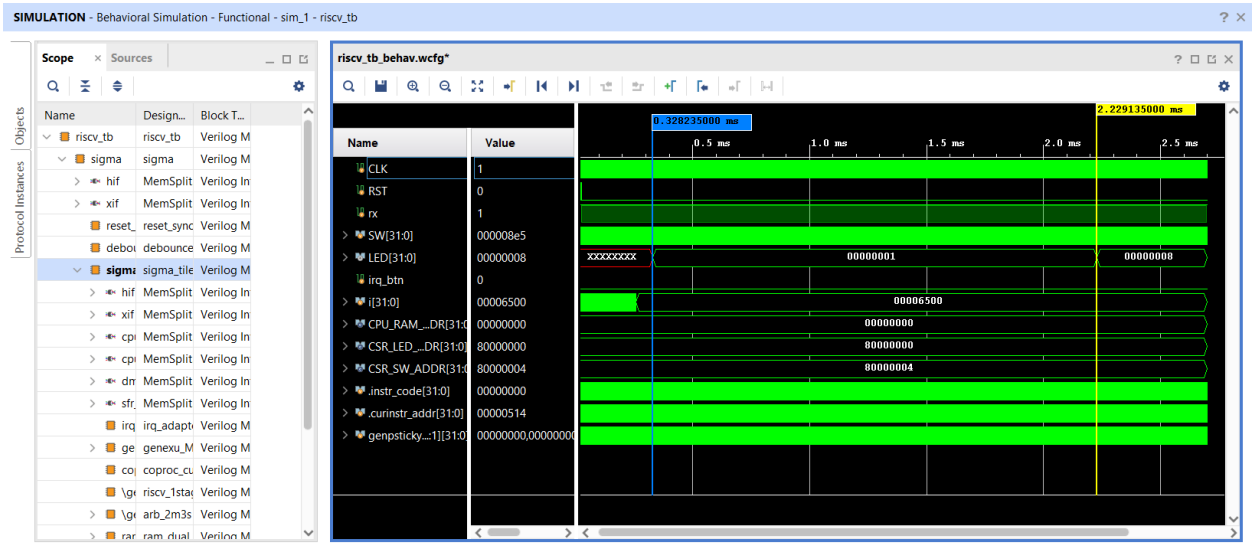

Рисунок 11 – Временная диаграмма работы программной реализации алгоритма БПФ

На диаграмме приведено время работы программы без аппаратных ускорителей. Значение LED = 1 устанавливается на отметке времени 0.328 мс, до неё идёт инициализация переменных и бит-инверсия, после же происходит основная работа алгоритма. Изменение значения на LED = 8 означает завершение работы алгоритма на отметке времени 2.229 мс, далее идёт пустой цикл, оставленный для удобства. Таким образом время работы основного цикла алгоритма БПФ на данном процессоре без применения аппаратных ускорителей составляет 1.901 мс.

Для верификации корректности реализованного алгоритма результаты его работы были выведены в консоль и преобразованы с помощью Excel в данные о спектре. Эти данные были сопоставлены с результатами, полученными с помощью Matlab на той же входной последовательности. Графические представления полученных спектров сигнала представлены на рисунках 12 и 13.

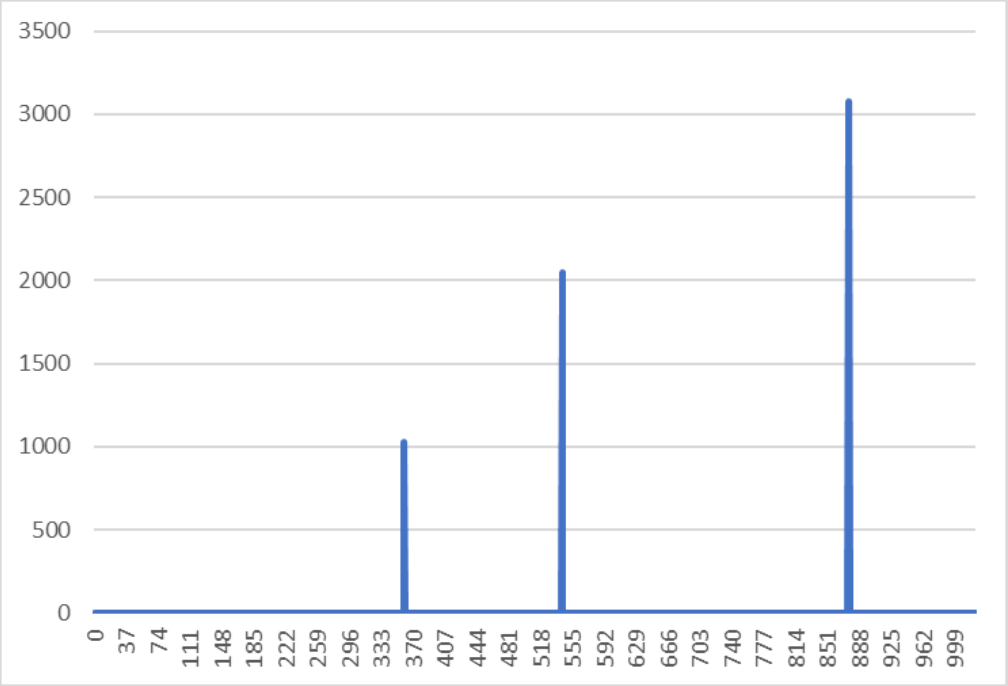

Рисунок 12 – Данные о спектре обработанные в Excel

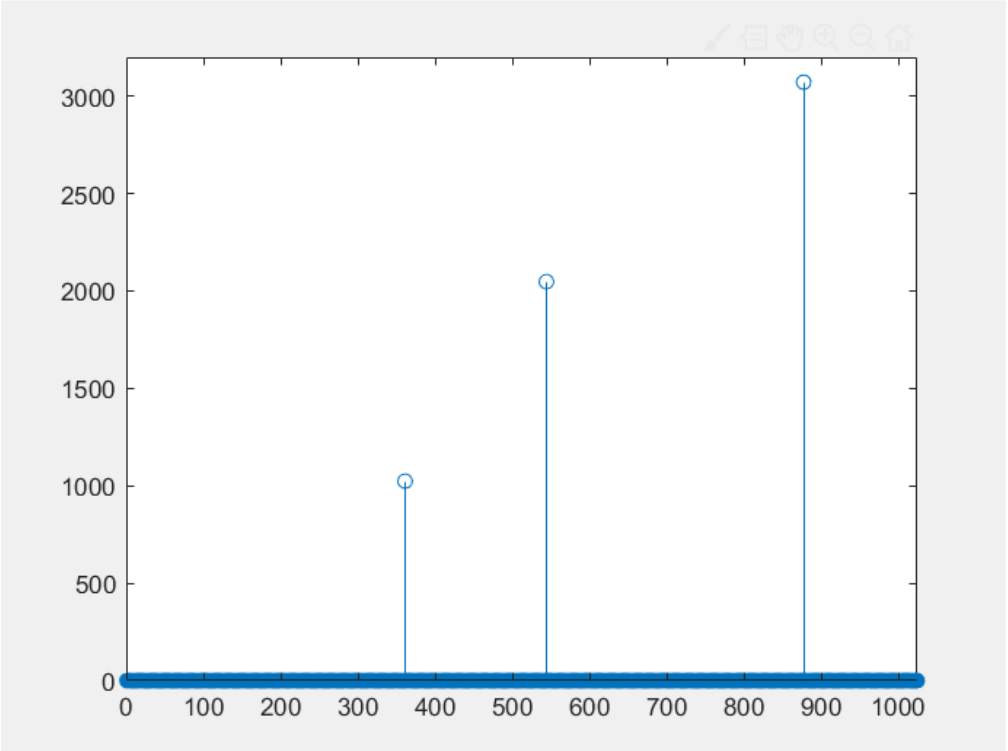

Рисунок 13 – Данные о спектре обработанные в Matlab

Как видно, спектры совпадают. Чтобы не выводить весь массив данных, для подтверждения корректности работы алгоритма на процессоре достаточно вывести контрольную сумму массива результатов. Только для её получения

желательно использовать функцию, не опирающуюся на саму операцию суммы, поскольку из-за особенностей алгоритма и его результатов, сумма будет равна нулю. В данном случае использовался XOR по всем элементам массива. Все реализации показали корректную работу.

Значения задействованных ресурсов ПЛИС брались из соответствующей таблицы в информации о проекте, после синтеза проекта. В результатах рассматриваются только отличающиеся значения, в том числе и касательно энергопотребления. Пример для программной реализации на рисунке 14.

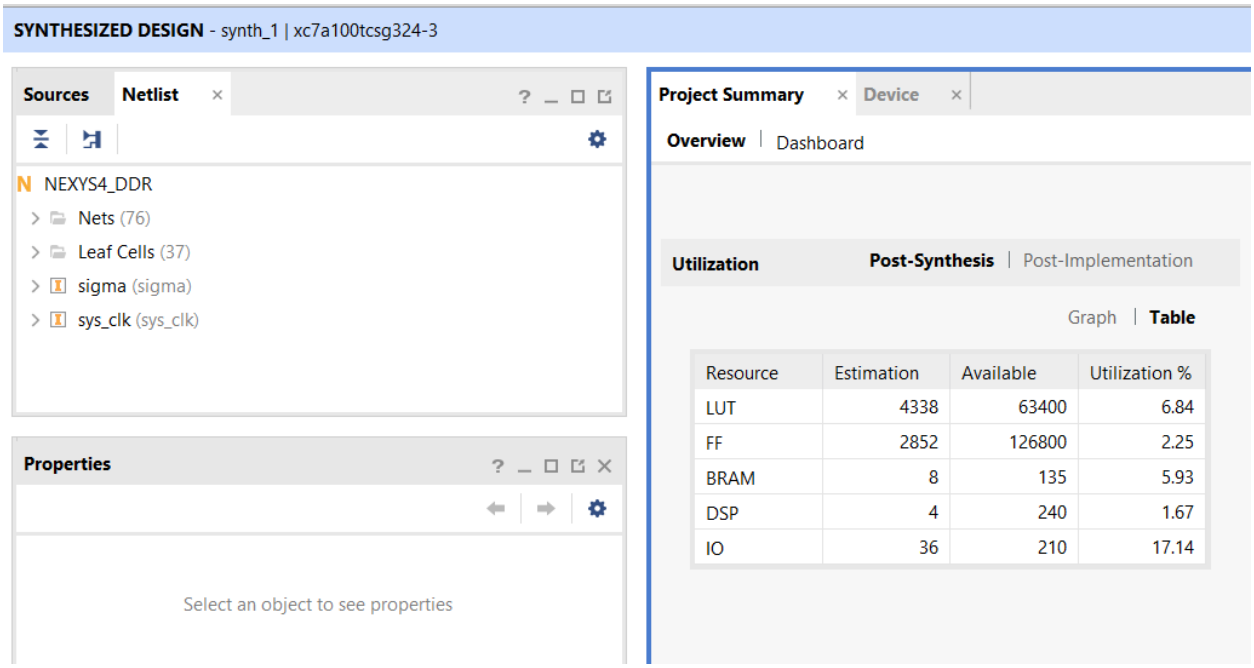

Рисунок 14 – Данные об использовании ресурсов ПЛИС

Глава 4. Анализ результатов

4.1 Время работы

Время работы программной реализации уже было показано на рисунке 11. На рисунках 15 и 16 приведено время работы реализации с сопроцессором и модулем на шине соответственно.

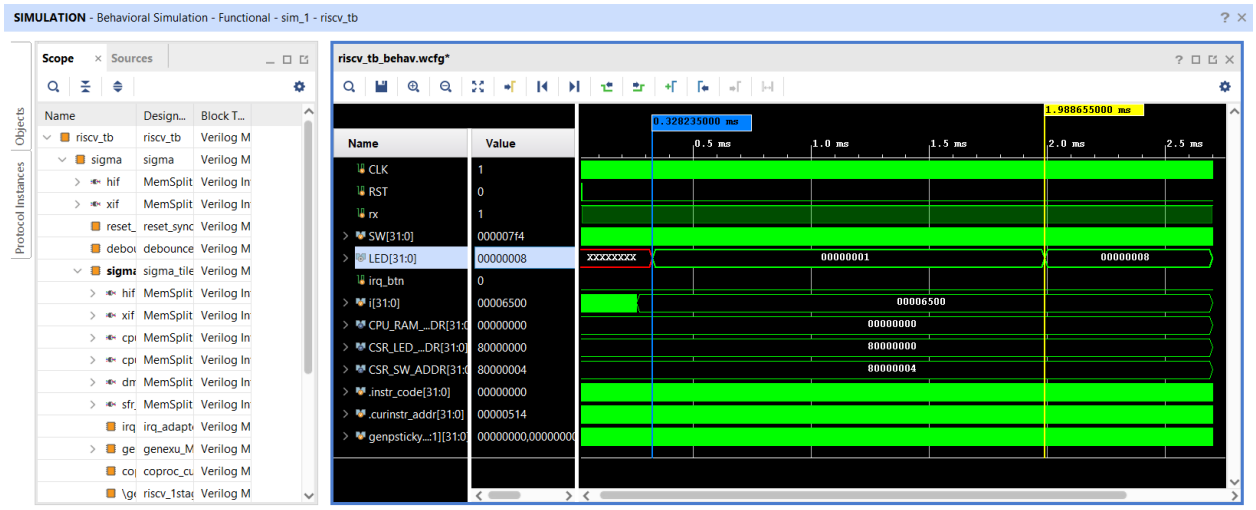

Рисунок 15 – Временная диаграмма работы сопроцессора

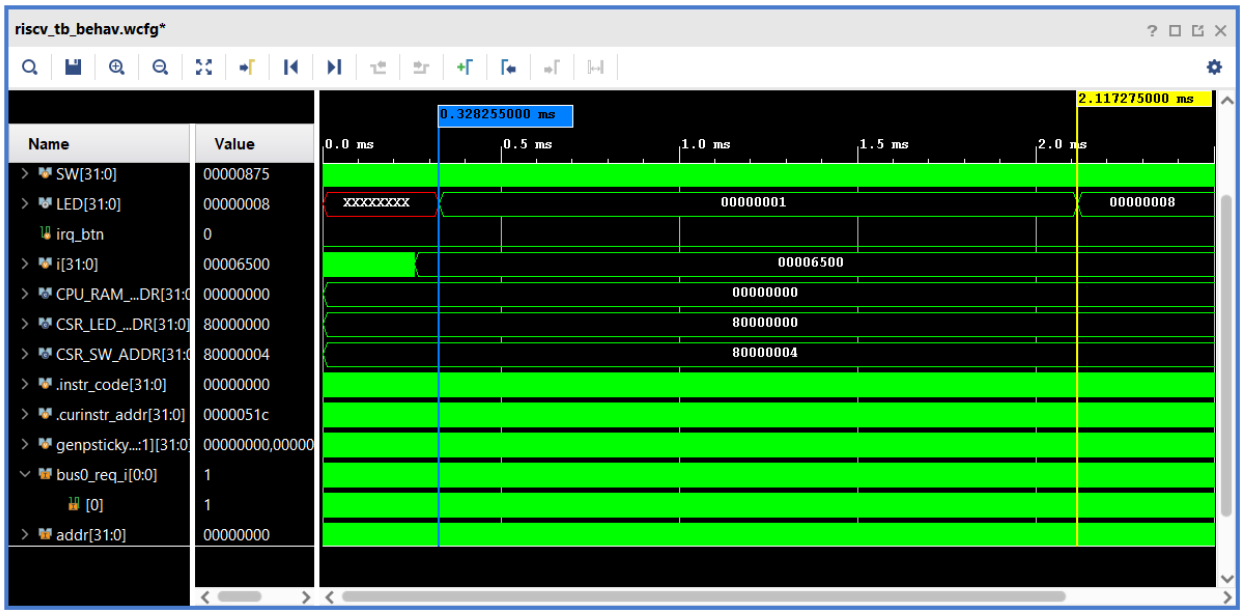

Рисунок 16 – Временная диаграмма работы модуля ускорения на шине расширения

В таблице 1 приведено время работы программы от окончания бит-инверсии до окончания циклов, поскольку во всех случаях бит-инверсия работает за одинаковое время и ускорители не затрагивают её работу. Данные по времени работы приведены для версии ядра с конвейером на 6 стадий, поскольку при другой длине конвейера прирост остаётся практически таким же.

Таблица 1 – Время работы программы

Реализация	**Время выполнения**	**Процент**
Программная	1,9 мс	100%
Сопроцессор	1,66 мс	87%
Шина UDM	1,79 мс	94%

Прирост производительности при использовании сопроцессора оказался более существенным, чем при использовании модуля расширения. Это связано с тем, что алгоритм часто обращается к памяти и возможность сопроцессора загружать сразу два операнда даёт больший прирост, чем параллельное выполнение математических операций. Возможно, асинхронная версия сопроцессора, реализующая весь функционал дискретного преобразования Фурье на двух точках, могла бы показать ещё лучшие результаты, но, вероятно, потребовался бы конвейер на 5 стадий, который было бы тяжело согласовать с конвейером самого процессора. Другим хорошим решением могло бы быть использование прямого доступа к памяти в модуле расширения, однако, это потребовало бы значительной подготовки и упорядочивания данных в памяти, чтобы минимизировать передачу адресов по системной шине.

4.2 Использование ресурсов ПЛИС

В таблице 2 приведены значения использованных аппаратных ресурсов ПЛИС после синтеза схемы в Vivado 2019.1. Данные также приведены для ядра с конвейером на 6 стадий. В данном случае разница между вариантами конвейеров только в использовании ресурсов программной реализацией, а сопроцессор и расширение на системной шине дают фиксированную прибавку к этим ресурсам.

Таблица 2 – Использование аппаратных ресурсов

Реализация	LUT	FF	DSP
Программная	4338	2852	4
Сопроцессор	4416	2897	5
Шина UDM	4511	2982	8

Можно заметить, что модуль расширения при меньшей производительности по сравнению с сопроцессором требует большего количества вычислительных блоков, а соответственно и площади на кристалле. Особенно это заметно по блокам DSP, которые реализуют умножение. Теоретически, их использование можно было бы сократить, сделав конвейер в модуле расширения длиннее, поскольку получение входных данных всё равно происходит дольше, чем работают умножители. В сопроцессоре блок умножения также можно было бы повторно использовать, взяв один из уже задействованных, однако сначала следует убедится, что ситуации, когда он одновременно понадобится в двух местах, не возникнет.

Данные по энергопотреблению различных версий нет смысла приводить, поскольку прирост хоть и есть, но составляет менее 1%, что в условиях реальной эксплуатации будет в пределах колебаний энергопотребления, вызванных внешними условиями.

Наилучшим вариантом по всем параметрам оказался сопроцессор. За небольшой прирост по занимаемым аппаратным ресурсам он даёт ощутимый прирост во времени работы алгоритма.

ЗАКЛЮЧЕНИЕ

Задачи работы включали:

1. Разработать программную реализацию алгоритма быстрого преобразования Фурье для архитектуры RISC-V.

2. Реализовать аппаратное ускорение алгоритма быстрого преобразования Фурье на основе интерфейсов системной шины и/или интерфейса расширения систем команд.

3. Выполнить сравнительный анализ полученных реализаций по критериям производительности, площади и энергопотребления, выбрать наилучшие варианты реализации.

В ходе работы был рассмотрен алгоритм быстрого преобразования Фурье и его оптимизации. С их учётом и обоснованием выбора была разработана программная реализация алгоритма.

Также в работе обоснован выбор архитектуры процессора, его реализации и, с учётом особенностей, реализованы оба предусмотренных задачей способа ускорения вычислений.

Результаты работы проанализированы и оценены сами реализации ускорителей, а также рассмотрены возможности по их улучшению и модификации для решения проблем, вызванных ограничениями аппаратной платформы.

В результате проделанной работы выполнены все задачи и достигнута цель – разработана программно-аппаратная система для ускорения работы алгоритма быстрого преобразования Фурье на основе процессора архитектуры RISC-V. Лучшим вариантом реализации оказалась реализация с сопроцессором.

Работа может быть продолжена согласно предложенным рекомендациям по возможным улучшениям, а именно вариант асинхронной работы более

сложного сопроцессора и прямой доступ к памяти для модуля расширения на системной шине. Так же возможны реализации, совмещающие оба варианта ускорителей с разделением задач между ними.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Cooley J. W., Tukey J. W. An algorithm for the machine calculation of complex Fourier series //Mathematics of computation. – 1965. – Т. 19. – №. 90. – С. 297-301.
2. Brigham E. O. The Fast Fourier Transform Prentice-Hall //Inc. Englewood Cliffs, NJ. – 1974. – Т. 974.
3. Oppenheim A. V. Discrete-time signal processing. – Pearson Education India, 1999.
4. Rader C. M. Discrete Fourier transforms when the number of data samples is prime //Proceedings of the IEEE. – 1968. – Т. 56. – №. 6. – С. 1107-1108.
5. Van Loan C. Computational frameworks for the fast Fourier transform. – Society for Industrial and Applied Mathematics, 1992.
6. Frigo M., Johnson S. G. FFTW: An adaptive software architecture for the FFT //Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181). – IEEE, 1998. – Т. 3. – С. 1381-1384.
7. Chen L. et al. Optimizing the fast fourier transform on a multi-core architecture //2007 IEEE International Parallel and Distributed Processing Symposium. – IEEE, 2007. – С. 1-8.
8. Zheng Z., Zhu X., Qian H. Design and Implementation of Arbitrary Point FFT Based on RISC-V SoC //2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC). – IEEE, 2021. – Т. 5. – С. 1216-1219.
9. Иванова Н. Н., Галанина Н. А., Моисеев Д. В. Особенности реализации алгоритма БПФ на ПЛИС типа FPGA //Вестник Чувашского университета. – 2018. – №. 3. – С. 182-191.
10. Antonov A. ActiveCore Laboratory work manual Using Sigma MCU in FPGA designs / A. Antonov. — Текст : электронный // GitHub : [сайт]. — URL:

https://github.com/AntonovAlexander/activecore/blob/master/designs/rtl/sigma/doc/sigma_lab_manual.pdf (дата обращения: 20.05.2023).

11. Antonov A. ActiveCore Laboratory work manual Using UDM bus transactor in FPGA designs / A. Antonov. — Текст : электронный // GitHub : [сайт]. — URL: https://github.com/AntonovAlexander/activecore/blob/master/designs/rtl/udm/doc/udm_lab_manual.pdf (дата обращения: 20.05.2023).
12. Waterman A., Asanovic K. The RISC-V Instruction Set Manual Volume I: Unprivileged ISA / Waterman A., Asanovic K. — Текст : электронный // riscv.org : [сайт]. — URL: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf (дата обращения: 20.05.2023).

ПРИЛОЖЕНИЕ

Приложение 1:

```
#include <math.h>
#include <stdlib.h>
#include <stdint.h>
#define IO_LED (*(volatile unsigned int *)(0x80000000))

typedef struct {
   double real;
   double img;
} complex;

int n = 1024;

static inline void bitReverse(complex *x, int n, int logn)
{
   int i, j, k;
   int t;
       complex temp;
   for (i = 0; i < n; i++)
   IO_LED = i;
   {
              k = i;
              j = 0;
      for (t = 0; t < logn; t++)
              {
                    j <<= 1;
                    j += (k & 1);
                    k >>= 1;
      }
      if (i<j){
         temp = x[i];
         x[i] = x[j];
         x[j] = temp;
      }
   }
}

static inline void add(complex a, complex b, complex *c)
{
   c->real = a.real + b.real;
   c->img  = a.img  + b.img ;
```

```
}

static inline void mul(complex a, complex b, complex *c)
{
   c->real = a.real*b.real - a.img*b.img ;
   c->img  = a.real*b.img  + a.img*b.real;
}

static inline void sub(complex a, complex b, complex *c)
{
   c->real = a.real - b.real;
   c->img  = a.img  - b.img ;
}

static inline void fft(complex *x, complex W)
{
   int i = 0, j = 0, k = 0, l = 0;
   int logn = 10;
   complex up, down, product;
   bitReverse(x, n, logn);
   IO_LED = 1;
   for (i = 0; i <logn; i ++){
      l = 1 << i;
      for (j = 0; j <n ; j += 2 * l){
         for (k = 0; k < l; k++){
            mul(x[j + k + l], W, &product);
            add(x[j + k], product, &up);
            sub(x[j + k], product, &down);
            x[j + k] = up;
            x[j + k + l] = down;
         }
      }
   }
}

int main()
{
   IO_LED = 0x55aa55aa;
   int i;
   complex W[512] = { {1.0000,0.0000},{1.0000,-0.0061},{0.9999,-
0.0123},{0.9998,-0.0184},{0.9997,-0.0245},{0.9995,-0.0307},{0.9993,-
0.0368},
```

```
   {0.9991,-0.0429},{0.9988,-0.0491},{0.9985,-0.0552},{0.9981,-
0.0613},{0.9977,-0.0674},{0.9973,-0.0736},{0.9968,-0.0797},{0.9963,-
0.0858},
   {0.9958,-0.0919},{0.9952,-0.0980},{0.9946,-0.1041},{0.9939,-
0.1102},{0.9932,-0.1163},{0.9925,-0.1224},{0.9917,-0.1285},{0.9909,-0.1346},
   {0.9901,-0.1407},{0.9892,-0.1467},{0.9883,-0.1528},{0.9873,-
0.1589},{0.9863,-0.1649},{0.9853,-0.1710},{0.9842,-0.1770},{0.9831,-
0.1830},
   {0.9820,-0.1891},{0.9808,-0.1951},{0.9796,-0.2011},{0.9783,-
0.2071},{0.9770,-0.2131},{0.9757,-0.2191},{0.9743,-0.2251},{0.9729,-
0.2311},
   {0.9715,-0.2370},{0.9700,-0.2430},{0.9685,-0.2489},{0.9670,-
0.2549},{0.9654,-0.2608},{0.9638,-0.2667},{0.9621,-0.2726},{0.9604,-
0.2785},
   {0.9587,-0.2844},{0.9569,-0.2903},{0.9551,-0.2962},{0.9533,-
0.3020},{0.9514,-0.3078},{0.9495,-0.3137},{0.9476,-0.3195},{0.9456,-
0.3253},
   {0.9436,-0.3311},{0.9415,-0.3369},{0.9395,-0.3427},{0.9373,-
0.3484},{0.9352,-0.3542},{0.9330,-0.3599},{0.9308,-0.3656},{0.9285,-
0.3713},
   {0.9262,-0.3770},{0.9239,-0.3827},{0.9215,-0.3883},{0.9191,-
0.3940},{0.9167,-0.3996},{0.9142,-0.4052},{0.9117,-0.4108},{0.9092,-
0.4164},
   {0.9066,-0.4220},{0.9040,-0.4276},{0.9013,-0.4331},{0.8987,-
0.4386},{0.8960,-0.4441},{0.8932,-0.4496},{0.8904,-0.4551},{0.8876,-
0.4605},
   {0.8848,-0.4660},{0.8819,-0.4714},{0.8790,-0.4768},{0.8761,-
0.4822},{0.8731,-0.4876},{0.8701,-0.4929},{0.8670,-0.4982},{0.8640,-
0.5035},
   {0.8609,-0.5088},{0.8577,-0.5141},{0.8546,-0.5194},{0.8514,-
0.5246},{0.8481,-0.5298},{0.8449,-0.5350},{0.8416,-0.5402},{0.8382,-
0.5453},
   {0.8349,-0.5505},{0.8315,-0.5556},{0.8280,-0.5607},{0.8246,-
0.5657},{0.8211,-0.5708},{0.8176,-0.5758},{0.8140,-0.5808},{0.8105,-
0.5858},
   {0.8068,-0.5908},{0.8032,-0.5957},{0.7995,-0.6006},{0.7958,-
0.6055},{0.7921,-0.6104},{0.7883,-0.6152},{0.7846,-0.6201},{0.7807,-
0.6249},
   {0.7769,-0.6296},{0.7730,-0.6344},{0.7691,-0.6391},{0.7652,-
0.6438},{0.7612,-0.6485},{0.7572,-0.6532},{0.7532,-0.6578},{0.7491,-
0.6624},
   {0.7451,-0.6670},{0.7410,-0.6716},{0.7368,-0.6761},{0.7327,-
0.6806},{0.7285,-0.6851},{0.7242,-0.6895},{0.7200,-0.6940},{0.7157,-
0.6984},
```

```
   {0.7114,-0.7028},{0.7071,-0.7071},{0.7028,-0.7114},{0.6984,-
0.7157},{0.6940,-0.7200},{0.6895,-0.7242},{0.6851,-0.7285},{0.6806,-
0.7327},
   {0.6761,-0.7368},{0.6716,-0.7410},{0.6670,-0.7451},{0.6624,-
0.7491},{0.6578,-0.7532},{0.6532,-0.7572},{0.6485,-0.7612},{0.6438,-
0.7652},
   {0.6391,-0.7691},{0.6344,-0.7730},{0.6296,-0.7769},{0.6249,-
0.7807},{0.6201,-0.7846},{0.6152,-0.7883},{0.6104,-0.7921},{0.6055,-
0.7958},
   {0.6006,-0.7995},{0.5957,-0.8032},{0.5908,-0.8068},{0.5858,-
0.8105},{0.5808,-0.8140},{0.5758,-0.8176},{0.5708,-0.8211},{0.5657,-
0.8246},
   {0.5607,-0.8280},{0.5556,-0.8315},{0.5505,-0.8349},{0.5453,-
0.8382},{0.5402,-0.8416},{0.5350,-0.8449},{0.5298,-0.8481},{0.5246,-
0.8514},
   {0.5194,-0.8546},{0.5141,-0.8577},{0.5088,-0.8609},{0.5035,-
0.8640},{0.4982,-0.8670},{0.4929,-0.8701},{0.4876,-0.8731},{0.4822,-
0.8761},
   {0.4768,-0.8790},{0.4714,-0.8819},{0.4660,-0.8848},{0.4605,-
0.8876},{0.4551,-0.8904},{0.4496,-0.8932},{0.4441,-0.8960},{0.4386,-
0.8987},
   {0.4331,-0.9013},{0.4276,-0.9040},{0.4220,-0.9066},{0.4164,-
0.9092},{0.4108,-0.9117},{0.4052,-0.9142},{0.3996,-0.9167},{0.3940,-
0.9191},
   {0.3883,-0.9215},{0.3827,-0.9239},{0.3770,-0.9262},{0.3713,-
0.9285},{0.3656,-0.9308},{0.3599,-0.9330},{0.3542,-0.9352},{0.3484,-
0.9373},
   {0.3427,-0.9395},{0.3369,-0.9415},{0.3311,-0.9436},{0.3253,-
0.9456},{0.3195,-0.9476},{0.3137,-0.9495},{0.3078,-0.9514},{0.3020,-
0.9533},
   {0.2962,-0.9551},{0.2903,-0.9569},{0.2844,-0.9587},{0.2785,-
0.9604},{0.2726,-0.9621},{0.2667,-0.9638},{0.2608,-0.9654},{0.2549,-
0.9670},
   {0.2489,-0.9685},{0.2430,-0.9700},{0.2370,-0.9715},{0.2311,-
0.9729},{0.2251,-0.9743},{0.2191,-0.9757},{0.2131,-0.9770},{0.2071,-
0.9783},
   {0.2011,-0.9796},{0.1951,-0.9808},{0.1891,-0.9820},{0.1830,-
0.9831},{0.1770,-0.9842},{0.1710,-0.9853},{0.1649,-0.9863},{0.1589,-
0.9873},
   {0.1528,-0.9883},{0.1467,-0.9892},{0.1407,-0.9901},{0.1346,-
0.9909},{0.1285,-0.9917},{0.1224,-0.9925},{0.1163,-0.9932},{0.1102,-0.9939},
   {0.1041,-0.9946},{0.0980,-0.9952},{0.0919,-0.9958},{0.0858,-
0.9963},{0.0797,-0.9968},{0.0736,-0.9973},{0.0674,-0.9977},{0.0613,-
0.9981},
```

```
    {0.0552,-0.9985},{0.0491,-0.9988},{0.0429,-0.9991},{0.0368,-
0.9993},{0.0307,-0.9995},{0.0245,-0.9997},{0.0184,-0.9998},{0.0123,-
0.9999},
    {0.0061,-1.0000},{0.0000,-1.0000},{-0.0061,-1.0000},{-0.0123,-0.9999},{-
0.0184,-0.9998},{-0.0245,-0.9997},{-0.0307,-0.9995},
    {-0.0368,-0.9993},{-0.0429,-0.9991},{-0.0491,-0.9988},{-0.0552,-0.9985},{-
0.0613,-0.9981},{-0.0674,-0.9977},{-0.0736,-0.9973},
    {-0.0797,-0.9968},{-0.0858,-0.9963},{-0.0919,-0.9958},{-0.0980,-0.9952},{-
0.1041,-0.9946},{-0.1102,-0.9939},{-0.1163,-0.9932},
    {-0.1224,-0.9925},{-0.1285,-0.9917},{-0.1346,-0.9909},{-0.1407,-0.9901},{-
0.1467,-0.9892},{-0.1528,-0.9883},{-0.1589,-0.9873},
    {-0.1649,-0.9863},{-0.1710,-0.9853},{-0.1770,-0.9842},{-0.1830,-0.9831},{-
0.1891,-0.9820},{-0.1951,-0.9808},{-0.2011,-0.9796},
    {-0.2071,-0.9783},{-0.2131,-0.9770},{-0.2191,-0.9757},{-0.2251,-0.9743},{-
0.2311,-0.9729},{-0.2370,-0.9715},{-0.2430,-0.9700},
    {-0.2489,-0.9685},{-0.2549,-0.9670},{-0.2608,-0.9654},{-0.2667,-0.9638},{-
0.2726,-0.9621},{-0.2785,-0.9604},{-0.2844,-0.9587},
    {-0.2903,-0.9569},{-0.2962,-0.9551},{-0.3020,-0.9533},{-0.3078,-0.9514},{-
0.3137,-0.9495},{-0.3195,-0.9476},{-0.3253,-0.9456},
    {-0.3311,-0.9436},{-0.3369,-0.9415},{-0.3427,-0.9395},{-0.3484,-0.9373},{-
0.3542,-0.9352},{-0.3599,-0.9330},{-0.3656,-0.9308},
    {-0.3713,-0.9285},{-0.3770,-0.9262},{-0.3827,-0.9239},{-0.3883,-0.9215},{-
0.3940,-0.9191},{-0.3996,-0.9167},{-0.4052,-0.9142},
    {-0.4108,-0.9117},{-0.4164,-0.9092},{-0.4220,-0.9066},{-0.4276,-0.9040},{-
0.4331,-0.9013},{-0.4386,-0.8987},{-0.4441,-0.8960},
    {-0.4496,-0.8932},{-0.4551,-0.8904},{-0.4605,-0.8876},{-0.4660,-0.8848},{-
0.4714,-0.8819},{-0.4768,-0.8790},{-0.4822,-0.8761},
    {-0.4876,-0.8731},{-0.4929,-0.8701},{-0.4982,-0.8670},{-0.5035,-0.8640},{-
0.5088,-0.8609},{-0.5141,-0.8577},{-0.5194,-0.8546},
    {-0.5246,-0.8514},{-0.5298,-0.8481},{-0.5350,-0.8449},{-0.5402,-0.8416},{-
0.5453,-0.8382},{-0.5505,-0.8349},{-0.5556,-0.8315},
    {-0.5607,-0.8280},{-0.5657,-0.8246},{-0.5708,-0.8211},{-0.5758,-0.8176},{-
0.5808,-0.8140},{-0.5858,-0.8105},{-0.5908,-0.8068},
    {-0.5957,-0.8032},{-0.6006,-0.7995},{-0.6055,-0.7958},{-0.6104,-0.7921},{-
0.6152,-0.7883},{-0.6201,-0.7846},{-0.6249,-0.7807},
    {-0.6296,-0.7769},{-0.6344,-0.7730},{-0.6391,-0.7691},{-0.6438,-0.7652},{-
0.6485,-0.7612},{-0.6532,-0.7572},{-0.6578,-0.7532},
    {-0.6624,-0.7491},{-0.6670,-0.7451},{-0.6716,-0.7410},{-0.6761,-0.7368},{-
0.6806,-0.7327},{-0.6851,-0.7285},{-0.6895,-0.7242},
    {-0.6940,-0.7200},{-0.6984,-0.7157},{-0.7028,-0.7114},{-0.7071,-0.7071},{-
0.7114,-0.7028},{-0.7157,-0.6984},{-0.7200,-0.6940},
    {-0.7242,-0.6895},{-0.7285,-0.6851},{-0.7327,-0.6806},{-0.7368,-0.6761},{-
0.7410,-0.6716},{-0.7451,-0.6670},{-0.7491,-0.6624},
```

```
   {-0.7532,-0.6578},{-0.7572,-0.6532},{-0.7612,-0.6485},{-0.7652,-0.6438},{-
0.7691,-0.6391},{-0.7730,-0.6344},{-0.7769,-0.6296},
   {-0.7807,-0.6249},{-0.7846,-0.6201},{-0.7883,-0.6152},{-0.7921,-0.6104},{-
0.7958,-0.6055},{-0.7995,-0.6006},{-0.8032,-0.5957},
   {-0.8068,-0.5908},{-0.8105,-0.5858},{-0.8140,-0.5808},{-0.8176,-0.5758},{-
0.8211,-0.5708},{-0.8246,-0.5657},{-0.8280,-0.5607},
   {-0.8315,-0.5556},{-0.8349,-0.5505},{-0.8382,-0.5453},{-0.8416,-0.5402},{-
0.8449,-0.5350},{-0.8481,-0.5298},{-0.8514,-0.5246},
   {-0.8546,-0.5194},{-0.8577,-0.5141},{-0.8609,-0.5088},{-0.8640,-0.5035},{-
0.8670,-0.4982},{-0.8701,-0.4929},{-0.8731,-0.4876},
   {-0.8761,-0.4822},{-0.8790,-0.4768},{-0.8819,-0.4714},{-0.8848,-0.4660},{-
0.8876,-0.4605},{-0.8904,-0.4551},{-0.8932,-0.4496},
   {-0.8960,-0.4441},{-0.8987,-0.4386},{-0.9013,-0.4331},{-0.9040,-0.4276},{-
0.9066,-0.4220},{-0.9092,-0.4164},{-0.9117,-0.4108},
   {-0.9142,-0.4052},{-0.9167,-0.3996},{-0.9191,-0.3940},{-0.9215,-0.3883},{-
0.9239,-0.3827},{-0.9262,-0.3770},{-0.9285,-0.3713},
   {-0.9308,-0.3656},{-0.9330,-0.3599},{-0.9352,-0.3542},{-0.9373,-0.3484},{-
0.9395,-0.3427},{-0.9415,-0.3369},{-0.9436,-0.3311},
   {-0.9456,-0.3253},{-0.9476,-0.3195},{-0.9495,-0.3137},{-0.9514,-0.3078},{-
0.9533,-0.3020},{-0.9551,-0.2962},{-0.9569,-0.2903},
   {-0.9587,-0.2844},{-0.9604,-0.2785},{-0.9621,-0.2726},{-0.9638,-0.2667},{-
0.9654,-0.2608},{-0.9670,-0.2549},{-0.9685,-0.2489},
   {-0.9700,-0.2430},{-0.9715,-0.2370},{-0.9729,-0.2311},{-0.9743,-0.2251},{-
0.9757,-0.2191},{-0.9770,-0.2131},{-0.9783,-0.2071},
   {-0.9796,-0.2011},{-0.9808,-0.1951},{-0.9820,-0.1891},{-0.9831,-0.1830},{-
0.9842,-0.1770},{-0.9853,-0.1710},{-0.9863,-0.1649},
   {-0.9873,-0.1589},{-0.9883,-0.1528},{-0.9892,-0.1467},{-0.9901,-0.1407},{-
0.9909,-0.1346},{-0.9917,-0.1285},{-0.9925,-0.1224},
   {-0.9932,-0.1163},{-0.9939,-0.1102},{-0.9946,-0.1041},{-0.9952,-0.0980},{-
0.9958,-0.0919},{-0.9963,-0.0858},{-0.9968,-0.0797},
   {-0.9973,-0.0736},{-0.9977,-0.0674},{-0.9981,-0.0613},{-0.9985,-0.0552},{-
0.9988,-0.0491},{-0.9991,-0.0429},{-0.9993,-0.0368},
   {-0.9995,-0.0307},{-0.9997,-0.0245},{-0.9998,-0.0184},{-0.9999,-0.0123},{-
1.0000,-0.0061} };
   IO_LED = 0x55aa56aa;
   complex x[1024] = {{6.0000,0.0000},{-0.7139,-1.9400},{0.8378,-3.1244},{-
3.4842,-1.9592},{-2.0287,3.3488},{-1.6254,0.3290},{3.6205,4.6756},{1.5478,-
1.9207},{2.1475,-1.4207},{-0.1456,-3.9710},{-4.4704,0.7825},{-0.8665,-
0.8701},{-1.4053,5.4808},{2.8379,0.4010},{2.0742,0.0095},{3.2494,-
2.1899},{-3.8455,-3.2827},{0.0894,-0.7631},{-
4.8019,2.1120},{1.2797,3.0053},{1.7231,0.8678},{3.5018,1.5206},{0.1776,-
5.4351},{0.1711,-0.2784},{-4.1597,-2.1809},{-
2.1145,2.9876},{1.2437,1.8235},{0.6323,3.5999},{4.5780,-3.3375},{-0.0234,-
0.4695},{-0.8228,-3.9541},{-4.0267,-0.2089},{-0.1597,2.7311},{-
```

2.4359,2.3940},{5.4963,1.6181},{0.6006,-0.8694},{2.0050,-2.6323},{-2.1856,-3.7739},{-2.4592,2.2467},{-3.0465,-0.5116},{1.9951,5.2651},{1.7116,-0.1340},{2.5408,-0.2960},{2.0373,-4.1507},{-3.8698,-0.3838},{-1.2745,-2.3528},{-2.8772,4.5413},{1.5806,1.8126},{1.5649,1.0215},{4.7461,-0.6978},{-2.4344,-3.6105},{0.7596,-2.0551},{-4.9671,0.3756},{-0.6923,3.0144},{0.6810,1.2283},{3.3521,3.5205},{1.4379,-4.4737},{1.4963,-0.7724},{-2.9222,-3.3789},{-3.4551,1.4628},{0.2278,1.3729},{-0.8674,4.7217},{4.6318,-1.6248},{1.2910,0.0584},{0.8478,-3.7654},{-3.7285,-2.2467},{-0.6200,1.6613},{-4.1257,2.0517},{4.1116,2.8100},{1.2105,0.4521},{2.9152,-1.2879},{-0.4322,-4.8485},{-2.0220,1.0651},{-3.8605,-1.8537},{0.1072,4.9189},{1.1904,1.2921},{2.1631,1.2123},{3.9477,-3.4914},{-2.6098,-0.9658},{-0.9098,-3.7271},{-3.8284,2.7980},{0.1210,2.4458},{0.2707,1.7481},{5.4203,1.1370},{-1.0052,-3.0475},{1.8756,-2.7104},{-4.2056,-1.4985},{-2.2034,2.3248},{-0.6896,0.8442},{2.4019,5.1206},{2.0443,-2.8948},{2.6649,-0.5028},{-0.9882,-3.9908},{-3.9351,-0.1879},{-0.4848,0.1834},{-2.5933,4.9786},{3.7843,0.0102},{1.9376,1.0052},{2.5420,-2.6971},{-2.7425,-3.7037},{-0.3447,0.3025},{-5.3219,0.9700},{2.1899,3.2726},{1.0410,1.5774},{3.2099,0.6385},{1.3404,-4.9965},{-0.8710,0.2172},{-3.8029,-3.3243},{-1.5831,3.6930},{0.2618,2.0002},{0.9739,2.6573},{5.1588,-2.1916},{-1.1051,-0.7957},{0.1326,-4.5530},{-3.9922,0.6352},{-1.0903,2.2635},{-1.5054,1.8576},{5.1369,2.8675},{0.0321,-1.7899},{3.0542,-2.5051},{-2.6271,-3.0524},{-2.9232,1.2454},{-1.9385,-0.2862},{0.8918,5.9272},{1.8634,-1.1408},{3.2580,0.4014},{1.2266,-3.7914},{-3.5627,-1.5122},{-0.5858,-1.4142},{-4.0975,4.2896},{2.2769,1.1652},{1.7301,1.9788},{3.8137,-0.9077},{-1.4107,-4.2845},{0.6330,-0.8916},{-5.6710,-0.5750},{0.1979,2.9136},{0.2539,2.1072},{2.7067,2.7100},{2.6845,-4.2767},{0.6363,-0.0102},{-2.8441,-4.4826},{-2.6896,1.8754},{-0.6718,1.8414},{-0.8278,3.6106},{5.4187,-0.6248},{0.1861,0.0592},{1.5489,-4.5188},{-3.3232,-1.4633},{-1.6726,1.4638},{-3.3146,1.2071},{4.0093,4.0600},{0.4121,-0.2234},{3.8668,-1.4559},{-0.5620,-3.9261},{-2.7432,0.1920},{-2.6572,-1.9312},{-0.7796,5.7402},{0.9996,0.3263},{3.0040,1.6146},{3.2325,-2.7843},{-2.5759,-2.1503},{0.0329,-2.9641},{-4.9692,2.8205},{0.5250,1.6011},{0.7272,2.5632},{4.3366,1.2379},{-0.1624,-3.9270},{2.0541,-1.5315},{-5.0275,-2.1687},{-1.4029,1.8792},{-0.8046,1.8067},{1.4558,4.4417},{3.2673,-2.9713},{2.0381,0.4601},{-1.1970,-4.9483},{-3.0073,-0.0913},{-1.2043,0.9141},{-2.8524,3.7838},{4.7188,0.7701},{0.8914,1.3141},{2.9169,-3.5325},{-2.0117,-3.0453},{-1.4256,0.4145},{-4.6851,-0.1123},{2.3630,4.4164},{0.0886,1.1838},{3.9664,0.1846},{1.5259,-3.9562},{-1.8033,-0.4373},{-2.5953,-3.6857},{-2.1715,4.6080},{-

0.2346,1.1522},{1.8598,2.7122},{4.5971,-1.2035},{-1.3680,-1.9380},{1.2457,-
4.0118},{-4.9536,0.9361},{-0.9972,1.3067},{-
0.7858,2.4433},{3.9948,3.2644},{0.6127,-2.8164},{3.5041,-1.4018},{-3.4973,-
3.3778},{-2.2814,0.5116},{-1.7099,0.6761},{-0.2701,5.4316},{2.9652,-
1.4996},{2.8982,1.4729},{0.7429,-4.5142},{-2.5654,-1.7301},{-1.0491,-
0.4667},{-4.6241,3.1097},{3.2934,1.6165},{0.8282,2.5470},{3.8189,-
1.7525},{-0.4309,-3.8110},{-0.3893,-0.4576},{-5.2517,-
1.7974},{0.6415,3.8606},{-0.7547,2.0126},{3.1933,1.9997},{3.1640,-
3.2182},{-0.4482,-0.3818},{-1.7311,-5.0829},{-2.9282,2.8154},{-
1.4086,1.1870},{0.0286,3.2974},{5.0616,0.5507},{-0.3695,-0.9548},{2.7275,-
4.2315},{-4.0268,-0.9036},{-1.8879,0.4960},{-
2.3783,1.5034},{2.9126,4.7142},{0.6798,-1.3299},{4.5303,-0.5216},{-1.4113,-
3.8760},{-2.3240,-0.7482},{-2.0860,-1.0447},{-2.0512,5.4691},{1.9008,-
0.2963},{2.9284,2.6869},{2.5089,-3.2102},{-1.6134,-2.6600},{-0.1289,-
1.8605},{-
5.7110,1.7593},{1.5532,1.7095},{0.0518,3.3202},{3.9648,0.4518},{0.9694,-
3.6901},{1.1643,-0.7954},{-4.8563,-3.4183},{-0.7207,2.5619},{-
1.7619,2.0100},{1.6281,3.5258},{4.0000,-2.0000},{0.8719,0.4021},{-0.2766,-
5.7240},{-2.8814,0.8043},{-2.0982,0.5213},{-
2.0919,3.1187},{4.6060,2.0244},{0.0788,0.4963},{4.0470,-3.5164},{-2.4071,-
2.2722},{-1.9286,-0.4581},{-3.5971,-
0.1324},{1.4155,5.2727},{0.0286,0.0720},{4.7705,0.8637},{0.7631,-3.5385},{-
1.6591,-1.4884},{-1.7183,-2.9384},{-
3.4381,4.5899},{0.4090,0.3142},{2.0694,3.6750},{3.6925,-1.3014},{-0.5433,-
2.7001},{1.3961,-2.8278},{-5.8446,0.0909},{-0.0305,1.0754},{-
1.1643,3.3073},{3.2774,2.5982},{1.7912,-2.8539},{2.8045,-0.4174},{-3.5899,-
4.5399},{-1.4202,0.8895},{-2.5115,1.1597},{-0.4237,4.3817},{3.8962,-
0.7142},{1.7309,1.7234},{1.3853,-5.3920},{-2.0988,-0.9468},{-2.0099,-
0.5458},{-4.0157,2.1456},{3.4492,2.8391},{-0.1750,1.9733},{4.7918,-
2.0087},{-0.4969,-2.8946},{-1.1438,-1.1381},{-4.0939,-2.1274},{-
0.0660,4.8530},{-1.1210,0.9750},{4.0593,2.3548},{2.5487,-2.4786},{-0.6133,-
1.4463},{-0.6194,-4.5245},{-4.0799,3.0613},{-
1.0558,0.2094},{0.5076,4.0514},{4.0562,0.7811},{0.2286,-1.9168},{3.1672,-
3.0545},{-4.9945,-1.4539},{-1.0562,-0.0357},{-
2.4111,2.3909},{1.9164,4.2211},{1.8036,-1.6595},{4.0617,0.6272},{-1.7654,-
4.8478},{-1.3671,-0.6898},{-2.6444,-0.3140},{-
2.5113,4.3742},{2.9654,0.2242},{1.8472,3.2079},{2.8110,-4.1144},{-0.8644,-
2.0533},{-1.0691,-1.5979},{-5.2997,0.5812},{1.9808,2.8000},{-
1.0500,3.0185},{4.6895,-0.0594},{1.2264,-2.7203},{0.2082,-1.2085},{-3.7228,-
4.0216},{-1.1217,3.6190},{-2.3816,1.1278},{2.4789,3.5155},{3.5860,-
1.0160},{0.3890,-0.5866},{0.9719,-5.3906},{-3.8250,1.3042},{-2.0473,-
0.4993},{-1.3768,3.5879},{3.5955,2.5563},{0.3874,-0.6025},{4.7253,-
2.4405},{-3.3778,-2.4776},{-1.3025,-1.2220},{-
3.2654,0.7011},{0.2351,4.9960},{1.0102,-0.5416},{4.5573,2.0715},{0.1711,-

4.2395},{-0.7060,-1.7409},{-1.9733,-2.0099},{-
4.1599,3.5487},{1.5364,0.5207},{1.1624,4.4032},{3.6249,-2.1594},{0.4039,-
2.3276},{0.5533,-2.2265},{-5.6635,-1.1940},{0.6467,1.9529},{-
2.2568,3.2874},{3.6904,1.8695},{2.3423,-1.9317},{1.7097,-0.5185},{-2.5790,-
5.3575},{-1.4816,1.9378},{-
3.3094,0.5098},{0.3420,3.9993},{3.7277,0.4152},{0.9522,0.8847},{2.6564,-
5.3054},{-2.7659,-0.2274},{-2.2528,-1.5024},{-
3.1163,2.2853},{2.5346,3.6248},{-0.1851,0.8095},{5.6382,-1.1264},{-1.3996,-
2.7411},{-0.7853,-2.0469},{-3.4142,-1.4142},{-1.3201,4.8243},{-
0.3493,0.1149},{4.1104,3.5067},{1.7652,-2.8580},{0.2321,-1.9820},{-0.5448,-
3.4631},{-5.0006,2.1728},{0.0591,0.0886},{-
0.1439,4.9075},{3.6268,0.0354},{1.2753,-1.8244},{2.4837,-2.1519},{-5.0632,-
2.7288},{-0.1790,0.5728},{-3.3834,2.6458},{1.9870,3.3333},{2.6023,-
0.8848},{2.9026,0.8470},{-0.9696,-5.8060},{-1.0934,0.2769},{-3.5332,-
0.6666},{-1.8904,3.6507},{3.0728,1.3911},{0.8271,2.5744},{3.9882,-
4.2817},{-1.2155,-1.1742},{-1.5812,-2.3867},{-
4.2858,0.3813},{1.2569,3.7773},{-1.3713,1.8687},{5.6237,0.5482},{0.4584,-
2.2325},{0.2524,-2.1676},{-2.7477,-3.4818},{-2.3366,3.8546},{-
1.8646,0.0879},{2.7876,4.5009},{2.6797,-1.0544},{1.0312,-1.3627},{1.3668,-
4.2751},{-4.8714,0.6554},{-1.0223,-0.9235},{-
1.7083,4.4869},{2.8499,1.9807},{1.4330,-0.8182},{4.2463,-1.3073},{-3.6996,-
3.6304},{-0.3007,-0.9124},{-4.0173,1.2086},{-0.0344,4.0265},{1.9975,-
0.0010},{3.4178,2.5858},{0.6764,-5.2589},{-0.1381,-0.9264},{-2.8648,-
2.0204},{-3.7322,2.5508},{1.9309,1.6212},{-0.0155,4.0103},{4.6030,-
2.5713},{0.3797,-1.3704},{-0.1888,-2.7597},{-4.6197,-
1.7104},{0.1898,3.0511},{-2.8478,2.2346},{4.6305,2.1434},{1.7689,-
1.1688},{1.4157,-1.4373},{-1.3932,-5.0298},{-2.5547,2.4346},{-3.0695,-
0.6184},{0.8851,4.7254},{2.7855,0.7075},{1.3168,-0.0770},{3.3307,-
4.2249},{-3.8608,-0.5728},{-1.3942,-2.1741},{-
3.0889,3.1455},{1.5509,3.2673},{0.7679,0.2813},{5.3926,0.1396},{-1.9586,-
3.6766},{0.2458,-2.0418},{-3.8686,-0.6920},{-
1.8982,3.8641},{0.7593,0.3604},{3.0791,4.2597},{1.9323,-3.8599},{1.0232,-
1.3850},{-1.3585,-3.1148},{-4.8024,0.9951},{0.7283,1.0335},{-
1.3744,4.7723},{4.3236,-0.5931},{1.5639,-0.8859},{1.5640,-2.3695},{-4.0829,-
3.5115},{-0.3218,1.7169},{-
4.1757,1.7728},{2.8566,3.2454},{2.2748,0.0672},{2.2756,0.0469},{0.3195,-
5.7153},{-1.9428,1.0087},{-3.5728,-1.7758},{-
1.1533,4.0528},{2.1917,1.9792},{0.8705,1.4921},{4.8765,-3.3287},{-2.2821,-
1.1849},{-0.9602,-3.2261},{-3.8937,1.1307},{0.1352,3.6748},{-
0.5854,1.0523},{5.6247,1.8327},{-0.3004,-2.8812},{1.2074,-2.4519},{-2.8595,-
2.5981},{-3.1692,2.9999},{-0.7083,0.0107},{1.9513,5.4159},{2.4959,-
1.9653},{1.9548,-1.0389},{0.6978,-3.5851},{-4.9257,-0.5907},{-0.1180,-
0.2015},{-2.8762,4.6097},{3.2139,1.1837},{2.0000,0.0000},{3.2139,-
1.1837},{-2.8762,-4.6097},{-0.1180,0.2015},{-

4.9257,0.5907},{0.6978,3.5851},{1.9548,1.0389},{2.4959,1.9653},{1.9513,-5.4159},{-0.7083,-0.0107},{-3.1692,-2.9999},{-2.8595,2.5981},{1.2074,2.4519},{-0.3004,2.8812},{5.6247,-1.8327},{-0.5854,-1.0523},{0.1352,-3.6748},{-3.8937,-1.1307},{-0.9602,3.2261},{-2.2821,1.1849},{4.8765,3.3287},{0.8705,-1.4921},{2.1917,-1.9792},{-1.1533,-4.0528},{-3.5728,1.7758},{-1.9428,-1.0087},{0.3195,5.7153},{2.2756,-0.0469},{2.2748,-0.0672},{2.8566,-3.2454},{-4.1757,-1.7728},{-0.3218,-1.7169},{-4.0829,3.5115},{1.5640,2.3695},{1.5639,0.8859},{4.3236,0.5931},{-1.3744,-4.7723},{0.7283,-1.0335},{-4.8024,-0.9951},{-1.3585,3.1148},{1.0232,1.3850},{1.9323,3.8599},{3.0791,-4.2597},{0.7593,-0.3604},{-1.8982,-3.8641},{-3.8686,0.6920},{0.2458,2.0418},{-1.9586,3.6766},{5.3926,-0.1396},{0.7679,-0.2813},{1.5509,-3.2673},{-3.0889,-3.1455},{-1.3942,2.1741},{-3.8608,0.5728},{3.3307,4.2249},{1.3168,0.0770},{2.7855,-0.7075},{0.8851,-4.7254},{-3.0695,0.6184},{-2.5547,-2.4346},{-1.3932,5.0298},{1.4157,1.4373},{1.7689,1.1688},{4.6305,-2.1434},{-2.8478,-2.2346},{0.1898,-3.0511},{-4.6197,1.7104},{-0.1888,2.7597},{0.3797,1.3704},{4.6030,2.5713},{-0.0155,-4.0103},{1.9309,-1.6212},{-3.7322,-2.5508},{-2.8648,2.0204},{-0.1381,0.9264},{0.6764,5.2589},{3.4178,-2.5858},{1.9975,0.0010},{-0.0344,-4.0265},{-4.0173,-1.2086},{-0.3007,0.9124},{-3.6996,3.6304},{4.2463,1.3073},{1.4330,0.8182},{2.8499,-1.9807},{-1.7083,-4.4869},{-1.0223,0.9235},{-4.8714,-0.6554},{1.3668,4.2751},{1.0312,1.3627},{2.6797,1.0544},{2.7876,-4.5009},{-1.8646,-0.0879},{-2.3366,-3.8546},{-2.7477,3.4818},{0.2524,2.1676},{0.4584,2.2325},{5.6237,-0.5482},{-1.3713,-1.8687},{1.2569,-3.7773},{-4.2858,-0.3813},{-1.5812,2.3867},{-1.2155,1.1742},{3.9882,4.2817},{0.8271,-2.5744},{3.0728,-1.3911},{-1.8904,-3.6507},{-3.5332,0.6666},{-1.0934,-0.2769},{-0.9696,5.8060},{2.9026,-0.8470},{2.6023,0.8848},{1.9870,-3.3333},{-3.3834,-2.6458},{-0.1790,-0.5728},{-5.0632,2.7288},{2.4837,2.1519},{1.2753,1.8244},{3.6268,-0.0354},{-0.1439,-4.9075},{0.0591,-0.0886},{-5.0006,-2.1728},{-0.5448,3.4631},{0.2321,1.9820},{1.7652,2.8580},{4.1104,-3.5067},{-0.3493,-0.1149},{-1.3201,-4.8243},{-3.4142,1.4142},{-0.7853,2.0469},{-1.3996,2.7411},{5.6382,1.1264},{-0.1851,-0.8095},{2.5346,-3.6248},{-3.1163,-2.2853},{-2.2528,1.5024},{-2.7659,0.2274},{2.6564,5.3054},{0.9522,-0.8847},{3.7277,-0.4152},{0.3420,-3.9993},{-3.3094,-0.5098},{-1.4816,-1.9378},{-2.5790,5.3575},{1.7097,0.5185},{2.3423,1.9317},{3.6904,-1.8695},{-2.2568,-3.2874},{0.6467,-1.9529},{-5.6635,1.1940},{0.5533,2.2265},{0.4039,2.3276},{3.6249,2.1594},{1.1624,-4.4032},{1.5364,-0.5207},{-4.1599,-3.5487},{-1.9733,2.0099},{-0.7060,1.7409},{0.1711,4.2395},{4.5573,-2.0715},{1.0102,0.5416},{0.2351,-4.9960},{-3.2654,-0.7011},{-1.3025,1.2220},{-3.3778,2.4776},{4.7253,2.4405},{0.3874,0.6025},{3.5955,-2.5563},{-1.3768,-

3.5879},{-2.0473,0.4993},{-3.8250,-
1.3042},{0.9719,5.3906},{0.3890,0.5866},{3.5860,1.0160},{2.4789,-3.5155},{-
2.3816,-1.1278},{-1.1217,-3.6190},{-
3.7228,4.0216},{0.2082,1.2085},{1.2264,2.7203},{4.6895,0.0594},{-1.0500,-
3.0185},{1.9808,-2.8000},{-5.2997,-0.5812},{-1.0691,1.5979},{-
0.8644,2.0533},{2.8110,4.1144},{1.8472,-3.2079},{2.9654,-0.2242},{-2.5113,-
4.3742},{-2.6444,0.3140},{-1.3671,0.6898},{-1.7654,4.8478},{4.0617,-
0.6272},{1.8036,1.6595},{1.9164,-4.2211},{-2.4111,-2.3909},{-
1.0562,0.0357},{-4.9945,1.4539},{3.1672,3.0545},{0.2286,1.9168},{4.0562,-
0.7811},{0.5076,-4.0514},{-1.0558,-0.2094},{-4.0799,-3.0613},{-
0.6194,4.5245},{-0.6133,1.4463},{2.5487,2.4786},{4.0593,-2.3548},{-1.1210,-
0.9750},{-0.0660,-4.8530},{-4.0939,2.1274},{-1.1438,1.1381},{-
0.4969,2.8946},{4.7918,2.0087},{-0.1750,-1.9733},{3.4492,-2.8391},{-4.0157,-
2.1456},{-2.0099,0.5458},{-2.0988,0.9468},{1.3853,5.3920},{1.7309,-
1.7234},{3.8962,0.7142},{-0.4237,-4.3817},{-2.5115,-1.1597},{-1.4202,-
0.8895},{-3.5899,4.5399},{2.8045,0.4174},{1.7912,2.8539},{3.2774,-
2.5982},{-1.1643,-3.3073},{-0.0305,-1.0754},{-5.8446,-
0.0909},{1.3961,2.8278},{-0.5433,2.7001},{3.6925,1.3014},{2.0694,-
3.6750},{0.4090,-0.3142},{-3.4381,-4.5899},{-1.7183,2.9384},{-
1.6591,1.4884},{0.7631,3.5385},{4.7705,-0.8637},{0.0286,-0.0720},{1.4155,-
5.2727},{-3.5971,0.1324},{-1.9286,0.4581},{-
2.4071,2.2722},{4.0470,3.5164},{0.0788,-0.4963},{4.6060,-2.0244},{-2.0919,-
3.1187},{-2.0982,-0.5213},{-2.8814,-0.8043},{-0.2766,5.7240},{0.8719,-
0.4021},{4.0000,2.0000},{1.6281,-3.5258},{-1.7619,-2.0100},{-0.7207,-
2.5619},{-4.8563,3.4183},{1.1643,0.7954},{0.9694,3.6901},{3.9648,-
0.4518},{0.0518,-3.3202},{1.5532,-1.7095},{-5.7110,-1.7593},{-
0.1289,1.8605},{-1.6134,2.6600},{2.5089,3.2102},{2.9284,-
2.6869},{1.9008,0.2963},{-2.0512,-5.4691},{-2.0860,1.0447},{-
2.3240,0.7482},{-1.4113,3.8760},{4.5303,0.5216},{0.6798,1.3299},{2.9126,-
4.7142},{-2.3783,-1.5034},{-1.8879,-0.4960},{-
4.0268,0.9036},{2.7275,4.2315},{-0.3695,0.9548},{5.0616,-0.5507},{0.0286,-
3.2974},{-1.4086,-1.1870},{-2.9282,-2.8154},{-1.7311,5.0829},{-
0.4482,0.3818},{3.1640,3.2182},{3.1933,-1.9997},{-0.7547,-2.0126},{0.6415,-
3.8606},{-5.2517,1.7974},{-0.3893,0.4576},{-
0.4309,3.8110},{3.8189,1.7525},{0.8282,-2.5470},{3.2934,-1.6165},{-4.6241,-
3.1097},{-1.0491,0.4667},{-2.5654,1.7301},{0.7429,4.5142},{2.8982,-
1.4729},{2.9652,1.4996},{-0.2701,-5.4316},{-1.7099,-0.6761},{-2.2814,-
0.5116},{-3.4973,3.3778},{3.5041,1.4018},{0.6127,2.8164},{3.9948,-
3.2644},{-0.7858,-2.4433},{-0.9972,-1.3067},{-4.9536,-
0.9361},{1.2457,4.0118},{-1.3680,1.9380},{4.5971,1.2035},{1.8598,-
2.7122},{-0.2346,-1.1522},{-2.1715,-4.6080},{-2.5953,3.6857},{-
1.8033,0.4373},{1.5259,3.9562},{3.9664,-0.1846},{0.0886,-1.1838},{2.3630,-
4.4164},{-4.6851,0.1123},{-1.4256,-0.4145},{-
2.0117,3.0453},{2.9169,3.5325},{0.8914,-1.3141},{4.7188,-0.7701},{-2.8524,-

3.7838},{-1.2043,-0.9141},{-3.0073,0.0913},{-1.1970,4.9483},{2.0381,-0.4601},{3.2673,2.9713},{1.4558,-4.4417},{-0.8046,-1.8067},{-1.4029,-1.8792},{-5.0275,2.1687},{2.0541,1.5315},{-0.1624,3.9270},{4.3366,-1.2379},{0.7272,-2.5632},{0.5250,-1.6011},{-4.9692,-2.8205},{0.0329,2.9641},{-2.5759,2.1503},{3.2325,2.7843},{3.0040,-1.6146},{0.9996,-0.3263},{-0.7796,-5.7402},{-2.6572,1.9312},{-2.7432,-0.1920},{-0.5620,3.9261},{3.8668,1.4559},{0.4121,0.2234},{4.0093,-4.0600},{-3.3146,-1.2071},{-1.6726,-1.4638},{-3.3232,1.4633},{1.5489,4.5188},{0.1861,-0.0592},{5.4187,0.6248},{-0.8278,-3.6106},{-0.6718,-1.8414},{-2.6896,-1.8754},{-2.8441,4.4826},{0.6363,0.0102},{2.6845,4.2767},{2.7067,-2.7100},{0.2539,-2.1072},{0.1979,-2.9136},{-5.6710,0.5750},{0.6330,0.8916},{-1.4107,4.2845},{3.8137,0.9077},{1.7301,-1.9788},{2.2769,-1.1652},{-4.0975,-4.2896},{-0.5858,1.4142},{-3.5627,1.5122},{1.2266,3.7914},{3.2580,-0.4014},{1.8634,1.1408},{0.8918,-5.9272},{-1.9385,0.2862},{-2.9232,-1.2454},{-2.6271,3.0524},{3.0542,2.5051},{0.0321,1.7899},{5.1369,-2.8675},{-1.5054,-1.8576},{-1.0903,-2.2635},{-3.9922,-0.6352},{0.1326,4.5530},{-1.1051,0.7957},{5.1588,2.1916},{0.9739,-2.6573},{0.2618,-2.0002},{-1.5831,-3.6930},{-3.8029,3.3243},{-0.8710,-0.2172},{1.3404,4.9965},{3.2099,-0.6385},{1.0410,-1.5774},{2.1899,-3.2726},{-5.3219,-0.9700},{-0.3447,-0.3025},{-2.7425,3.7037},{2.5420,2.6971},{1.9376,-1.0052},{3.7843,-0.0102},{-2.5933,-4.9786},{-0.4848,-0.1834},{-3.9351,0.1879},{-0.9882,3.9908},{2.6649,0.5028},{2.0443,2.8948},{2.4019,-5.1206},{-0.6896,-0.8442},{-2.2034,-2.3248},{-4.2056,1.4985},{1.8756,2.7104},{-1.0052,3.0475},{5.4203,-1.1370},{0.2707,-1.7481},{0.1210,-2.4458},{-3.8284,-2.7980},{-0.9098,3.7271},{-2.6098,0.9658},{3.9477,3.4914},{2.1631,-1.2123},{1.1904,-1.2921},{0.1072,-4.9189},{-3.8605,1.8537},{-2.0220,-1.0651},{-0.4322,4.8485},{2.9152,1.2879},{1.2105,-0.4521},{4.1116,-2.8100},{-4.1257,-2.0517},{-0.6200,-1.6613},{-3.7285,2.2467},{0.8478,3.7654},{1.2910,-0.0584},{4.6318,1.6248},{-0.8674,-4.7217},{0.2278,-1.3729},{-3.4551,-1.4628},{-2.9222,3.3789},{1.4963,0.7724},{1.4379,4.4737},{3.3521,-3.5205},{0.6810,-1.2283},{-0.6923,-3.0144},{-4.9671,-0.3756},{0.7596,2.0551},{-2.4344,3.6105},{4.7461,0.6978},{1.5649,-1.0215},{1.5806,-1.8126},{-2.8772,-4.5413},{-1.2745,2.3528},{-3.8698,0.3838},{2.0373,4.1507},{2.5408,0.2960},{1.7116,0.1340},{1.9951,-5.2651},{-3.0465,0.5116},{-2.4592,-2.2467},{-2.1856,3.7739},{2.0050,2.6323},{0.6006,0.8694},{5.4963,-1.6181},{-2.4359,-2.3940},{-0.1597,-2.7311},{-4.0267,0.2089},{-0.8228,3.9541},{-0.0234,0.4695},{4.5780,3.3375},{0.6323,-3.5999},{1.2437,-1.8235},{-2.1145,-2.9876},{-4.1597,2.1809},{0.1711,0.2784},{0.1776,5.4351},{3.5018,-1.5206},{1.7231,-0.8678},{1.2797,-3.0053},{-4.8019,-2.1120},{0.0894,0.7631},{-3.8455,3.2827},{3.2494,2.1899},{2.0742,-

```
0.0095},{2.8379,-0.4010},{-1.4053,-5.4808},{-0.8665,0.8701},{-4.4704,-
0.7825},{-0.1456,3.9710},{2.1475,1.4207},{1.5478,1.9207},{3.6205,-
4.6756},{-1.6254,-0.3290},{-2.0287,-3.3488},{-
3.4842,1.9592},{0.8378,3.1244},{-0.7139,1.9400}};

   IO_LED = 0x55aa66aa;
   fft(x, (complex) {1.0000, 1.0000});
   IO_LED = 8;
   while (1) {}
   return 0;
}
```

Приложение 2:

```
`include "coproc_if.svh"

module coproc_custom0_wrapper (
input logic unsigned [0:0] clk_i
, input logic unsigned [0:0] rst_i
, output logic unsigned [0:0] stream_resp_bus_genfifo_req_o
, output resp_struct stream_resp_bus_genfifo_wdata_bo
, input logic unsigned [0:0] stream_resp_bus_genfifo_ack_i
, input logic unsigned [0:0] stream_req_bus_genfifo_req_i
, input req_struct stream_req_bus_genfifo_rdata_bi
, output logic unsigned [0:0] stream_req_bus_genfifo_ack_o
);

assign stream_req_bus_genfifo_ack_o = stream_req_bus_genfifo_req_i;
logic unsigned [31:0] ste1_res, ste2_res, ste3_res, ste4_res, x_1_real, x_1_img,
x_2_real, x_2_img, resp, a_real, a_img, b_real, b_img, c_real, c_img;
assign stream_resp_bus_genfifo_wdata_bo = resp;

always @(posedge clk_i)
   begin
      if (rst_i)
         begin
         stream_resp_bus_genfifo_req_o <= 1'b0;
         ste1_res <= 0;
         ste2_res <= 0;
         ste3_res <= 0;
         ste4_res <= 0;
         x_1_real <= 0;
         x_1_img <= 0;
         x_2_real <= 0;
         x_2_img <= 0;
         resp <= 0;
```

```
        a_real<= 0;
        a_img <= 0;
        b_real <= 0;
        b_img <= 0;
        c_real <= 0;
        c_img <= 0;
        end
      else
        begin
        stream_resp_bus_genfifo_req_o <= 1'b0;
        if (stream_req_bus_genfifo_rdata_bi.instr_code[1:0] == 0)
          begin
          resp <= x_1_img;
          a_real <= stream_req_bus_genfifo_rdata_bi.src0_data;
          b_real <= stream_req_bus_genfifo_rdata_bi.src1_data;
          ste1_res <=
stream_req_bus_genfifo_rdata_bi.src0_data*stream_req_bus_genfifo_rdata_bi.sr
c1_data;
          end
        else if(stream_req_bus_genfifo_rdata_bi.instr_code[1:0] == 1)
          begin
          resp <= x_2_img;
          a_img <= stream_req_bus_genfifo_rdata_bi.src0_data;
          b_img <= stream_req_bus_genfifo_rdata_bi.src1_data;
          ste2_res <=
stream_req_bus_genfifo_rdata_bi.src0_data*stream_req_bus_genfifo_rdata_bi.sr
c1_data;
          end
        else if(stream_req_bus_genfifo_rdata_bi.instr_code[1:0] == 2)
          begin
          c_real <= stream_req_bus_genfifo_rdata_bi.src0_data;
          c_img <= stream_req_bus_genfifo_rdata_bi.src1_data;
          ste3_res <= a_real*b_img;
          x_1_real <= c_real + ste1_res - ste2_res;
          x_2_real <= c_real - ste1_res + ste2_res;
          resp <= x_1_real;
          end
        else if(stream_req_bus_genfifo_rdata_bi.instr_code[1:0] == 3)
          begin
          ste4_res <= b_real*a_img;
          x_1_img <= c_img + ste1_res + ste2_res;
          x_2_img <= c_img - ste1_res - ste2_res;
          resp <= x_2_real;
          end
```

```
        end
      end
   end
endmodule
```

Приложение 3:

```
module FindMaxVal_pipelined (
   input clk_i
   , input rst_i

   , input [31:0] num_bi [5:0]

   , output logic [31:0] x_i [3:0]
);

//// stage 0 ////
// intermediate signals declaration
logic [31:0] inputs0_next [5:0];
logic [31:0] results0 [5:0];
// combinational logic
always @*
   begin
   results0 [0] = num_bi[0] * num_bi[1];
   results0 [1] = num_bi[2] * num_bi[3];
   results0 [2] = num_bi[0] * num_bi[3];
   results0 [3] = num_bi[1] * num_bi[2];
   results0 [4] = num_bi[4];
   results0 [5] = num_bi[5];`
   end
// writing to registers
always @(posedge clk_i)
   begin
   if (rst_i)
      begin
      for (integer i=0; i<6; i++) inputs0_next[i] <= 0;
      end
   else
```

```
        begin
        for (integer i=0; i<6; i++) inputs0_next[i] <= results0[i];
        end
    end
//// stage 1 ////
// intermediate signals declaration
logic [31:0] inputs1_next [3:0];
logic [31:0] results1 [3:0];
// combinational logic
always @*
    begin
    results1[0] = inputs0_next[0] - inputs0_next[1];
    results1[1] = inputs0_next[2] + inputs0_next[3];
    results1[2] = inputs0_next[4];
    results1[3] = inputs0_next[5];
    end
// writing to registers
always @(posedge clk_i)
    begin
    if (rst_i)
        begin
        for (integer i=0; i<4; i++) inputs1_next[i] <= 0;
        end
    else
        begin
        for (integer i=0; i<4; i++) inputs1_next[i] <= results1[i];
        end
    end
//// stage 2 ////
// intermediate signals declaration
logic [31:0] results2 [3:0];
// combinational logic
always @*
    begin
    results2[0] = inputs1_next[0] + inputs1_next[2];
    results2[1] = inputs1_next[2] - inputs1_next[0];
    results2[2] = inputs1_next[1] + inputs1_next[3];
    results2[3] = inputs1_next[3] - inputs1_next[1];
    end
// writing to registers
always @(posedge clk_i)
    begin
    if (rst_i)
        begin
```

```
        for (integer i=0; i<4; i++) x_i[i] <= 0;
        end
    else
        begin
        for (integer i=0; i<4; i++) x_i[i] <= results2[i];
        end
    end
endmodule
```