Министерство образования и науки Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(Национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Инфокоммуникационные технологии»

	Допустить к защите
Рецензент	Руководитель направления
_____	_____ Ю.Т. Карманов
“____” _____2017г.	“____”_____2017г.

Оптимизация алгоритма БПФ при реализации на цифровых сигнальных процессорах

Направление **11.04.02 «Инфокоммуникационные технологии и системы связи»**
*магистерская программа***«Системы мобильной связи»**
ЮУрГУ – М 11.04.02.2017.051.00 ПЗ

Научный руководитель:
_____А.Н. Николаев
“____” _____2017г.

Магистрант:
студент группы КЭ-272
_____П.Ю. Калита
“____”_____2017г.

Нормоконтролер:
_____В.Д. Спицына
“____” _____2017г.

Челябинск 2017

`

РЕФЕРАТ

Калита П.Ю. Оптимизация алгоритма БПФ при реализации на цифровых сигнальных процессорах. – Челябинск: ЮУрГУ, ВШЭиКН, 2017, 77 с., 34 илл., 3 табл., 6 прил., библиогр. список – 55 наим.

Ключевые слова: быстрое преобразование Фурье, цифровые сигнальные процессоры, цифровая обработка сигналов, алгоритмы и графы БПФ, время вычисления БПФ.

Объектами исследования являются алгоритмы вычислительных графов быстрого преобразования Фурье. Цель работы выявление оптимального алгоритма для конкретного сигнального процессора. Исследование проведено при помощи моделирования в системе Matlab и последующей реализации в режиме симулирования в системе VisualDSP. По результатам работы получены временные зависимости быстродействия алгоритма в зависимости от числа отсчетов сигнала и в зависимости от рассматриваемого вычислительного графа.

Результаты данного исследования могут быть применены при проектировании систем приема, обработки и передачи сигналов. Итог данной работы предполагает использование более эффективных алгоритмов БПФ по сравнению ссуществующими.

В ходе выполнения работы были использованы следующие программные продукты: Matlab, VisualDSP, MSOffice.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ……...……6
1 Обзор источников……..……..……8
2 Моделирование в среде Matlab…….…..15
2.1Дискретное преобразование Фурье…….....…...15
2.2 Алгоритм БПФ с прореживанием повремени……..…..……...…18
2.3 Алгоритм БПФ с прореживанием по частоте……...…….……28
2.4 Алгоритм БПФ по основанию 2 с постоянной структурой, без замещения, с нормальным порядком отсчетов на входе и двоично-инверсным на выходе…….……36
2.5 Алгоритм БПФ по основанию 2 с постоянной структурой, без замещения, с двоичной инверсией на входе и прямым порядком на выходе…….…….....39
2.6 Алгоритм БПФ по основанию 2 с прореживанием по времени, без замещения, с нормальным порядком на входе и выходе……...……42
3 Моделирование в среде разработки VisualDSP……..…..……45
3.1 Реализация алгоритмов на процессоре ADSP-BF-537….....45
3.2 Реализация алгоритмов на процессоре ADSP-TS-201……..….52
3.3 Оценка эффектов квантования в алгоритмах БПФ…….....…..56
ЗАКЛЮЧЕНИЕ..…….63
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....……..……...…….…65
ПРИЛОЖЕНИЯ
Приложение А. Фрагмент листинга программы, реализующей алгоритм Кули-Тьюки на процессоре ADSP-BF-537……...……68
Приложение Б. Фрагмент листинга программы, реализующей алгоритм с постоянной структурой на процессоре ADSP-BF-537…...…71

Приложение В. Фрагмент листинга программы, реализующей алгоритм без использования бит-реверсивной перестановки на процессоре ADSP-BF-537……..…….72
Приложение Г. Фрагмент листинга программы, реализующей алгоритм Кули-Тьюки на процессоре ADSP-TS-201…….……..…….73
Приложение Д. Фрагмент листинга программы, реализующей алгоритм с постоянной структурой на процессоре ADSP-TS-201….……74
Приложение Е. Фрагмент листинга программы, реализующей алгоритм без использования бит-реверсивной перестановки на сигнальном процессоре ADSP-TS-201……...…..76

ВВЕДЕНИЕ

Быстрое преобразование Фурье является одним из ключевых средств цифровой обработки сигналов. В связи с этим особо важным моментом является повышение вычислительной способности (оптимизации) алгоритма БПФ. В настоящее время происходит непрерывное развитие микропроцессорной техники, повышается ее производительность и энергоэффективность, однако, несмотря на это от разработчика программного обеспечения требуется написание оптимального и оптимизированного кода для решения конкретных задач и в тоже время наиболее емко использующего возможности конкретного цифрового сигнального процессора. На данный момент в открытых источниках не удалось найти однозначного решения этой задачи, что подчеркивает актуальность темы исследования.

Реализация данного исследования поможет найти оптимальные алгоритмы вычислительных графов быстрого преобразования Фурье для конкретных исследуемых моделей сигнальных процессоров.

Объектом исследования является быстродействие нестандартных вычислительных алгоритмов БПФ. Предмет исследования алгоритмы быстрого преобразования Фурье.

Целью исследования является выявление оптимального алгоритма БПФ для конкретного сигнального процессора. Для реализации данной цели необходимо решить следующие задачи:

– изучить и проанализировать статьи и литературу по данной теме;

– смоделировать исследуемые вычислительные графы БПФ средствами Matlab;

– провести оценку времени вычисления алгоритма в зависимости от количества отсчетов сигнала;

– оценить время вычисления различных алгоритмов с фиксированным количеством отсчетов сигнала;

– рассмотреть влияние шумов квантования.

Инструмент быстрого преобразования Фурье давно детально изучен и внедрен во все сферы ЦОС, однако, мало внимания уделено внутренней структуре алгоритма, в связи с этим есть предположение, что различные графы БПФ имеют различную производительность на различных устройствах, в ходе выполнения работы данная гипотеза будет проверена.

Основными этапами работы является изучение и анализ существующих наработок в данной сфере, моделирование алгоритмов БПФ в среде моделирования Matlab, с целью проверки работоспособности данных алгоритмов, и далее реализация и моделирование в среде VisualDSPи исследования разработанных алгоритмов непосредственно на изучаемых сигнальных процессорах.

Данное исследование можно считать теоретически значимым, так полученные результаты можно использовать для исследования других алгоритмов БПФ, а так же применять их для различных цифровых сигнальных процессоров.

Практическая значимость работы предполагает использование полученных результатов при проектировании систем цифровой обработки сигналов, а так же устройств приема и передачи сигналов.

В ходе работы основными источниками информации служили учебные пособия по цифровой обработке сигналов и научные статьи в аналогичной сфере исследования, которые будут рассмотрены и проанализированы более подробно ниже.

1 Обзор источников

В области цифровой обработки сигналов известно большое число способов оптимизации и повышения быстродействия дискретного преобразования Фурье. Одним из самых важных из них является алгоритм быстрого преобразования Фурье. Однако в ряде случаев требуется оптимизация и самого алгоритма быстрого преобразования. Среди причин можно выделить великое множество семейств цифровых сигнальных процессоров, в силу своих особенностей не удовлетворяют по параметру быстродействия стандартным алгоритмам БПФ. В научном сообществе существует множество таких способов. Рассмотрим наиболее полезные и яркие из них.

В Чувашском университете разработан программный комплекс моделирования быстрого преобразования Фурье, который включает программу генерации сигнала, его аналого-цифровое преобразование и модули, которые реализуют вычисление преобразования Фурье в позиционной и непозиционных системах счисления. Проведена оценка временных и аппаратурных затрат [1].

Реализованы оба алгоритма были на сигнальных процессорах семейства TigerShark 101 от компании AnalogDevices.

Разработанная программабыла реализована в программном комплексеVisualDSP++ и предназначена для оценки затрат памяти и времени при вычислении БПФ с использованием средств непозиционной и позиционной арифметики.

Исследованы следующие возможные случаи:

– вычисление БПФ в позиционной системе счисления без учёта возможного переполнения в результате выполненной операции;

– вычисление БПФ в позиционной системе счисления с учётом возможного переполнения в результате выполненной операции;

– вычисление БПФ в системе остаточных классов с оптимизацией по времени;

– вычисление БПФ в системе остаточных классов с оптимизацией по памяти.

Оценка временных затрат проводилась путём получения значения процессорного счётчика циклов соответственно до и после вычисления БПФ указанным методом, после чего вычислялась разность между этими двумя значениями. На рисунке 1 представлены результаты испытаний системы.

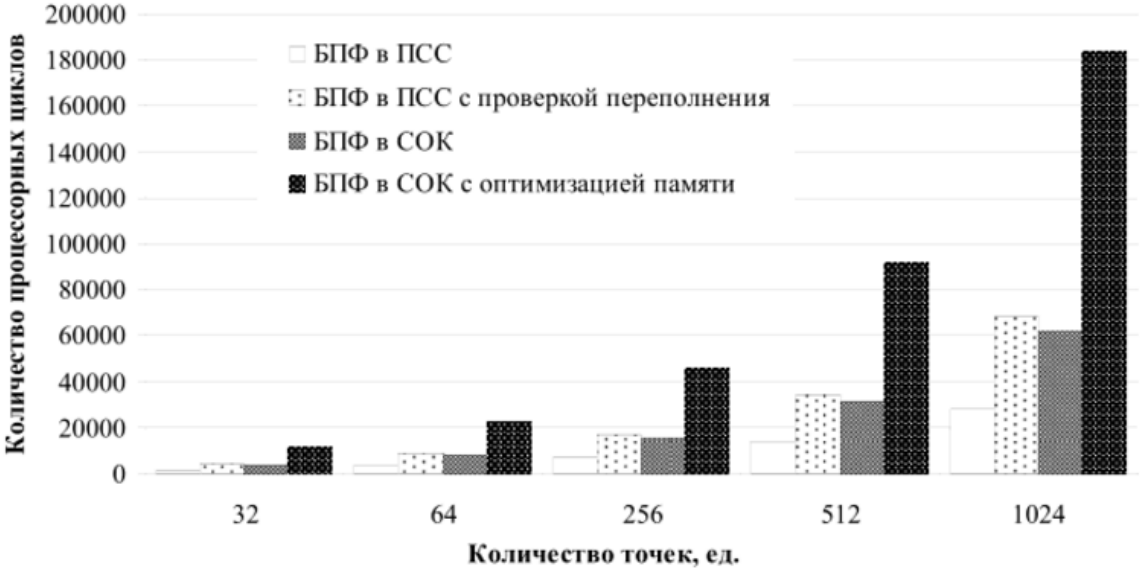


Рисунок 1 – Результаты испытаний методов вычисления БПФ

Реализация вычисления БПФ в СОК, как можно увидеть из рисунка 1, выигрывает в скорости вычислений у реализации вычисления БПФ в ПСС с проверкой переполнения регистров. Следует заметить, что наименьшее количество циклов затрачивается на вычисление БПФ в ПСС, но без осуществления проверок переполнения, что оправдано только при очень малом динамическом диапазоне сигнала и не подходит для наиболее распространенных практических случаев. Необходимо отметить, что специфика вычисления БПФ в СОК требует некоторого увеличения аппаратных затрат, хотя наличие двух вычислительных блоков в процессоре ADSPTS-101 позволяет организовать одновременное вычисление в двух каналах БПФ в СОК.

Следующим шагом имеет место показать оптимальную реализацию быстрого преобразования Фурье на процессоре NVCOM-01.

В статье Омского Вестника описывается реализация алгоритма вычисления БПФ для сигнального процессора NVCom-01. Высокая вычислительная производительность была достигнута благодаря максимально полному использованию архитектуры процессора, а так же правильного упорядочивания команд с целью предотвращения блокировок конвейера и использования некоторых особенностей алгоритма. В данной работе так же приведен расчет теоретического быстродействия процессора NVCom-01 при вычислении быстрого преобразования Фурье [2].

Алгоритм проверялся на отладочной плате с процессором NVCom-01 при использовании режима отладки среды MCStudio3.1. Расчет БПФ на реальном процессоре потребовал 8367 тактов для 1024 чисел формата 32+j32. Увеличение времени по сравнению с максимальным результатом происходит из-за наличия дополнительных инициализирующих операций и сбивания конвейера при входе в цикл. Этот результат примерно на 2500 тактов быстрее, чем заявленная в документации скорость вычислений БПФ для процессора NVCom-01. Кроме того, полученный результат гораздо лучше, чем и у процессоров производства TexasInstruments и AnalogDevices. Расчетная часть программы занимает 62 команды.

Но предложенный алгоритм имеет и существенный недостаток — это требования к свободной памяти. Помимо того, что требуется, по крайней мере, один дополнительный буфер размером 1024×8 байт для хранения вычисленных значений, также требуется буфер примерно такого же объема для хранения массива поворачивающих коэффициентов. В предложенном алгоритме они располагаются последовательно друг за другом для каждой из 1024 «бабочек». Конечно, можно сделать и повторное использование поворачивающих коэффициентов, но это приведет к увеличению команд и, соответственно, уменьшению скорости выполнения расчета. Так что в каждом конкретном случае разработчик должен решать, что важнее — используемая память или скорость вычисления.

Альтман Евгений Анатольевичиз Омского государственного университета рассматривает способы оптимизации вычислительной схемы быстрого преобразования Фурье[3].

В его работе предлагается метод повышения быстродействия функции быстрого преобразования Фурье, которая основана на использовании дополнительного буфера данных с целью ускорения доступа к памяти и уменьшения количества операций над указателями.

Алгоритмы Кули-Тьюки позволяют организовать вычислительный процесс таким образом, чтобы на каждой стадии вычислений результаты размещались в тех же ячейках памяти, что и входные данные (так называемые вычисления «на месте», in-place). Эта схема оказалась удобной в реализации и получила широкое распространение.

Вместе с тем вычисление БПФ «на месте» обладает рядом недостатков. Основной недостаток связан с тем, что на каждой стадии данные из памяти нужно считывать не последовательно, а в определенном порядке. В некоторых вычислительных системах последовательное считывание данных из памяти выполняется быстрее, чем при произвольном доступе к памяти; кроме того, для произвольного доступа требуется использование адресной арифметики.

Для увеличения быстродействия БПФ автором предложена следующая идея: вычислительный процесс нужно организовать таким образом, чтобы на каждой стадии данные для обработки можно было бы считывать из памяти последовательно. Для этого на предыдущей стадии результаты вычислений нужно сохранять в требуемом порядке.

При реализации этой идеи данные на различных стадиях вычислений будут расположены различным образом, т.е. организовать вычисления «на месте» при данном подходе не получится. Поэтому для хранения данных нужно использовать два буфера. В текущей стадии один из них используется для входных данных, другой — для выходных, на следующей стадии они меняются местами. В русском языке для обозначения такого подхода часто применяется термин «пинг-понг», на

английском, по аналогии с обычной схемой, более удачным будет название «out-place».

Рассмотренный алгоритм был реализован на процессоре ZSP400 фирмы LSI Logic. Время выполнения преобразования Фурье для 128 точек (вход — комплексные 16-битные целые числа в формате Q15 нормированные на половину диапазона, для предотвращения переполнения после каждой стадии данные делятся на два) включая сохранение контекста составило 1900 тактов (менее 15 тактов на точку). Этот результат на 20% быстрее наилучшего из известных результатов (2400 тактов, менее 19 тактов на точку) для алгоритмов, выполняющих вычисления «на месте» [3].

Таким образом, алгоритмы типа «пинг-понг» позволяют получать в общем случае, более быстрые и простые функции для вычисления спектров сигналов. Они были использованы при разработке функций БПФ математической библиотеки для процессоров на ядрах ZSP. Разработанные функции имеют производительность на 10-20% выше оценочных значений производительности классических алгоритмов.

Группа инженеров Южного федерального университета провела важное исследование ошибок усечения результатов арифметических операций с фиксированной точкой в алгоритмах быстрого преобразования Фурье [4].

В своей работе они рассмотрели ошибки вычисления быстрого преобразования Фурье (БПФ) с прореживанием по частоте в отсутствии масштабирования.Они обусловленны ограничением разрядности результатов арифметических операций умножения при обработке данных с фиксированной точкой в зависимости от размера преобразования, номера спектральной составляющей, а также от способа приведения результата к системной длине слова (усечение или округление данных). Получены аналитические выражения среднеквадратического значения ошибок вычисления БПФ белого гауссовского шума, как при усечении, так и при округлении результатов операций умножения в алгоритме БПФ с прореживанием по частоте в отсутствии масштабирования.

Было показано, что в области низких частот для приближенной оценки уровня шумов, вызванных усечением разрядности результатов арифметических операций, можно использовать зависимость модуля математического ожидания ошибок усечения от номера спектрального отсчета. Поставлен математический эксперимент по вычислению погрешности арифметических операций БПФ для входной последовательности в виде белого гауссовского шума. Приведены графики среднеквадратических значений ошибок усечения и округления в зависимости от номера спектральной составляющей. Показано, что зависимость среднеквадратического значения ошибок усечения арифметических операций от номера спектральной составляющей носит асимметричный характер и приводит к значительным искажениям спектра в области низких частот. Моделирование показало совпадение уровней ошибок, полученных экспериментальным и аналитическим путём.

Поиск и анализ рассмотренных статей, найденных в открытых источниках и сети интернет, дал понять, что оптимизации алгоритмов БПФ уделено достаточно немного внимания. И в тоже время не найдено ни одного упоминания о реализации различных вычислительных графов БПФ. В ходе выполнения данной работы попытаемся восполнить данный пробел.

За основу исследования взяты пять алгоритмов быстрого преобразования Фурье из учебного пособия Л. Рабинера и Б. Гоулда:

– алгоритм быстрого преобразования Фурье с прореживанием по времени;

– алгоритм быстрого преобразования Фурье с прореживанием по частоте;

– алгоритм БПФ по основанию 2 с прореживанием по времени, без замещения, с нормальным порядком на входе и выходе;

– алгоритм БПФ по основанию 2 с постоянной структурой, без замещения, с нормальным порядком отсчетов на входе и двоично-инверсным на выходе;

– алгоритм БПФ по основанию 2 с постоянной структурой, без замещения, с двоичной инверсией на входе и прямым порядком на выходе.

В основе всех выбранных алгоритмов лежит элементарная операция, называемая «бабочкой БПФ». Различие алгоритмов состоит лишь в различной группировке отсчетов сигнала после выполнения каждой стадии алгоритма.

В следующей главе будут рассмотрены некоторые особенности исследуемых алгоритмов, их модели, реализованные в MATLAB, а так же будет дана оценка их эффективности, трудоемкость программной реализации, их достоинства и недостатки.

2 Моделирование в среде Matlab

2.1 Дискретное преобразование Фурье

Обработка сигналов, связанная с анализом их спектров, называется спектральным анализом. Спектральный анализ используется во многих алгоритмах,в частности, при распознавании, обнаружении и сжатии сигналов. Математической основой спектрального анализа является ДПФ:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, k = 1,2, \dots, N-1,$$

где N – длина исходной последовательности и размерность ДПФ; для краткости последовательность и ДПФ называют N-точечными;

W_N^{kn} – поворачивающий множитель:

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

Оценим вычислительную сложность алгоритма ДПФ: при фиксированном значении k и $n=0, 1,\dots, (N-1)$ требуется выполнить N операций умножения и$(N-1) \approx N$операций сложения, всего $2N$ операций; в целом же, при $k=0, 1,\dots, (N-1)$ необходимо выполнить $N \times 2N = 2N^2$ арифметических операций с комплексными числами.

Как правило, представляет интерес оценка порядка вычислительной сложности алгоритма относительно длины N, для чего используют символику $O(\cdot)$. Порядок вычислительной сложности алгоритма ДПФ оценивается как $O(N^2)$, что при больших N весьма существенно.

В реальном времени обработка исходных последовательностей выполняется по частям (сегментам) длины N . Численное значение N зависит от физической природы сигнала и решаемой задачи. Для снижения порядка вычислительной сложности были разработаны алгоритмы быстрого вычисления ДПФ.

Ниже награфике на рисунке 2 приведен спектр сигнала, вычисленный с помощью прямого дискретного преобразования Фурье, для моделирования использовался программный продукт Matlab. В качестве сигнала использовался сигнал s=sin(2π450t), частота сигнала 450 Гц.

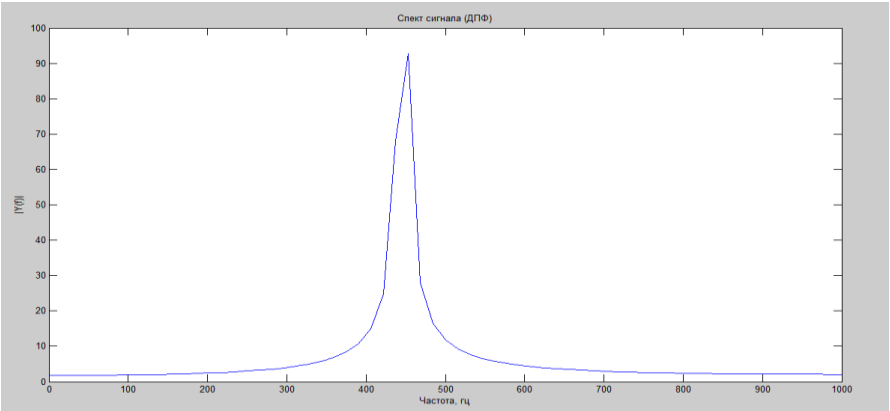


Рисунок 2 – Спектр сигнала ДПФ (128 отсчетов)

На рисунке 3 представлена блок схема программы по вычислению прямого дискретного преобразования Фурье.

Имея комплексный сигнал, получаем $2 \times N \times N$ вещественных умножений

2.2 Алгоритм быстрого преобразования Фурье с прореживанием по времени

Быстрым преобразованием Фурье (БПФ) называют не еще одну разновидность преобразования Фурье, а целый набор алгоритмов, предназначенных для быстрого вычисления ДПФ. Поскольку подобных алгоритмов много, для их обозначений после аббревиатуры БПФ ставят уточнение: алгоритм БПФ с основанием 2, алгоритм БПФ с основанием 4, алгоритм БПФ Винограда и т. п.

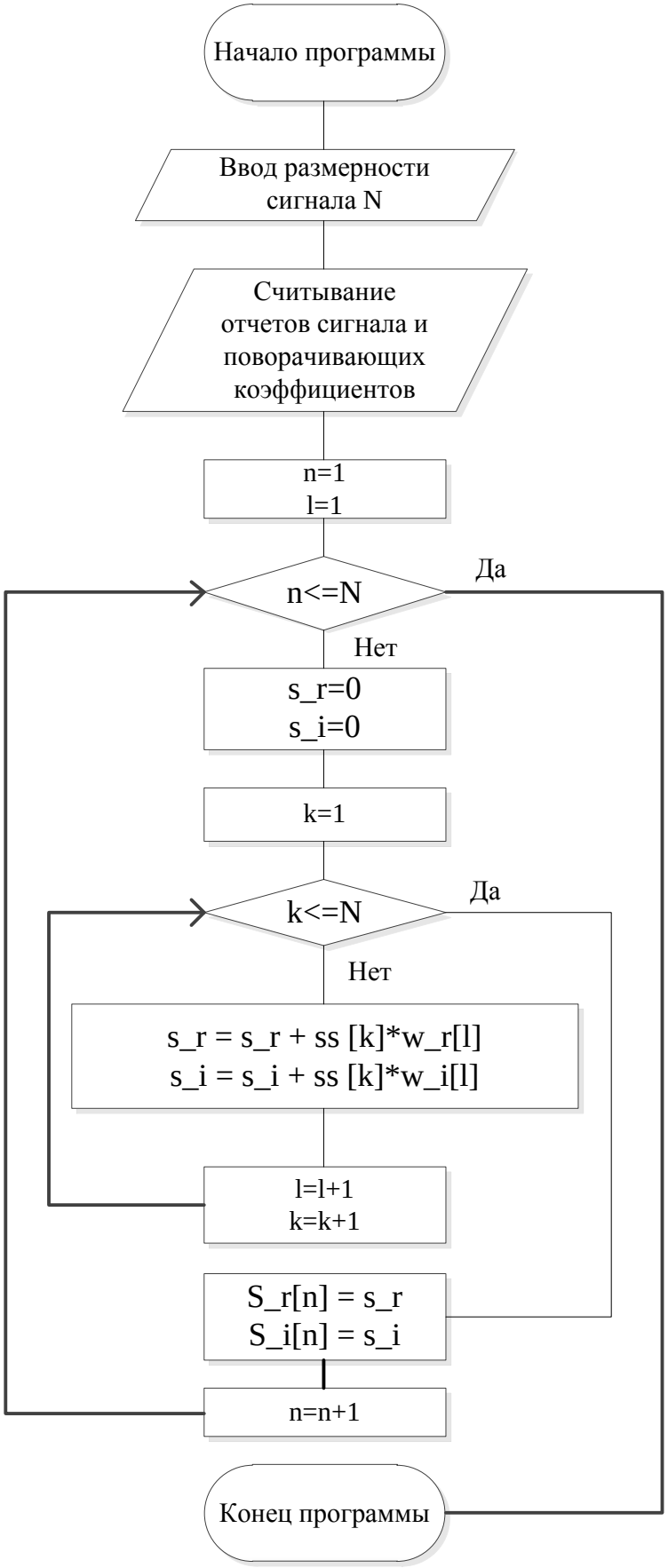


Рисунок 3 – Блок схема программы ДПФ

Наибольшее распространение получил алгоритм БПФ с основанием 2, известный как алгоритм БПФ Кули–Тьюки (по имени разработчиков) и впервые опубликованный в 1965 году в США. Существуют два эквивалентных по эффективности алгоритма БПФ с основанием 2: с прореживанием по времени и с прореживанием по частоте. В любом из них длина N исходной последовательности должна быть равной:

$$N = 2^{\nu},$$

где ν – целое положительное число.

Если условие не выполняется, исходную последовательность следует дополнить необходимым количеством нулей. Отметим, что это является недостатком данных алгоритмов, поскольку количество нулей может оказаться значительным.

Основная идея алгоритма БПФ с прореживанием по времени заключается в поэтапном вычислении N-точечного ДПФ на ν этапах, на каждом из которых текущее ДПФ определяется как комбинация ДПФ вдвое меньшей размерности.

Алгоритм БПФ с прореживанием по времени можно описать следующим образом (рисунок4, снизу вверх):

- задание начальных условий:
  отсчеты N-точечной последовательности расставляются по определенному правилу;
- на первом этапе определяется 2-точечное ДПФ каждой пары отсчетов последовательности;
- на втором этапе определяются 4-точечные ДПФ как комбинация 2-точечных ДПФ;
- на i-*ом* этапе определяются 2^{i}-*точечные* ДПФ как комбинация 2^{i-1}-*точечных* ДПФ;

- на $(\nu - 1)$-ом этапе определяются *N/2*-точечные ДПФ как комбинация *N/4*-точечных ДПФ;
- на -ом (последнем) этапе определяется искомое *N*-точечное ДПФ как комбинация *N/2*-точечных ДПФ, отсчеты ДПФ следуют в естественном порядке *k*=0, 1,…, (*N*—1).

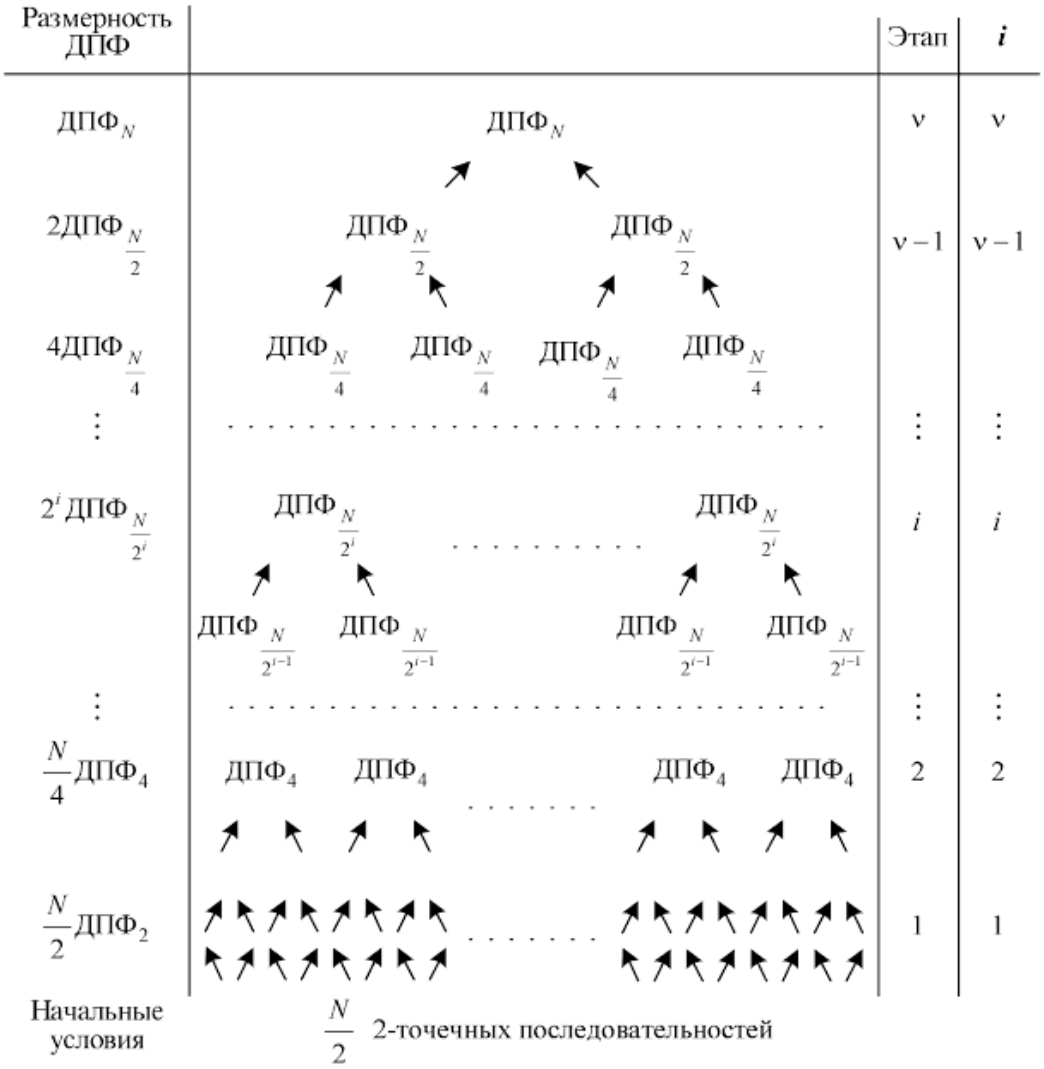


Рисунок 4 – Идея алгоритма с прореживанием по времени

Для реализации данного алгоритма необходимо знать:

– Правила начальной расстановки отсчетов *N*-точечной последовательности;

– Общую формулу расчета ДПФ для произвольного *i*-го этапа.

Получим их, используя следующий прием:

- предположим, что алгоритм БПФ состоит из одного, ν-го этапа.Рассмотрим начальную расстановку отсчетов последовательности и формулу *N*-точечного ДПФ как комбинацию *N/2*-точечных;
- предположим, что алгоритм БПФ состоит из двух, ν-*го и* $(\nu-1)$-*го* этапов. Рассмотрим начальную расстановку отсчетов последовательности и формулы *N*/2-точечных ДПФ как комбинацию N/4-точечных;

- найдем закономерность в начальной расстановке отсчетов последовательности и определении ДПФ как комбинации ДПФ вдвое меньшей размерности. Распространим ее на -этапный алгоритм.

Общая формула расчета на произвольном *i*-м этапе, имеет вид:

$$\begin{cases} X_m^i(k) = X_{2m}^{i-1}(k) + W_L^k X_{2m+1}^{i-1}(k); \\ X_m^i\left(k+\frac{L}{2}\right) = X_{2m}^{i-1}(k) - W_L^k X_{2m+1}^{i-1}(k) \\ i = 1,2,\dots,\nu; \\ m = 0,1,\dots,M-1; \\ k = 0,1,\dots,\frac{L}{2}-1, \end{cases},$$

где *i* – номер этапа;

m– номер ДПФ;

k – номеротсчета ДПФ;

M– количество*L*-точечных ДПФ:

$$M = \frac{N}{2^i} = \frac{2^\nu}{2^i} = 2^{\nu-i}$$

L – размерность ДПФ:

$$L = 2^i;$$

$X_m^i(k)$ – *L*-точечное ДПФ;

$X_{2m}^{i-1}(k)$, $X_{2m+1}^{i-1}(k)$– четное и нечетное *L/2*-точечное ДПФ соответственно.

Основной операцией алгоритма БПФ, называемой базовой, является одно временное (параллельное) вычисление двух отсчетов ДПФ – *k-го* и (*k*+*L/2)-го* по двум формулам. На рисунке 5 приведен направленный граф базовой операции,

получивший название "бабочка" по ассоциации с изображением графа. Кружок обозначает арифметическую операцию сложения/вычитания, верхний выход соответствует сумме, а нижний – разности; стрелка обозначает операцию умножения на поворачивающий множитель, стоящий над ней.

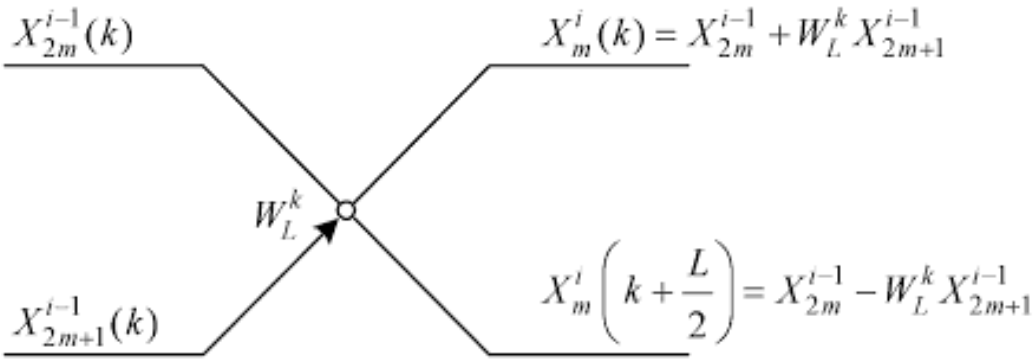


Рисунок 5 – Направленный граф базовой операции алгоритма БПФ с прореживанием по времени («Бабочка»)

Важно отметить, что количество «бабочек», т.е. базовых операций алгоритма, на любом *i*-м этапе одинаково: согласно вышеизложенному, на *i*-мэтапе при фиксированном значении *m* и *k=0,1,…,L/2* требуется *L/2*«бабочек», следовательно, при *m=0,1,…,(M–1)* будем иметь *ML/2*«бабочек». Подставляя значения *M* и *L*, получим одинаковое количество "бабочек" на любом *i*-м этапе:

$$M\frac{L}{2} = \frac{2^i}{2}2^{\nu-i} = \frac{N}{2}.$$

Произведем моделирование данного алгоритма на примере 8-точечного ДПФ ($N = 8, \nu = 3$).

Исследуемый алгоритм БПФ представляет собой трехэтапную процедуру определения ДПФ по общей формуле при *i=1, 2, 3*.

Начальные условия алгоритма формируются в результате трехкратного разбиения исходной 8-точечной последовательности на четные и нечетные отсчеты, а именно:

– первый раз 8-точечная последовательность разбивается на две 4точечные:

– четных [x(0), x(2), x(4), x(6)] отсчетов;

– нечетных [x(1), x(3), x(5), x(7)] отсчетов;

– второй раз каждая из 4-точечных последовательностей вновь разбивается на две 2-точечные: четных и нечетных отсчетов по порядку их следования, считая от нуля:

– 4-точечная последовательность [x(0), x(2), x(4), x(6)] разбивается на две 2-точечные:

– четных [x(0), x(4)] отсчетов;

– нечетных [x(2), x(6)] отсчетов;

– 4-точечная последовательность [x(1), x(3), x(5), x(7)] разбивается на две 2-точечные:

– четных [x(1), x(5)] отсчетов;

– нечетных [x(3), x(7)] отсчетов;

– третий раз каждая из 2-точечных последовательностей вновь разбивается на два отсчета — четный и нечетный:

– 2-точечная последовательность [x(0), x(4)] – на два отсчета:

четный x(0) и нечетный x(4);

– 2-точечная последовательность [x(2), x(6)] – на два отсчета:

четный x(2) и нечетный x(6);

– 2-точечная последовательность [x(1), x(5)] – на два отсчета:

четный x(1) И нечетный x(5);

– 2-точечная последовательность [x(3), x(7)] – на два отсчета:

четный x(3) И нечетный x(7).

Таким образом, получаем начальную расстановку отсчетов – прореженную последовательность: x(0), x(4); x(2), x(6); x(1), x(5); x(3), x(7).

После этого, каждой паре отсчетов ДПФ – четному $X_{2m}^{0}(0)$и нечетному $X_{2m+1}^{0}(0)$ при $m = 0,1,2,3$ присваиваются значения четного и нечетного отсчетов прореженной 8-точечной последовательности (рисунок6):

$$\begin{cases} X_0^0(0) = x(0) \\ X_1^0(0) = x(4) \end{cases} \quad \begin{cases} X_2^0(0) = x(2) \\ X_3^0(0) = x(6) \end{cases}$$
$$\begin{cases} X_4^0(0) = x(1) \\ X_5^0(0) = x(5) \end{cases} \quad \begin{cases} X_6^0(0) = x(3) \\ X_7^0(0) = x(7) \end{cases}.$$

Первый этап: i=1. Определяются четыре 2-точечных ДПФ при$m = 0,1,2,3$ с учетом начальных условий.

$$\begin{cases} X_0^1(0) = x(0) + W_2^0 x(4) \\ X_0^1(1) = x(0) - W_2^0 x(4) \end{cases} \quad \begin{cases} X_1^1(0) = x(2) + W_2^0 x(6) \\ X_1^1(1) = x(2) - W_2^0 x(6) \end{cases}$$
$$\begin{cases} X_2^1(0) = x(1) + W_2^0 x(5) \\ X_2^1(1) = x(1) - W_2^0 x(5) \end{cases} \quad \begin{cases} X_3^1(1) = x(3) + W_2^0 x(7) \\ X_3^1(1) = x(3) - W_2^0 x(7) \end{cases}.$$

На рисунке 6 показан направленный граф алгоритма.

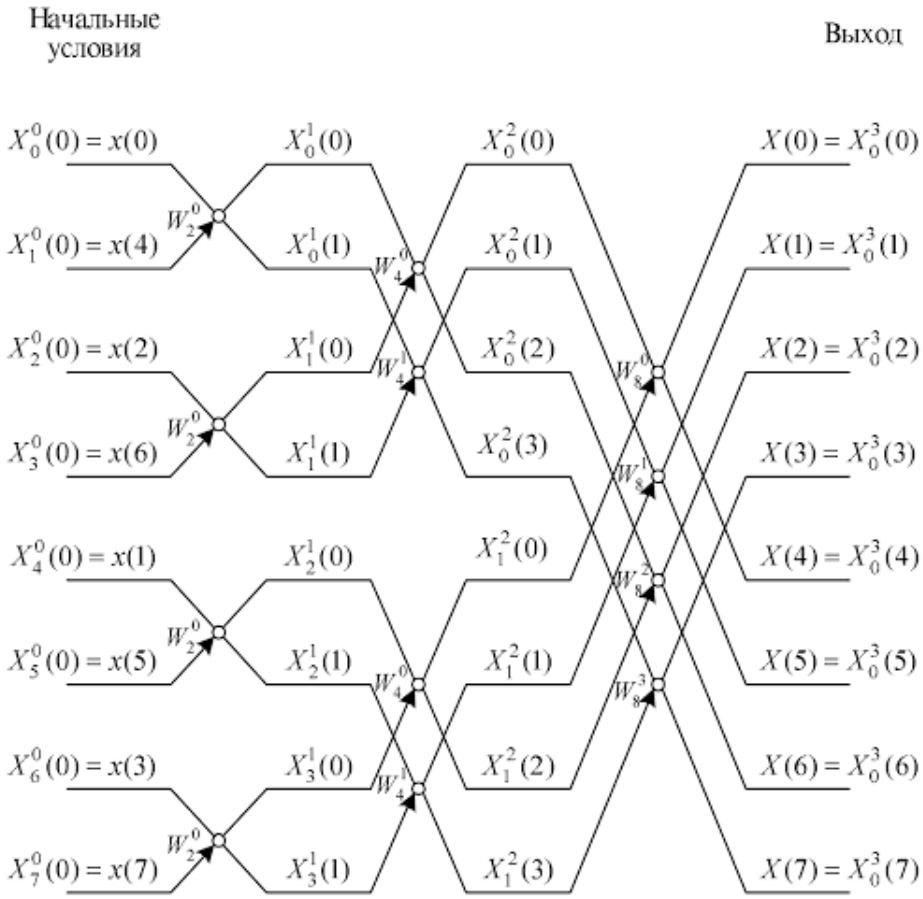


Рисунок 6 – Направленный граф алгоритма БПФ с прореживанием по времени для 8-точечного ДПФ

Поворачивающий множитель оставлен для единообразия с общей формулой, в действительности W_2^0 =1.

Второй этап: i=2. Определяются два4-точечных ДПФ при$m = 0,1$

$$\begin{cases} X_0^2(k) = X_0^1(k) + W_4^k X_1^1(k); \\ X_0^2(k+2) = X_0^1(k) - W_4^k X_1^1(k); \\ k = 0,1; \end{cases}$$

$$\begin{cases} X_1^2(k) = X_0^2(k) + W_8^k X_1^2(k); \\ X_1^2(k+2) = X_0^2(k) - W_8^k X_1^2(k). \\ k = 0,1. \end{cases}$$

Третий этап: i=3. Определяется искомое8-точечное ДПФ при$m = 0$

$$\begin{cases} X_0^3(k) = X_2^1(k) + W_4^k X_3^1(k); \\ X_0^3(k+2) = X_2^1(k) - W_4^k X_3^1(k). \\ k = 0,1,2,3. \end{cases}$$

Полученные отсчеты 8-точечного ДПФ следуют в естественном порядке:

$$\begin{cases} x(k) = X_0^3(k) \\ k = 0,1,\dots,N-1 \end{cases}.$$

Правило расстановки отсчетов исходной последовательности: операция битреверсии.

При больших значенияхN процедура многоэтапного разбиения исходной последовательности на группы четных и нечетных отсчетов весьма трудоемка, однако ее легко формализовать. В таблице 1 показаны две 8точечные последовательности: исходная и прореженная, а также двоичные номера отсчетов данных последовательностей.

Таблица 1 – Бит-реверсивная перестановка

Исходная последовательность		Бит-реверсивный порядок	
Отсчет	Двоичный номер	Двоичный номер	Отсчет
x(0)	000	000	x(0)
x(1)	001	100	x(4)
x(2)	010	010	x(2)
x(3)	011	110	x(6)
x(4)	100	001	x(1)
x(5)	101	101	x(5)
x(6)	110	011	x(3)
x(7)	111	111	x(7)

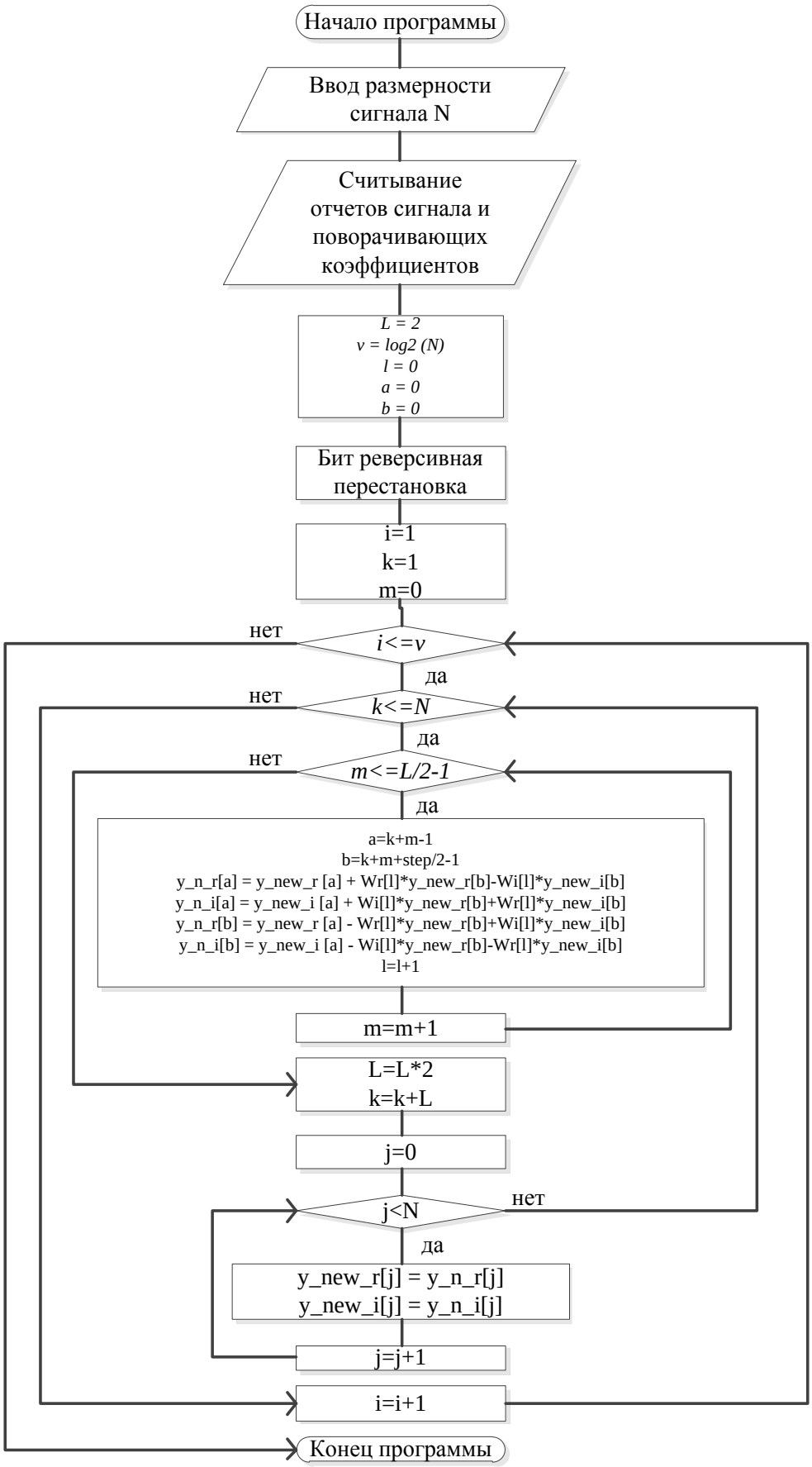


Рисунок 7 – Блок схема программы, реализующей алгоритм с прореживанием по времени

На рисунках 8 и 9 изображены результаты работы программы, по вычислению спектра синусоиды по алгоритму с прореживанием по времени:

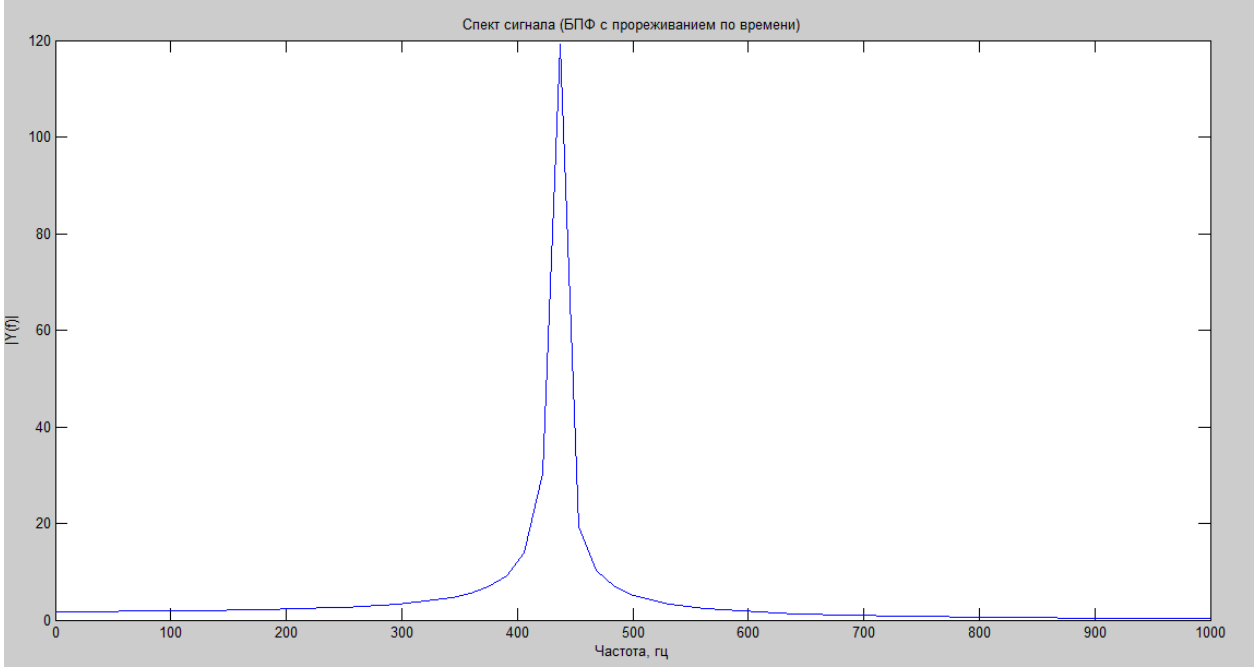


Рисунок 8 – Спектр сигнала (N=128)

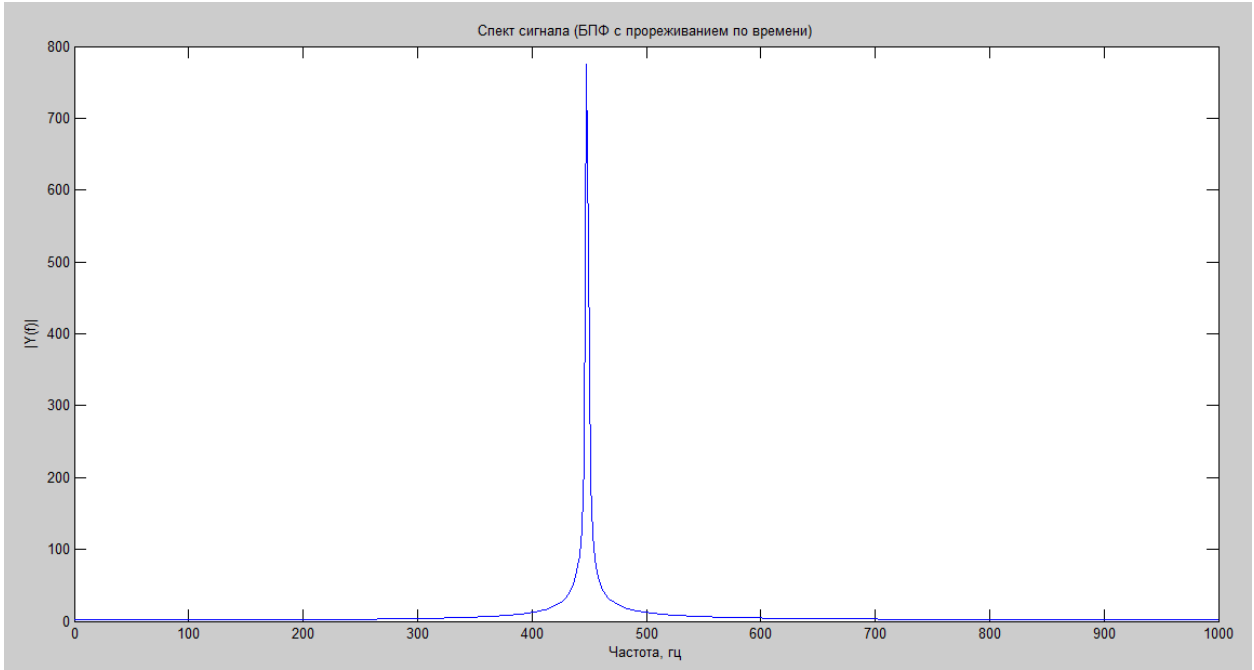


Рисунок 9 – Спектр сигнала (N=1024)

Из полученных результатов можно сделать вывод, что программа составлена верно, за один этап вычисляется 2 отсчета сигнала.

2.3 Алгоритм БПФ с прореживанием по частоте

Основная идея алгоритма БПФ с прореживанием по частоте заключается в поэтапном вычислении N-точечного ДПФ на v этапах, на каждом из которых ДПФ определяется через ДПФ вдвое большей размерности (рисунок 10). Алгоритм БПФ с прореживанием по частоте можно описать следующим образом:

–задание начальных условий:

N-точечная последовательность не прореживается, сохраняется естественный порядок следования отсчетов $n = 0, 1, \dots, N-1$;

– на первом этапе определяются $N/2$-точечные ДПФ $N/2$-точечных последовательностей (двух половин исходной последовательности);

– на втором этапе определяются $N/4$-точечные ДПФ как комбинация $N/2$--точечных ДПФ;

– на i-ом этапе определяются 2^{i-1}–точечные ДПФ как комбинация 2^{i}–точечных ДПФ;

– на i-ом (последнем) этапе определяются 2-точечные ДПФ как комбинация 4-точечных ДПФ. Последовательность из $N/2$2-точечных ДПФ представляет собой искомое N-точечное ДПФ, отсчеты ДПФ следуют в бит реверсивном порядке двоичных номеров.

Для реализации данного алгоритма необходимо знать общую формулу расчета ДПФ на произвольном i-ом этапе. Она была получена путем замены входа и выхода, а также обращения стрелки направленного графа –«бабочки»– алгоритма БПФ с прореживанием по времени (рисунок11). Направленные графы — исходный и полученный в результате указанного преобразования называют дуальными. Таким образом, алгоритмы БПФ с прореживанием по времени и частоте являются дуальными: каждый из них получается из другого путем замены входа и выхода и обращения всех стрелок направленного графа.

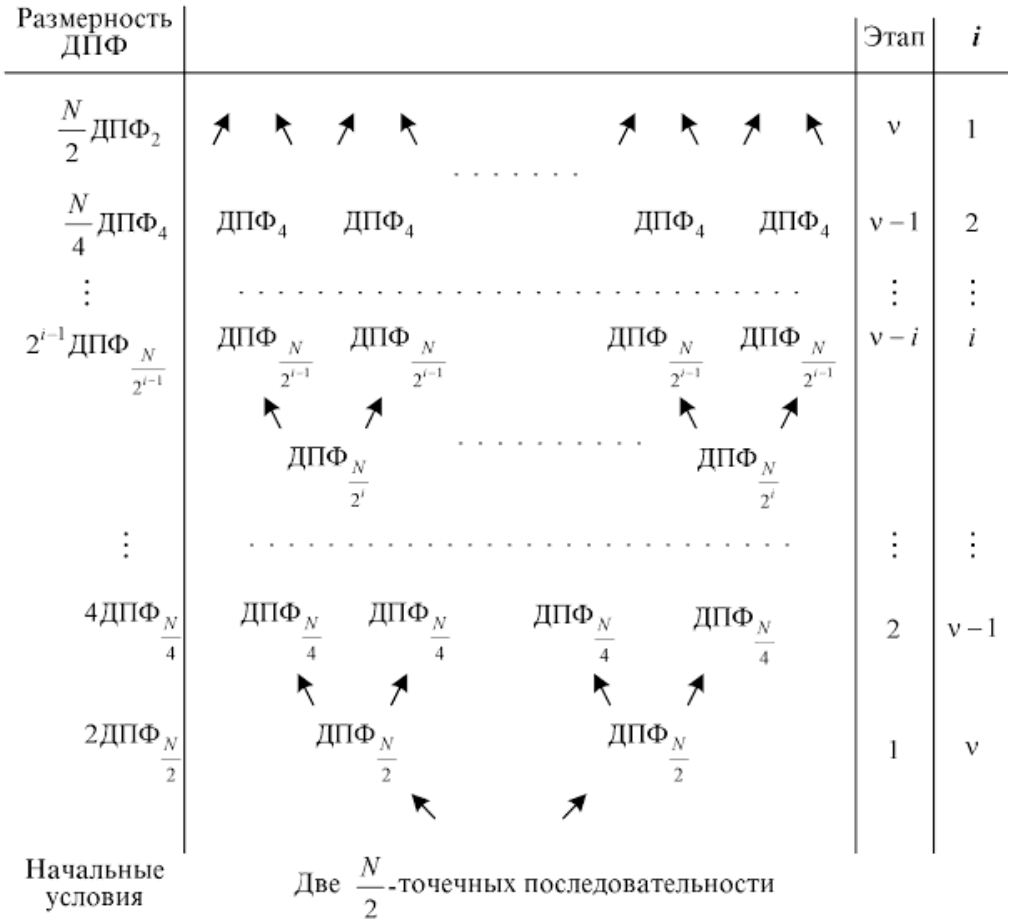


Рисунок 10 – Идея алгоритма БПФ с прореживанием по частоте

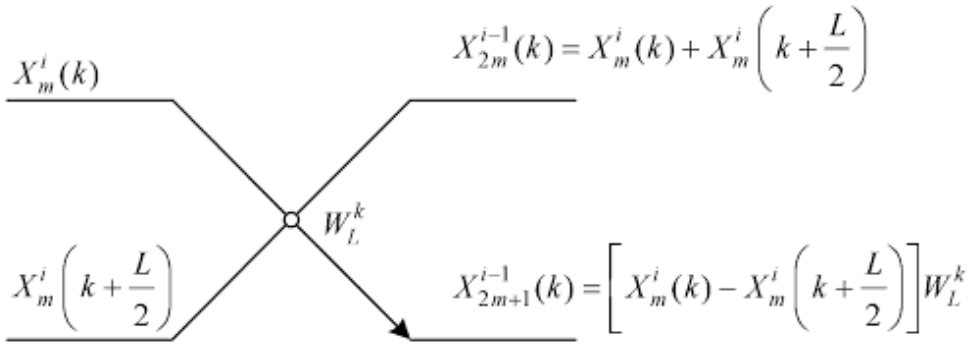


Рисунок 11 – Направленный граф базовой операции алгоритма БПФ с прореживанием по частоте («бабочка»)

Определив базовую операцию «бабочка», несложно записать общую формулу расчета ДПФ для произвольного *i*-го этапа:

$$\begin{cases} X_{2m}^{i-1}(k) = X_m^i(k) + X_m^i\left(k + \frac{L}{2}\right); \\ X_{2m+1}^{i-1}(k) = \left[X_m^i(k) - X_m^i\left(k + \frac{L}{2}\right)\right] \cdot W_L^k \\ i = v, v-1, \dots, 1; \\ m = 0,1, \dots, M-1; \\ k = 0,1, \dots, \frac{L}{2} - 1, \end{cases},$$

где i – номер этапа;

m – номер ДПФ;

k – номеротсчета ДПФ;

M – количествоL-точечных ДПФ:

$$M = \frac{N}{2^i} = \frac{2^v}{2^i} = 2^{v-i};$$

L – размерность ДПФ:

$$L = 2^i;$$

$X_m^i(k)$ – L-точечное ДПФ;

$X_{2m}^{i-1}(k)$, $X_{2m+1}^{i-1}(k)$– четное и нечетное *L/2*-точечное ДПФ соответственно.

Проведем моделирование данного алгоритма на примере 8-точечного преобразования:

Алгоритм БПФ представляет собой трехэтапную процедуру вычисления ДПФ по общей формуле при *i=3, 2, 1*:

Начальные условия алгоритма задаются как:

$$\begin{cases} X_0^3(k) = x(k) \\ k = 0,1, \dots, 7 \end{cases}.$$

Первый этап: $i=v=3$.

Определяются два 4-точечных ДПФ при$m=0$ с учетом начальных условий:

$$\begin{cases} X_0^2(k) = x(k) + x(k+4); \\ X_1^2(k) = [x(k) - x(k+4)]W_8^k; \\ k = 0,1,2,3. \end{cases}$$

Второй этап: *i=v-1*=2.

Определяются четыре 2-точечных ДПФ при *m=0,1:*

$$\begin{cases} X_{2m}^{1}(k) = X_{m}^{2}(k) + X_{m}^{2}(k+2); \\ X_{2m+1}^{1}(k) = [X_{m}^{2}(k) - X_{m}^{2}(k+2)]W_{4}^{k}; \\ k = 0,1; \end{cases}$$

Третий этап: *i=v-2*=1.

Определяются 8-точечное ДПФ при *m=0,1,2,3:*

$$\begin{cases} X_{2m}^{0}(0) = X_{m}^{1}(0) + X_{m}^{1}(1); \\ X_{2m+1}^{0}(0) = [X_{m}^{1}(0) - X_{m}^{1}(1)]W_{2}^{0}; \end{cases}$$

Значения $X_{2m}^{0}(0)$ и $X_{2m+1}^{0}(0)$ при *m*=0, 1, 2, 3 и есть искомое 8-точечное ДПФ, отсчеты которого, согласно (12.34), следуют в бит-реверсивном порядке двоичных номеров:

$$\begin{cases} x(0) = X_{0}^{0}(0) \\ x(4) = X_{1}^{0}(0) \end{cases} \begin{cases} x(2) = X_{2}^{0}(0) \\ x(6) = X_{3}^{0}(0) \end{cases}$$
$$\begin{cases} x(1) = X_{4}^{0}(0) \\ x(5) = X_{5}^{0}(0) \end{cases} \begin{cases} x(3) = X_{6}^{0}(0) \\ x(7) = X_{7}^{0}(0) \end{cases}.$$

Сравнивая направленные графы алгоритмов БПФ с прореживанием по времени и частоте на рисунке 12, нетрудно убедиться вихдуальности:

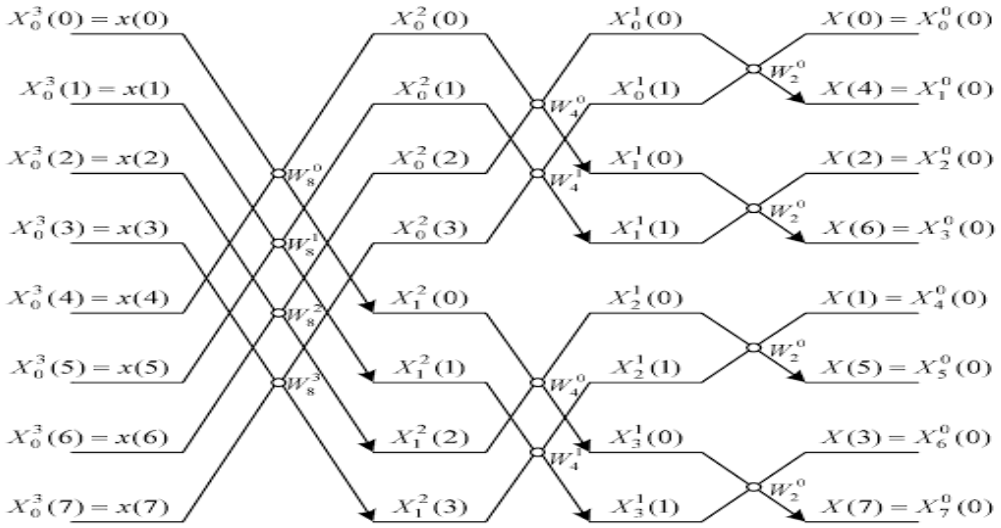


Рисунок 12 – Направленный граф алгоритма БПФ с прореживанием по частоте

На рисунке 13 изображена блок-схема алгоритма

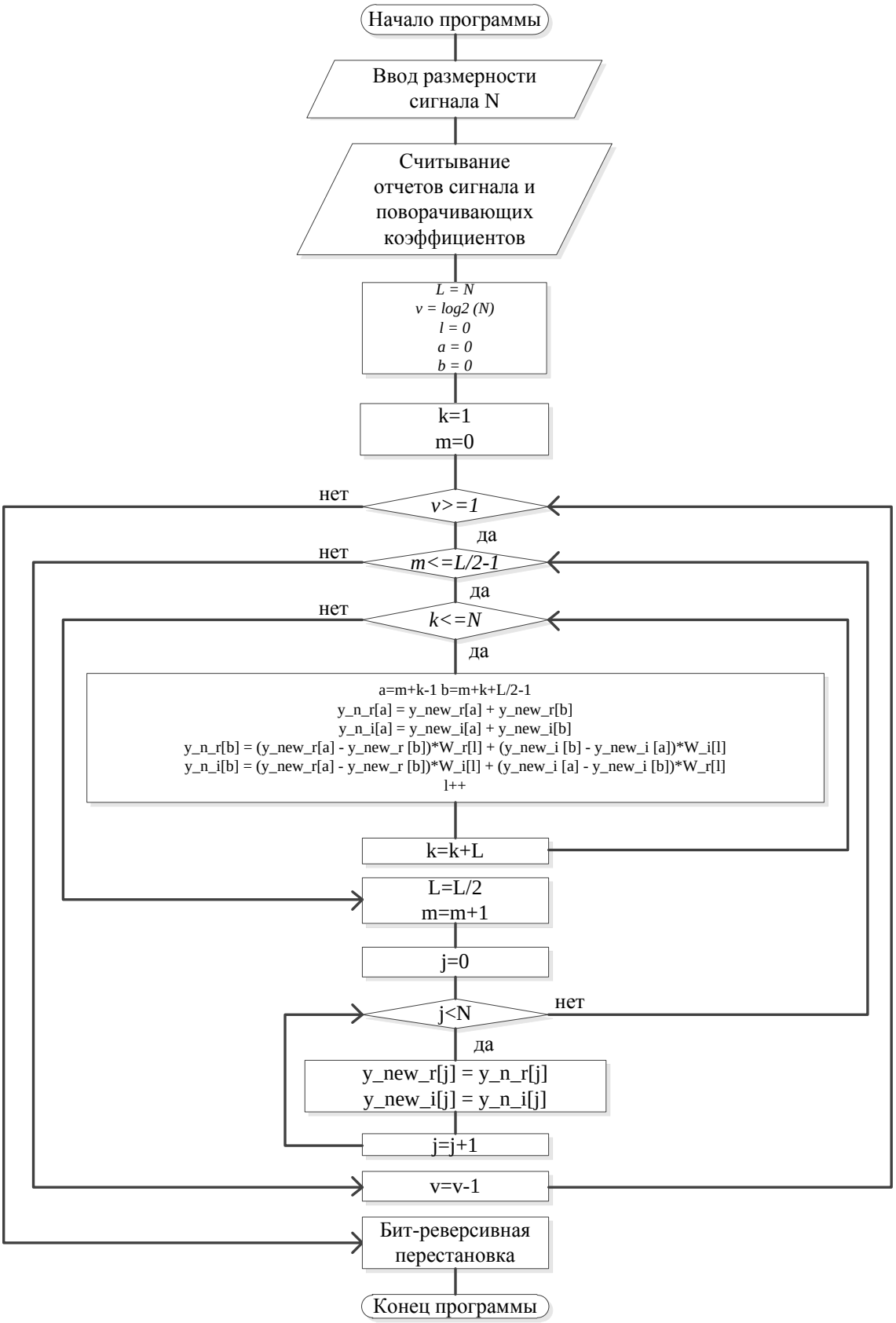


Рисунок 13 – Блок схема программы, реализующий алгоритм БПФ с прореживанием по частоте

На графиках 14 и 15 изображены результаты работы программы, по вычислению спектра тестового сигнала синусоиды с частотой 400 Гц по алгоритму с прореживанием по частоте.

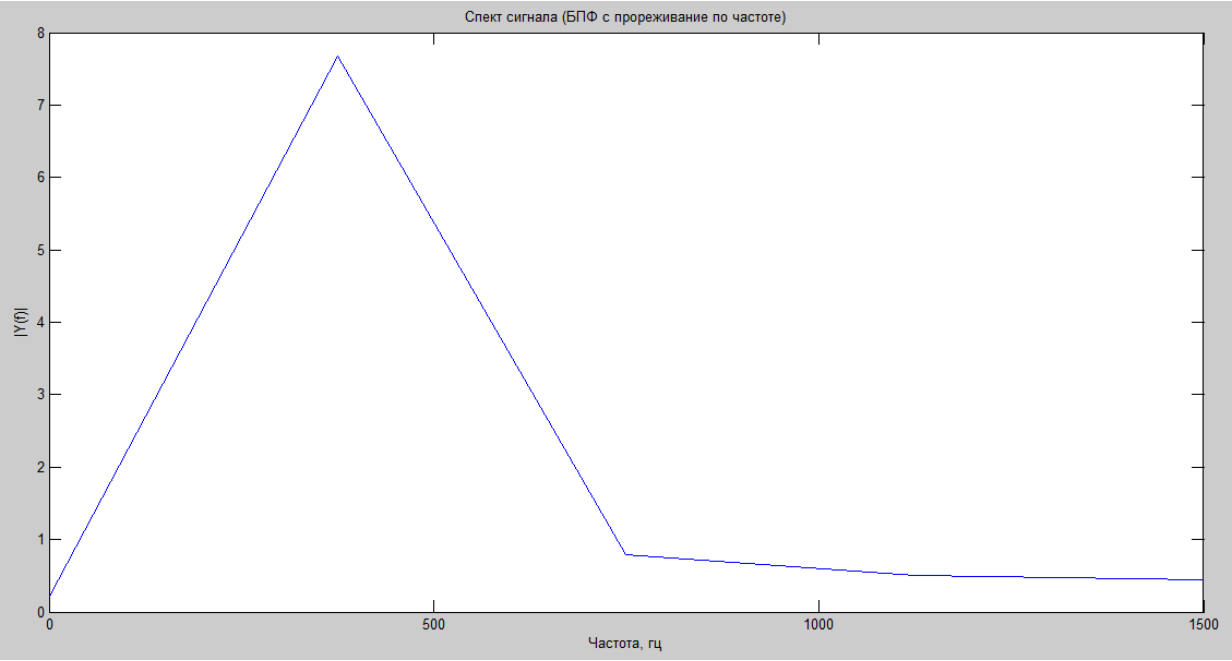


Рисунок 14 – Спектр сигнала (N=8)

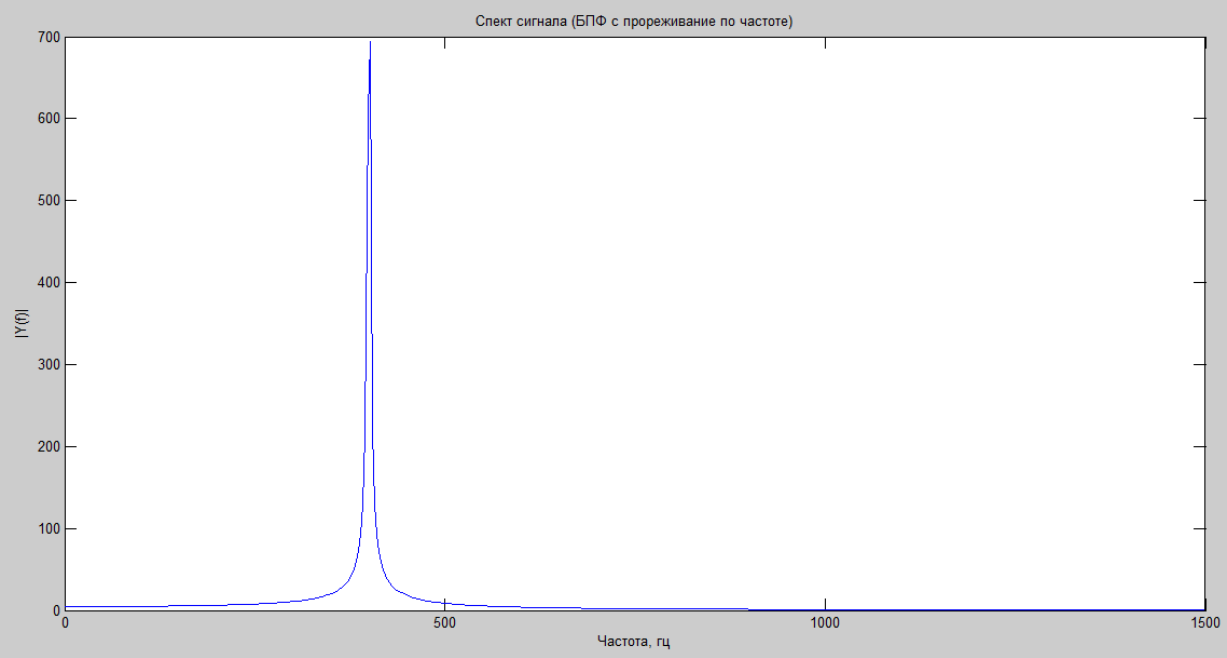


Рисунок 15 – Спектр сигнала (N=1024)

Оценка выигрыша в количестве операций при вычислении ДПФ с помощью алгоритма БПФ с основанием 2

Оценим выигрыш в количестве арифметических операций при вычислении N-точечного ДПФ с помощью рассмотренных алгоритмов БПФ с основанием 2. Любой из этих алгоритмов состоит из v этапов, где $v = \log_2 N$.

Количество "бабочек" на любом этапе алгоритмов БПФ с прореживанием по времени и по частоте одинаково и равно *N/2*. Для одной «бабочки», при фиксированных m и k требуются два сложения (для верхней и нижней формул) и одно умножение на поворачивающий множитель – всего три арифметические операции. Следовательно, для *N/2* «бабочек» на каждом этапе необходимо, а в целом, на всех v этапах, – $\log_2 N$ арифметических операций с комплексными числами. Порядок вычислительной сложности алгоритма БПФ оценивается как $O(N\log_2 N)$, в то время как при прямом вычислении ДПФ он равен $O(N^2)$. Наглядное представление о получаемом выигрыше в объеме вычислений в зависимости от длины N исходной последовательности можно получить из таблицы 2:

Таблица 2 – Теоретическая оценка выигрыша в количестве операций

N	Оценка вычислительной сложности		Оценка выигрыша $N^2 / N\log_2 N$
	Прямое вычисление ДПФ N^2	Вычисление с помощью БПФ $N\log_2 N$	
8	64	24	2,7
16	256	64	4,0
32	1024	160	6,4
64	4096	384	10,7
128	16384	896	18,3
256	65536	1024	32,0
512	262144	4096	56,9
1024	1048576	10240	102,4

В предыдущем разделе мы рассмотрели два самых распространенных алгоритма БПФ Кули-Тюки с прореживанием по времени и частоте, данные алгоритмы, как было показано выше, имеют большое преимущество перед ДПФ.

Однако существуют некоторые модификации алгоритмов БПФ, которые имеют преимущество перед рассмотренными алгоритмами БПФ. Исследуем их особенности и преимущества, а так же составим из модели и проведем симуляцию их работы в Matlab.

2.4 Алгоритм БПФ по основанию 2 с постоянной структурой, без замещения, с нормальным порядком отсчетов на входе и двоично-инверсным на выходе

Алгоритм 16-точечного БПФ с основанием 2 и постоянной структурой на всех этапах показан на рисунке 16.

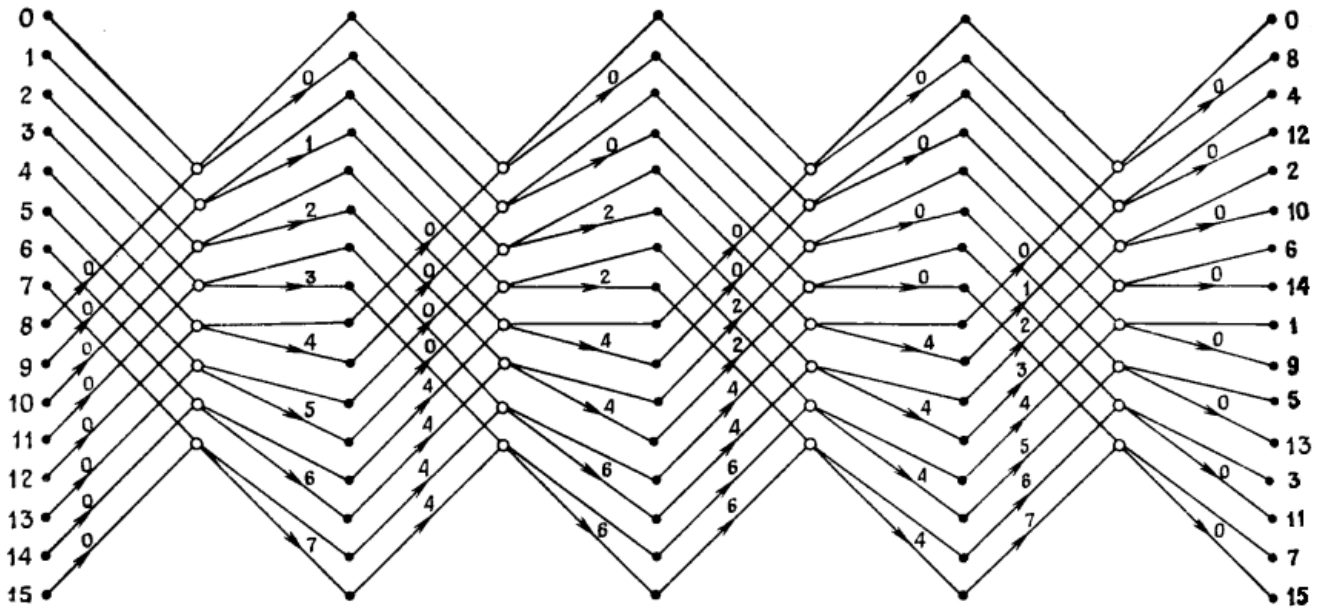


Рисунок 16 – Сигнальный граф алгоритма

Здесь результаты базовой операции не возвращаются в те регистры, откуда были взяты исходные числа, поэтому этот алгоритм относится к алгоритмам без замещения. На всех этапах характер нумерации не меняется,что позволяет в некоторых случаях упростить программы или аппаратуру. Входные отсчеты располагаются в нормальном порядке, а выходные гармоники — в двоично-инверсном. Вообще справедливо следующее правило: при работе с замещением необходимо *N*комплексных регистров, ври работе без замещения— *2N*комплексных регистров. На рисунке 17 изображена блок-схема алгоритма.

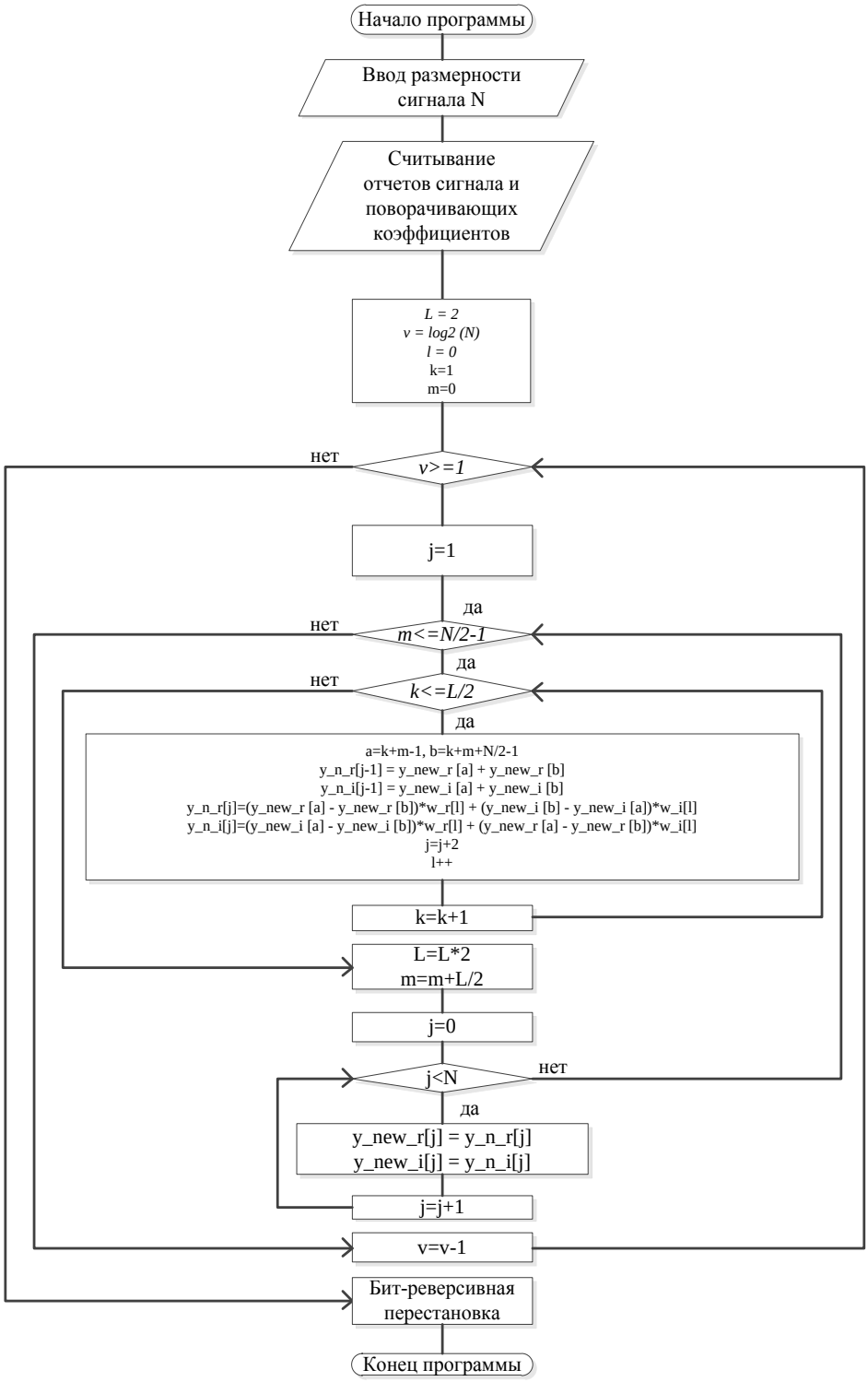


Рисунок 17 – Блок схема программы, реализующей алгоритм

За счет использования сигнального графа с постоянной структурой получается простая в реализации программная часть алгоритма.

Результат работы программы изображен на рисунке 18. В качестве сигнала использовалась смесь суммы синусоид с частотами 150 и 450 Гц и белого Гауссовского шума.

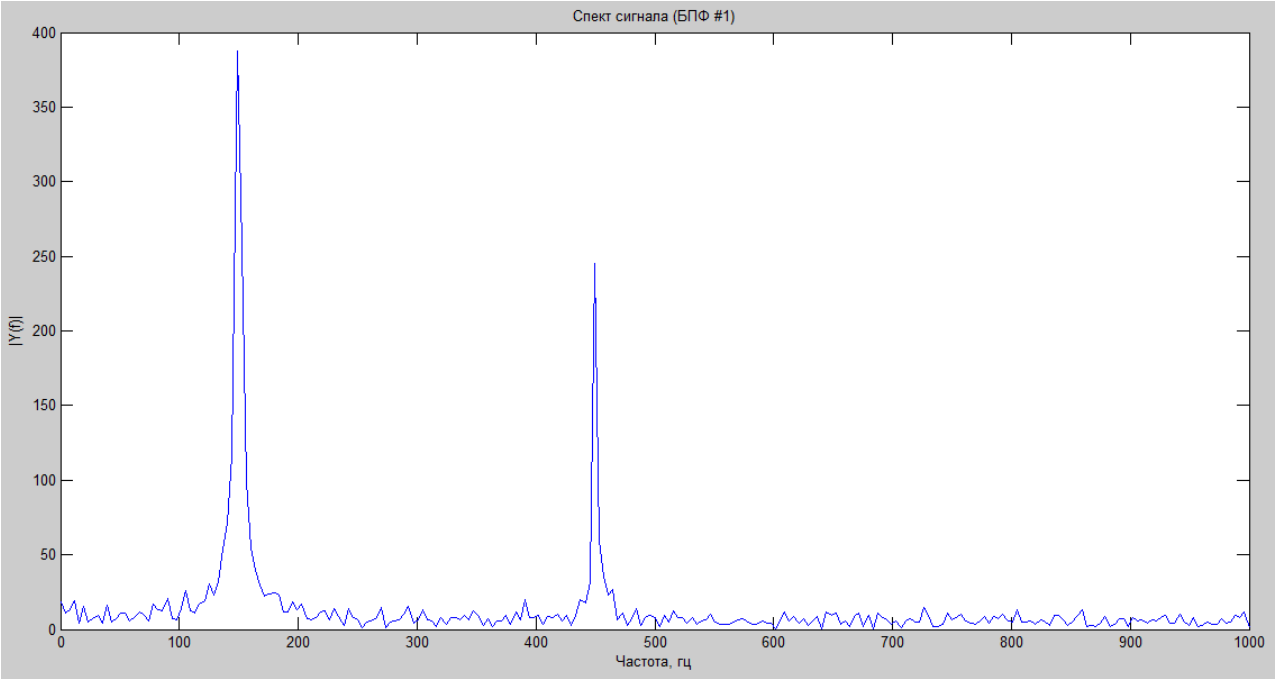


Рисунок 18– Спектр сигнала (N=512)

2.5 Алгоритм БПФ по основанию 2 с постоянной структурой, без замещения, с двоичной инверсией на входе и прямым порядком на выходе

На рисунке 19 представлен сигнальный граф, с помощью которого реализуется алгоритм с постоянной структурой, в котором входные отсчеты располагаются в двоично-инверсном порядке, а выходные гармоники — в прямом.

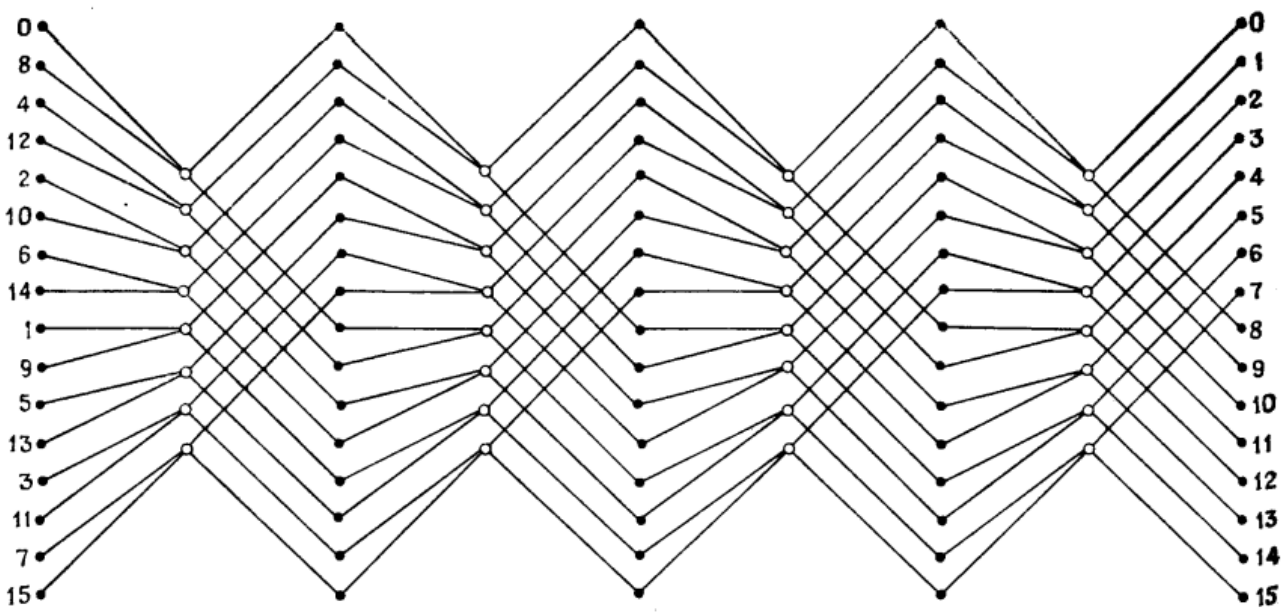


Рисунок 19 – Сигнальный граф алгоритма

Поворачивающие множители для вариантов прореживания повремени и по частоте можно найти из схемы алгоритма на рисунке 16, рассматривая ее от выхода к входу и поменяв направления всех стрелок.

На рисунке 20 изображена блок схема программы, реализующая рассмотренный алгоритм. За счет использования сигнального графа с постоянной структурой получается простая в реализации программная часть алгоритма.

Результат работы программы изображен на рисунке 21. В качестве сигнала использовалась смесь суммы синусоид с частотами 150 и 450 Гц и белого Гауссовского шума (минус 10дБ).

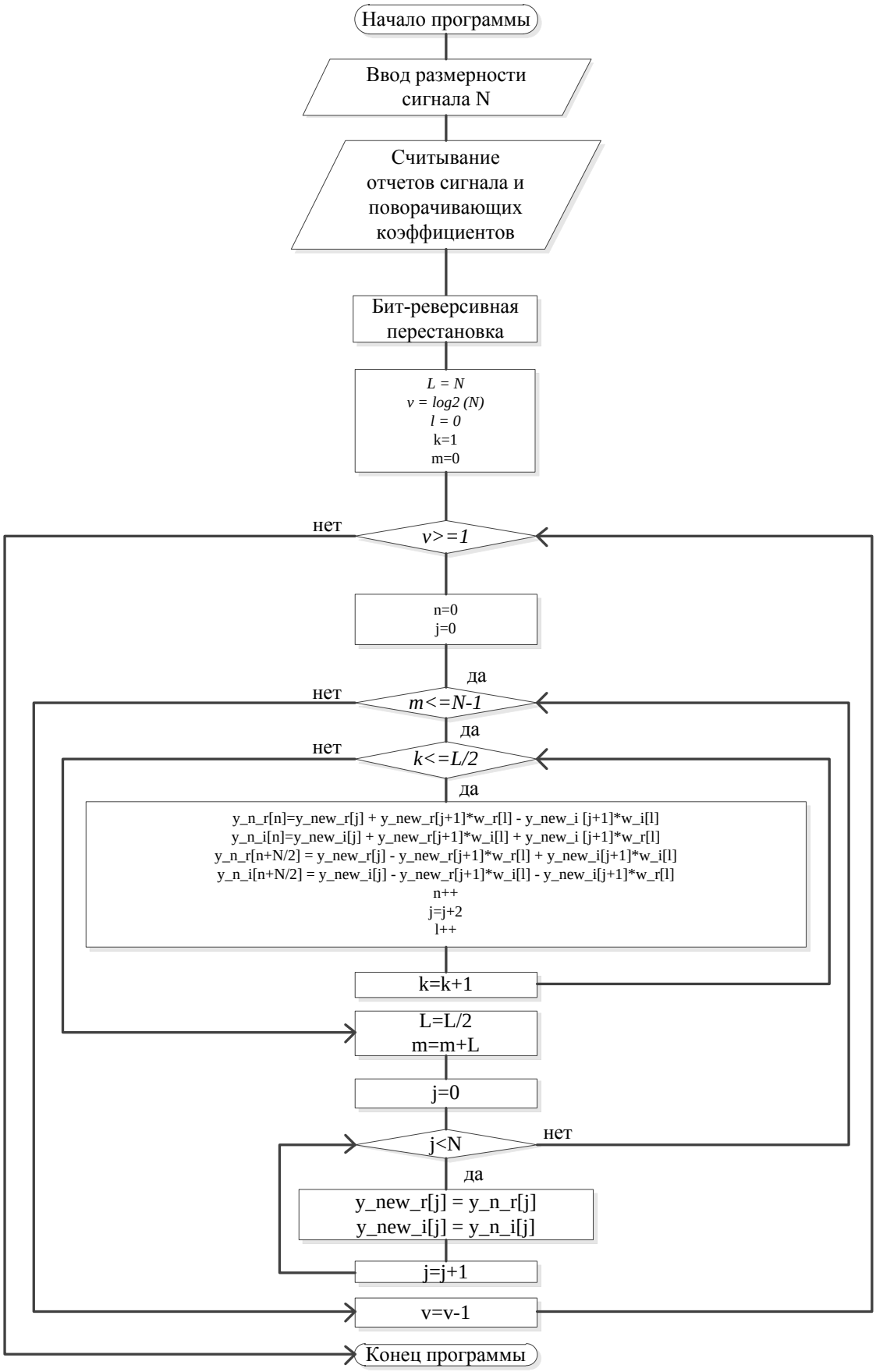


Рисунок 20 – Блок схема программы, реализующей алгоритм

Результат работы программы изображен на рисунке 21. В качестве сигнала использовалась смесь суммы синусоид с частотами 150 и 450 Гц и белого Гауссовского шума (-10дБ).

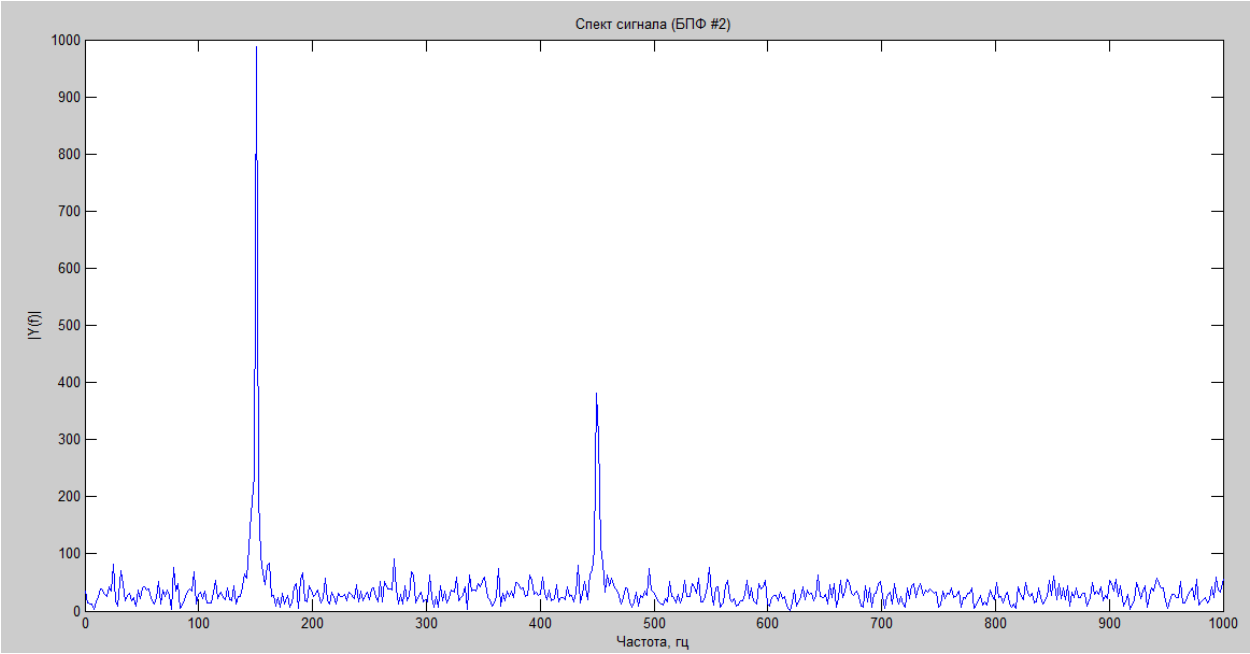


Рисунок 21 – Спектр сигнала (N=1024)

2.6 Алгоритм БПФ по основанию 2 с прореживанием по времени, без замещения, с нормальным порядком на входе и выходе

На рисунке 22 представлен алгоритм, в котором за счет работы без замещения удается избежать двоичной инверсии выходных гармоник. Здесь показан только вариант прореживания по времени. Вариант прореживания по частоте нетрудно получить, переместив стрелки за вершины графа.

На рисунке 24 изображена блок схема программы, реализующая рассмотренный алгоритм. Особый алгоритм позволяет избавиться от блока бит-реверсивной перестановки, тем самым повышая производительность всего алгоритма.

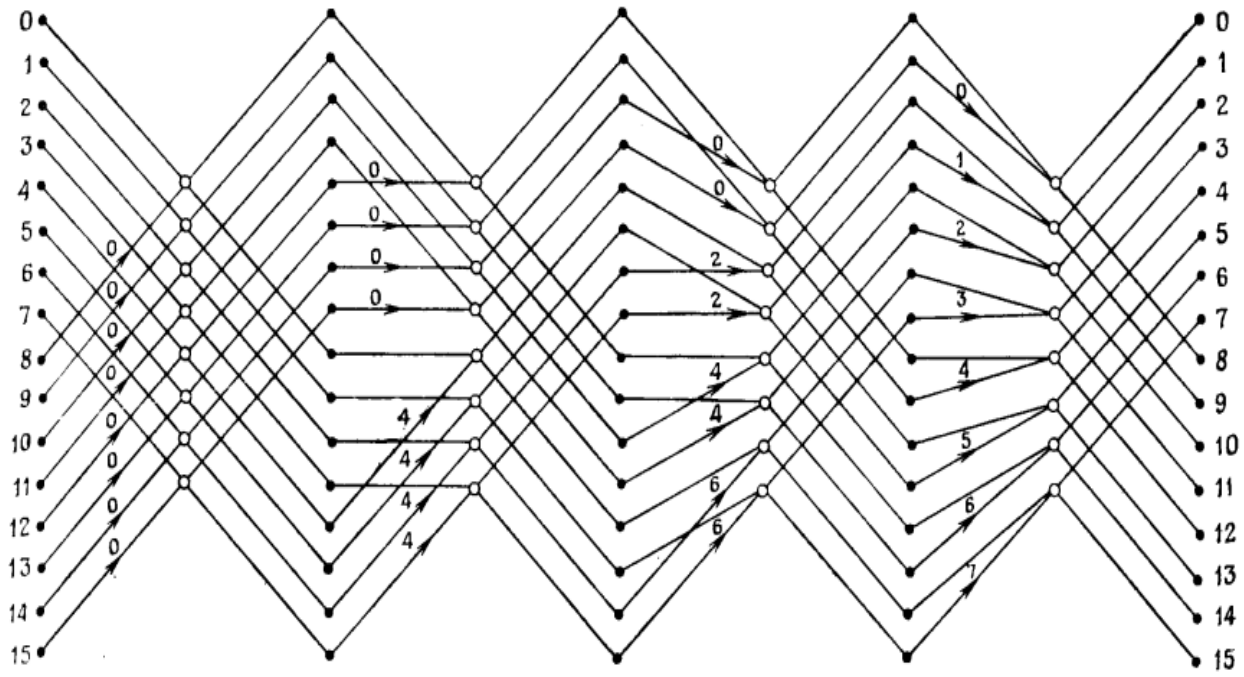


Рисунок 22 – Сигнальный граф алгоритма

Результат работы программы изображен на рисунке 23. В качестве сигнала использовалась смесь суммы синусоид с частотами 150 и 450 Гц и белого Гауссовского шума.

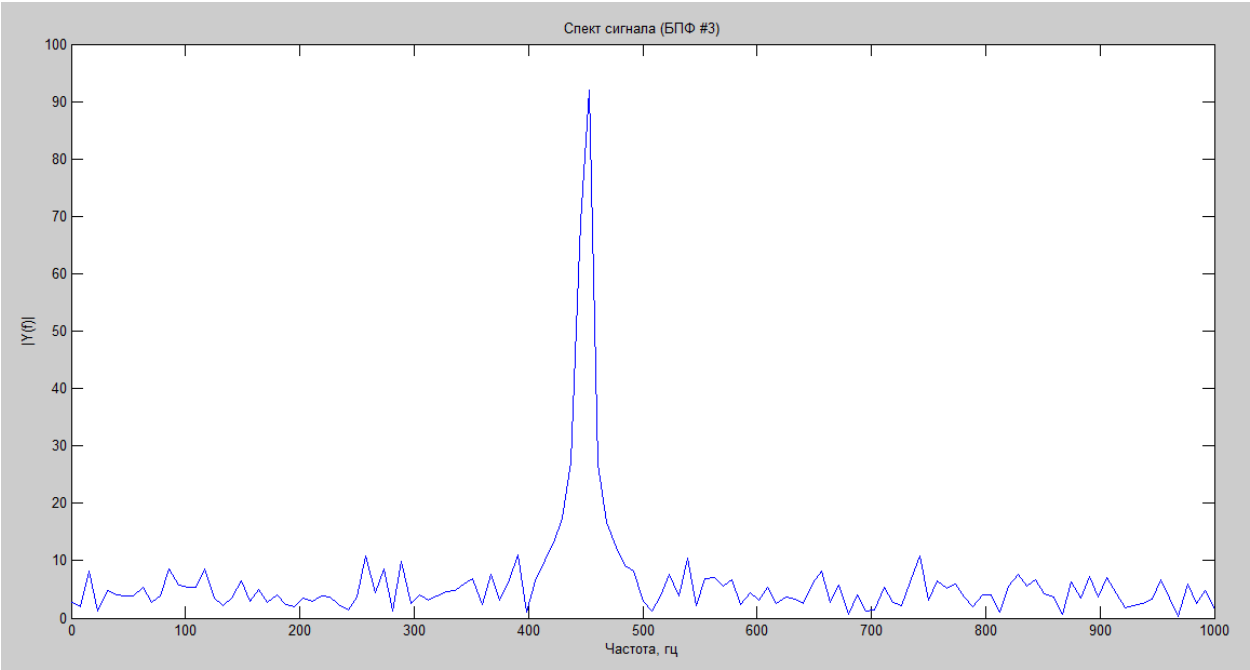


Рисунок 23 – Спектр сигнала (N=256)

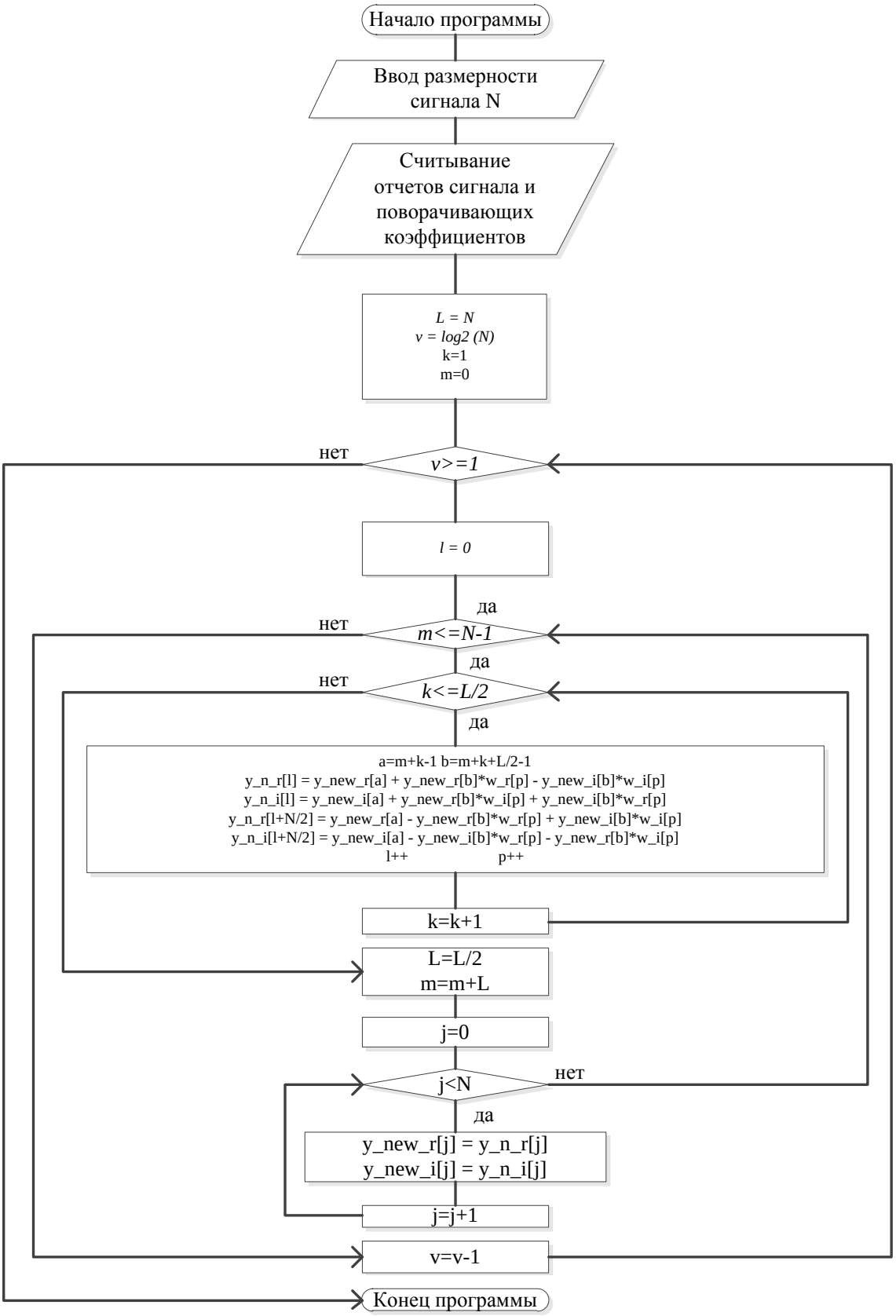


Рисунок 24 – Блок схема программы, реализующей алгоритм

3 Моделирование в среде разработки VisualDSP

3.1 Реализация алгоритмов на сигнальном процессореADSP-BF-537

Для вычисления быстрого преобразования Фурье используется программа, написанная на языке ассемблер.

Описание программы

Эта программа преобразует комплексный входной сигнал во временной областив частотную область с помощью алгоритма быстрого преобразования Фурье по основанию 2.

Переменная fft_sizeсодержит размер входного массива и выходного массивов, и представляет собой количество точек БПФ. Входной и выходной массивы располагаются в разных областях памяти, что позволяет избежать коллизии этих двух массивов и улучшает производительность во время выполнения программы. Если входные данные будут переполнены, то оптимальное использование памяти может быть достигнуто указанием входного массива в качестве выходного массива.

Таблица с поворачивающими коэффициентамитwiddle передается в аргументе twiddle_table, который должна содержать, по крайней мере,fft_size / 2 коэффициентовtwiddle. Таблица составлена из+cosine и –sinekомпонент и может быть инициализирована использованием функции twidfftrad2_fr32. Для оптимальной производительности массивtwiddle располагается в отдельном блоке памяти.

Аргумент twiddle_stride должен быть равенединице, если таблица twiddle первоначально создана для БПФ размера fft_size. Если таблица поворачивающих множителей была создана для большего БПФ размера N × fft_size (где N является степеньюдвух), тогда аргументtwiddle_stride следует установить в положение N.

Этот функция позволяет использовать одну таблицы twiddle для расчетаБПФ разных размеров.

Аргумент scale_method определяет способ применения функции масштабирования при вычислении преобразования Фурье. Используется статическое масштабирование (деление входных данных на каждом этапе на два); динамическое масштабирование(деление входных данныхна два на каждой стадии, при которой наибольший абсолютный входное значение больше или равно 0,5); так же присутствует вариант без масштабирования.

Если выбран статический способ масштабирования, функция всегда будет масштабировать промежуточные результаты, тем самым предотвращая переполнение. Потеря точности увеличивается в соответствии с увеличением параметрafft_size и наиболее выражена для сигналов с малой амплитудой (так как выходной сигнал масштабируется 1/fft_size).

В случае выбора динамического масштабирования, функция проверяет промежуточные результаты и применяет масштабирование там, где это необходимо для предотвращения переполнения. В этом случае потеря точности увеличивается в соответствии с размером БПФ и наиболее выражена для входных сигналов с большой амплитудой (так как поворачивающие коэффициенты увеличивают необходимость в масштабировании).

Если выбрать способ без масштабирования, программа никогда не будет масштабировать промежуточные результаты. Потери точности не произойдет только в случае, если не произойдет переполнение. В этом случае функция будет генерировать насыщенные результаты.Вероятность насыщения увеличивается в соответствии с fft_size и наиболее выражена для входных сигналов с большой амплитудой.

Условия ошибки

Функция cfft_fr32 прекратит работу, если размер БПФ меньше восьми или если значение twiddle меньше единицы.

Алгоритм

С помощью функции исследуется три алгоритма, которые были описаны и рассмотрены в предыдущих разделах.Для выбора алгоритма Кули-Тьюки, необходимо установить аргумент alg_method равным двум. Чтобы выбрать алгоритм с постоянной структурой, необходимо установить параметр alg_methodв значениетри. Чтобы выбрать алгоритм без необходимости бит реверсивной перестановки, необходимо задать аргумент scale_methodединице.

Реализация

Функция состоит из четырех этапов. На первом этапе выполняется бит реверсивная перестановка (если необходимо). В это же время, считываются входные данные,и определяется, есть ли необходимость в масштабировании на следующем этапе.

Доступ к входному массиву на первом этапе вычисляется с использованием «Базовый адрес + сдвиг с обратным битом». Это устраняет вероятность для входного буфера, находиться на границе памяти, которая является кратной размеру БПФ.

Бит реверсивная перестановка вычисляется для БПФ размером N / 4, при этом последующий доступ к отсчетам осуществляется через фиксированные смещения:

```
 Normal      Normal                  Bit-reversed
 order       order                      order
(n = 16)     (n/4)                   (n = 16)
   0           0   -> bit reversed =    0
   1                                     8   ( 0 + N/2 )
   2                                     4   ( 0 + N/2 - N/4 )
   3                                    12   ( 0 + N/2 - N/4 + N/2 )
   4           1   -> bit reversed =    2
   5                                    10   ( 2 + N/2 )
   6                                     6   ( 2 + N/2 - N/4 )
   7                                    14   ( 2 + N/2 - N/4 + N/2 )
   8           2   -> bit reversed =    1
   9                                     9   ( 1 + N/2 )
  10                                     5   ( 1 + N/2 - N/4 )
  11                                    13   ( 1 + N/2 - N/4 + N/2 )
  12           3   -> bit reversed =    3
  13                                    11   ( 3 + N/2 )
  14                                     7   ( 3 + N/2 - N/4 )
  15                                    15   ( 3 + N/2 - N/4 + N/2 )
```

На втором и третьем этапе вычисляются «бабочки», в которых умножение на поворачивающие множители носят тривиальных характер. Здесь так же каждый этап используется сканирования данных для определения необходимости масштабирования на следующем этапе.

Заключительный этап используется для вычисления оставшихся вычислений. Умножение дробных чисел различается в зависимости от того, требуется ли масштабирование или нет. В случае необходимости масштабирования комплексноеумножение вычисляется с использованием формата умножения 1.30 на 1.30 дробное умножение без округления. Масштабирование неявно обрабатываются с использованием множителя IS(с насыщением) и сдвигом 16 вместо 15.

```
A1  =x.re.hi * twiddle.re.lo (M), A0  = x.re.hi * twiddle.re.hi (IS);
A1 += x.re.lo * twiddle.re.hi (M);
A1 -= x.im.hi * twiddle.im.lo (M);
A1 -= x.im.lo * twiddle.im.hi (M), A0 -= x.im.hi * twiddle.im.hi (IS);
A1 = A1 >>> 16;
R2 = (A0 += A1);
```

Мнимая часть произведения вычисляется по тем же правилам, что и реальная часть выше. Проблемы точности устраняются путем масштабирования результатов умножения (смещение любых ошибок округления).

Без масштабирования продукт должен сохранять большую точность. По этой причине, используется формат 1.31 на 1.31 дробное умножение с округлением.

```
A1 = x.re.hi * twiddle.re.lo (M),
A0 = x.re.lo * twiddle.re.lo (FU);
A1 + = x.re.lo * twiddle.re.hi (M);
A1 + = (-x.im.hi) * twiddle.im.lo (M),
A0 + = (-x.im.lo) * twiddle.im.lo (FU);
A1 - = x.im.lo * twiddle.im.hi (M);
A1 + = 0x4000,
A0 + = 0x8000;
A0 = A0 >> 16;
A0 + = A1;
A0 = A0 >>> 15;
A0 + = x.re.hi * twiddle.re.hi;
R2 = (A0 - = x.im.hi * twiddle.im.hi);
```

Мнимая часть произведения вычисляется по тем же строкам, что и реальная часть выше.

Для улучшения производительности оставшихся этапов данные сканируются только в случае выбора динамического масштабирования (за исключением последнего этапа, когда такая проверка не требуется).

Для тестирования программы служит программа, написанная на языке C++, листинг которой приведен в приложении А, для вычисления быстрого преобразования Фурье используется подпрограмма cfft_fr32, написанная на языке ассемблер, листинг программы приведен в приложение Б.

Рассмотренные в нашем исследовании алгоритмы с прореживанием по времени и частоте имеют одинаковую производительность и обратную структуру. Аналогично устроены алгоритмы БПФ по основанию 2 с постоянной структурой. В связи с этим целесообразно будет провести моделирование только одного алгоритма и каждой пары. Итого в исследовании будут сравниваться три алгоритма БПФ.

Проект в VisualDSP++ состоит из следующих исходных файлов:

Предоставляемые исходные файлы:

– Cfft_fr32.asm – комплексное быстрое преобразование Фурье по основанию 2 с использованием нескольких алгоритмов;

– Twidfftrad2_fr32.c – генератор таблицы поворачивающих множителей для БПФ с основанием 2.

Описание

Функция twidfftrad2_fr32 вычисляет комплексные поворачивающие коэффициенты для БПФ размером fft_size и возвращает коэффициенты в векторе twiddle_table. Размер вектора, который известен как таблица twiddle, должен быть не меньше fft_size / 2. Он содержит пары синуса и косинуса значения, которые используются функцией FFT для вычисления Быстрого преобразования Фурье.

Если таблица twiddle была создана для 1024 точек БПФ, то эта же таблица может также использоваться для вычисления 256-точечного БПФ, установив для аргумента striddlestride значение четыре.

Алгоритм

Twiddle_table [k] .re = cos (2 * PI * k / fft_size)
Twiddle_table [k] .im = -sin (2 * PI * k / fft_size)
Где k = 0, ..., (fft_size / 2 - 1)
Test_fft.c – тестовая программа для проверки работы алгоритма

Генератор таблицытwiddle ссылаются на библиотечные функции fract32 cos_fr32 и sin_fr32

Cos_fr32.c – функция косинуса

Sin_fr32.c – функция синуса

Также предоставляются два файла заголовков

filter.h

math_bf.h

На рисунке 25 показаны результаты тестирования программы для процессора ADSPBF-537

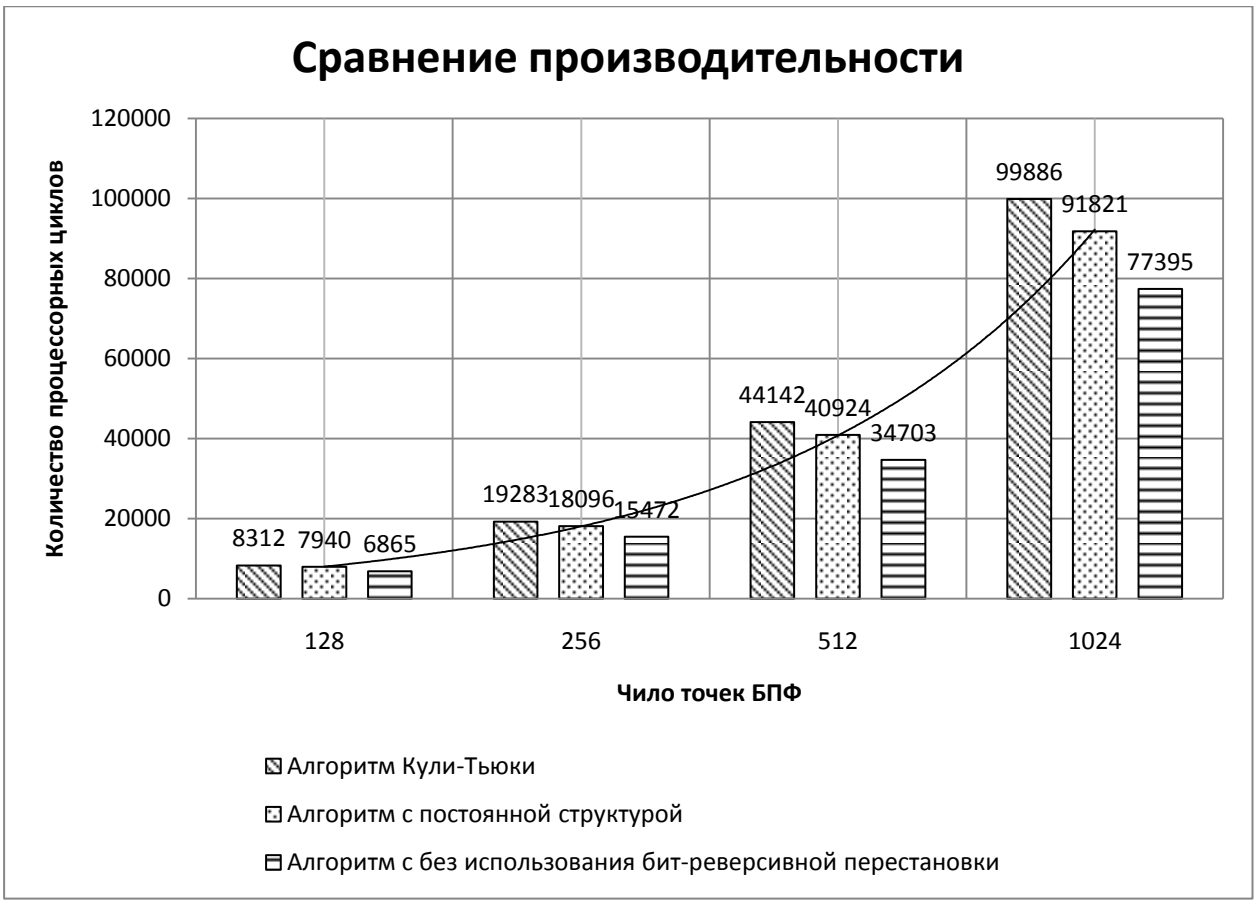


Рисунок 25 – Результаты тестирования программы

Самый быстрый результат показал алгоритм БПФ по основанию 2 с прореживанием по времени, без замещения, с нормальным порядком на входе и выходе. Это объясняется тем, что отсутствует необходимость бит-реверсивной перестановки как входного, так и выходного массивов. Так же к достоинствам данного алгоритма можно отнести относительно простая реализация. Для вычисления 1024-точечного алгоритма потребовалось 77395 процессорных циклов.

При вычислении БПФ с помощью алгоритма с постоянной структурой, выигрыш в производительности, по сравнению с алгоритмом Кули-Тьюки, достигается за счет простоты реализации и одинаковых действий на каждом этапе вычислений. Для вычисления 1024-точечного алгоритма БПФ с постоянной структурой потребовалось 91821 процессорный цикл.Для вычисления 1024-

точечного алгоритма БПФ Кули-Тьюки потребовалось 99886 процессорных циклов. На рисунке 26 изображен спектр сигнала.

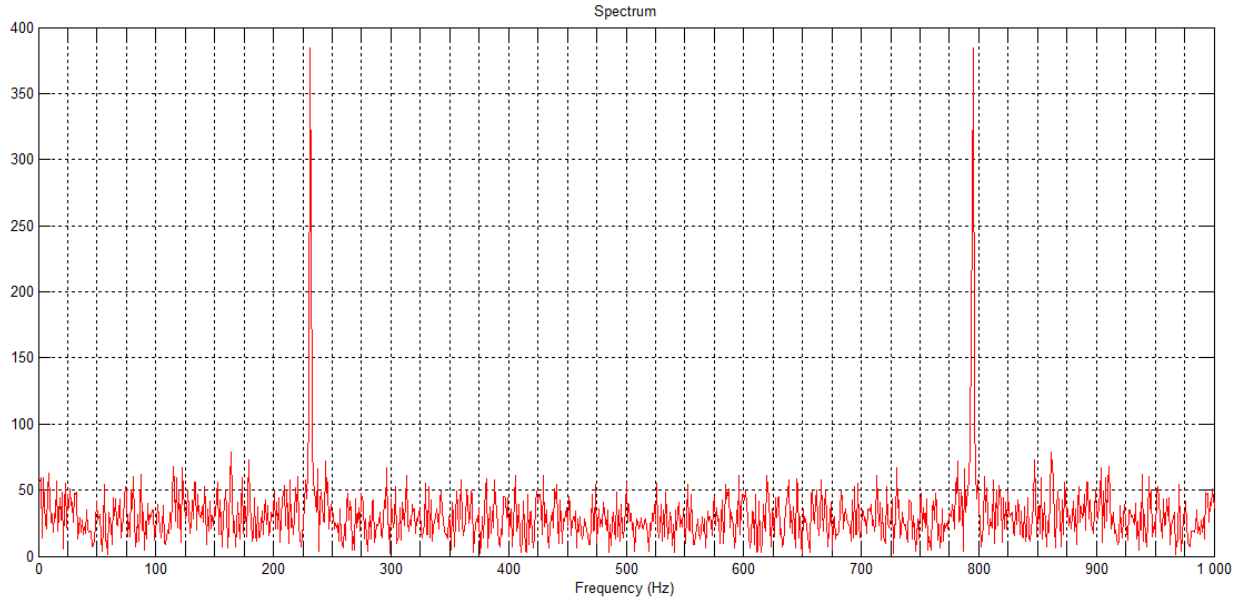


Рисунок 26 – Спектр сигнала

3.2Реализация алгоритмов на сигнальном процессоре ADSP-TS-201

Алгоритмы реализованы в программном продукте компании AnalogDevicesVisualDSP++. Программа составлена на языке ассемблер, и вызывается с помощью с-функции.

Программа служит для вычисления комплексного с-вызываемогоБПФ по основанию 2 на процессоре семейства TigerSHARC.

Описание функции

а. Входные параметры

j4 ->input // Входноймассив
j5 ->ping-pongbuffer 1/2 // Промежуточныебуфер
j7 ->output // Выходноймассив
j27+0x18 -> N = Number of points // ЧислоточекБПФ
j27+0x19 ->REALorCOMPLEX // Действительноеили комплексное БПФ

б. Примервызовафункции

Fft32 (& (input), & (ping_pong_buffer1), & (ping_pong_buffer2), & (output), N, COMPLEX);

в. Ограничения

Все массивы должны быть выравнены по границе памяти, кратной 4.

Значение N должно быть между 32 и MAX_FFT_SIZE.

Буфер ping_pong_buffer1 может быть задействован как входной.

Буфер ping_pong_buffer2 может быть выходным буферов, если число стадий БПФ четное (т.е.log2(N) является четным) или ping_pong_buffer1, если числостадий БПФ нечетное (то есть log2(N) является нечетным).

г. Максимальное число точек БПФ MAX_FFT_SIZE можно выбрать с помощью директивы #define. Большие значения позволяют делать больший выбор N, но в тоже время будут занимать много памяти.

д. Вызываемая функция может обрабатывать до 64K блоков данных на процессоре ADSP-TS-201.

Описание алгоритма БПФ

– Входные данные рассматриваются как комплексная N-точечная последовательность.

– Из-за перестановки входных отсчетов на каждом этапеиспользуются алгоритмы без замещения.

– Бит-реверсивная перестановка(кроме последнего алгоритма) и первые два этапа объединяются в один цикл. Этот цикл принимает данные из входных данных и сохраняет их в промежуточном буфереping-pong1.

– Каждые последующие этапыиспользуют промежуточные буферы для хранения промежуточных результатов. На последнем этапе для вывода используется буфер output.

– Алгоритм БПФ спроектирован для вызова с любым размером точкиN <=MAX_FFT_SIZE

Ниже приведена таблица 2, в которой в первом столбце указано названия файла в составе проекта, во втором столбце его функция и краткое описание.

Таблица 2 – Файлы проекта

Имя файла	Описание и выполняемая функция
InputsFolder	В данной папке хранятся файлы с входными массивами различных размерностей, которые генерируются с помощью программы составленной в Matlab
OutputsFolder	Папка, предназначенная для хранения результатов работы программы
Tcl_TS201 Folder	Tclскрипт используется для построения, выполнения и сохранения результатов вычисления БПФ
TwiddlesFolder	Папка для хранения таблицы с поворачивающими множителями
ADSP-TS201_C.ldf	Linker Description File for TS201
init.c	Подпрограмма для вызываемой функции для вычисления комплексного БПФ
main.c	Основная программа для вызова ассемблерного файла fft_flp32.asm
fft_flp32.asm	Алгоритм быстрого преобразования Фурье
variables.asm	Таблица поворачивающих коэффициентов
fft_flp32_TS201.dpj	Файл для запуска проекта
defts201.h	Определяется для ADSP-TS201
FFTDef.h	Определяет процедуры БПФ
ini_cache.h	Содержит программу, которая предварительно загружает кеш с twiddless
cache_macros.h	Макросы для инициализации кэша

На рисунке 28 приведен график зависимости количества процессорных циклов от числа точек БПФ.

Рисунок 27 – Результаты тестирования программы

На рисунке 28 приведен спектр сигнала.

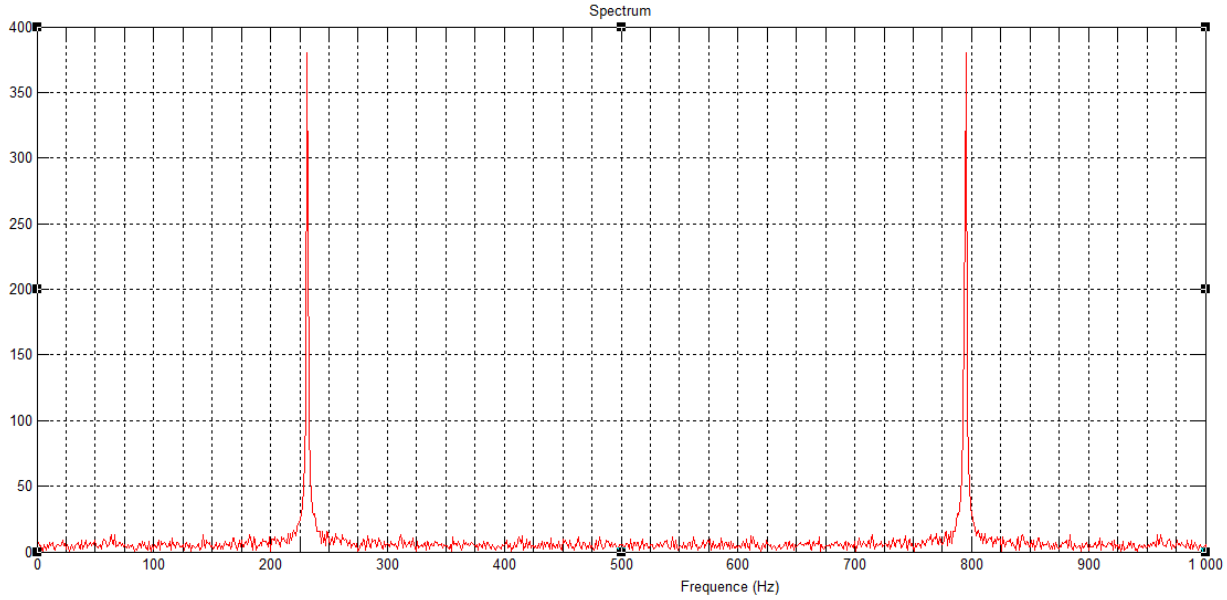


Рисунок 28 – Вычисленный спектр

Быстрое преобразование Фурье вычисленное на цифровом сигнальном процессоре ADSP-TS-201 показало выигрыш в производительности на порядок в сравнении с ADSP-BF-537. Наряду с этим самый быстрый результат так же

показал алгоритм БПФ по основанию 2 с прореживанием по времени, без замещения, с нормальным порядком на входе и выходе. Для вычисления 1024-точечного алгоритма потребовалось 9992процессорных цикла.

При вычислении БПФ с помощью алгоритма с постоянной структурой, потребовалось 11662 процессорных циклов.Для вычисления 1024-точечного алгоритма БПФ Кули-Тьюки потребовалось 12170 процессорных циклов.

В следующем разделе будут рассмотрены эффекты квантования и произведена оценка численных результатов.

3.3 Оценка эффектов квантования в алгоритмах БПФ

При составлении программ необходимо учитывать эффекты, которые связаны с представлением обрабатываемых чисел и коэффициентов с ограниченной точностью. К таким эффектам относятся:

- шумы округления при усечении результатов умножений;
- погрешности масштабирования промежуточных результатов с целью устранения возможных переполнений;
- погрешности преобразования, связанные с конечной точностью представления коэффициентов W_N^kи отсчетов сигнала.

Уэлч и ряд других авторов подробно проанализировали влияние шума округления и масштабирования для алгоритмов БПФ в системах с фиксированной и плавающей точкой. Рассмотрим сначала вопросы реализации алгоритмов БПФ с основанием 2 при выполнении арифметических операций с фиксированной точкой, в нашей работе, реализованной на сигнальном процессоре ADSPBF-537.

Если последовательность *{x (n)}* из *N* отсчетов имеет коэффициенты ДПФ *{X (к)},* то, согласно теореме Парсеваля,

$$\sum_{n=0}^{N-1} x^2(n) = \frac{1}{N}\sum_{n=1}^{N-1} |X(k)|^2,$$

т.е. средняя мощность выходных гармоник в *N* раз превышает среднюю мощность исходной последовательности. Следовательно, значения ДПФ

последовательности будут, вообще говоря, существенно превышать значения самой последовательности. Поэтому при использовании арифметики с фиксированной точкой необходимо во избежание переполнений ввести масштабирование. Чтобы показать, как могут возникать переполнения, рассмотрим базовую операцию на *m-м* этапе БПФ с прореживанием по времени. Пусть$f_m\ (i)$ и $f_m\ (j)$ – входные числа базовой операции, $f_{m+1}\ (i)$ и $f_{m+1}\ (j)$ – выходные числа, причем W^p – поворачивающий множитель. Тогда

$$f_{m+1}(i) = f_m(i) + W^p f_m(j),$$
$$f_{m+1}(j) = f_m(i) - W^p f_m(j).$$

Из формулы видно, что при переходе от этапа к этапу модули чисел, вообще говоря, увеличиваются, поэтому их нужно масштабировать сдвигом вправо. Нетрудно показать, что от этапа к этапу максимальное значение модуля комплексных чисел не убывает и, как видно из соотношения, удовлетворяет неравенству

$$\max\{|f_m(i)|, |f_m(j)|\} \leq \max\{|f_{m+1}(i)|, |f_{m+1}(j)|\} \leq 2 \cdot \max\{|f_m(i)|, |f_m(j)|\}.$$

Следовательно, уровень сигнала на каждом этапе увеличивается не быстрее, чем на один двоичный разряд.

Учитывая вышеизложенное, можно предложить три методамасштабирования:

– Сдвиг вправо на один разряд на каждом этапе. Если $|f_0(i)| \leq 1/2$ для любых *i*и числа сдвигаются на разряд вправо послекаждого этапа итерации (кроме последнего), то переполнения не произойдет.

– Контроль последовательности, гарантирующий выполнение условия $|f_m(i)| \leq 1/2$ для всех *i*. На каждом этапе БПФ вычисляется массив $f_m(i)$, и если хотя бы один отсчет массива превысит по модулю 1/2,весьмассив масштабируетсясдвигом вправо на разряд.

– Проверка на переполнение. При этом исходная последовательность масштабируется так, чтобы $|Re\{f_0(i)\}|$и $|Im\{f_0(i)\}|$ были меньше единицы (а не 1/2, как в двух предыдущих методах). Если в ходе базовой операции фиксируется

переполнение, то все числа последовательности, включая результаты уже выполненных на данном этапе (без переполнения) базовых операций, сдвигаются па разряд вправо, после чего итерация продолжается с той базовой операции, где произошло переполнение. На каждом этапе может произойти более чем одно переполнение, но не больше двух.

Первый метод связан с наименьшими затратами времени и наиболее прост для программирования. Однако он дает наименьшую точность, так как масштабирование на каждом этапе приводит к ненужному ухудшению точности в случаях, когда его можнобылоне производить. Второй метод требует больших затрат времени(поскольку на каждом этапе приходится вычислять модули всех чисел массива), и в то же время он не слишком точный, так как весь массив всегда масштабируется таким образом, чтобы модули всех его элементов не превышали 1/2 поэтому один разряд памятиданных не используется. Третий метод является самым точным, но здесь приходится повторно обрабатывать всю последовательность всякий раз, когда обнаруживается переполнение.

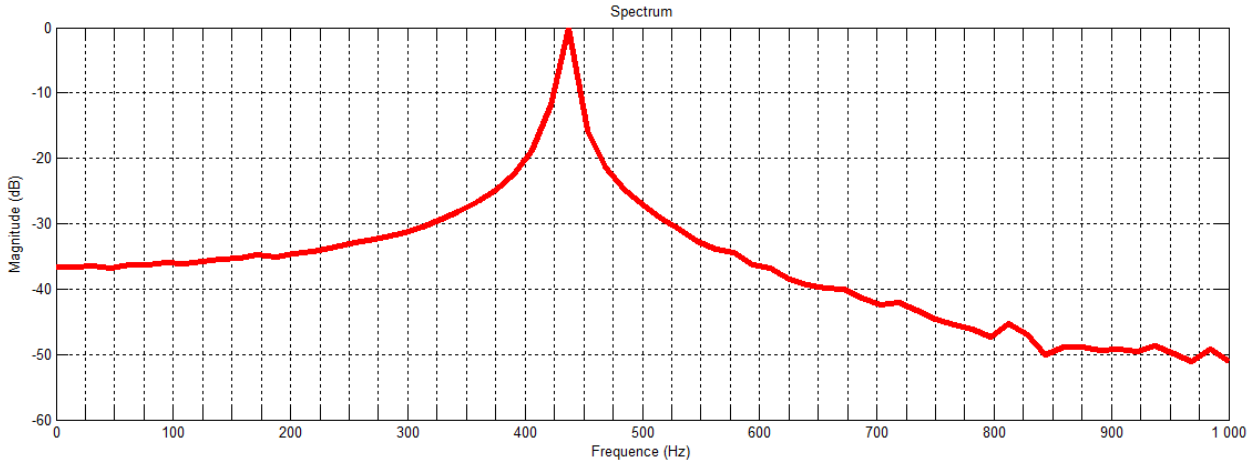
Уэлч разработал довольно простую методику расчета граничных значений уровня шума округления, возникающего при БПФ. Наличие шума округления обусловлено двумя факторами:

– Появлениемпогрешностиприокругленииипроизведения $W^p f_m(j)$. Если W^p и $f_m(j)$представлены b-разрядными числами (т.е. действительная и мнимая части W^p и $f_m(j)$содержат по bразрядов), то после округления каждого из четырех действительных произведений, входящих в $W^p f_m(j)$, до b-разрядного числа получается ошибка. Эта ошибка равномерно распределена на интервале $(-2^{-b}/2\,, 2^{-b}/2)$ и имеет нулевое среднее и дисперсию, равную$2^{-2b}/12$.

– Сдвигом суммы на один разряд вправо, если при сложении происходит переполнение. Если младший разряд, при сдвиге выходящий за пределы регистра, равен нулю, то ошибки не будет.

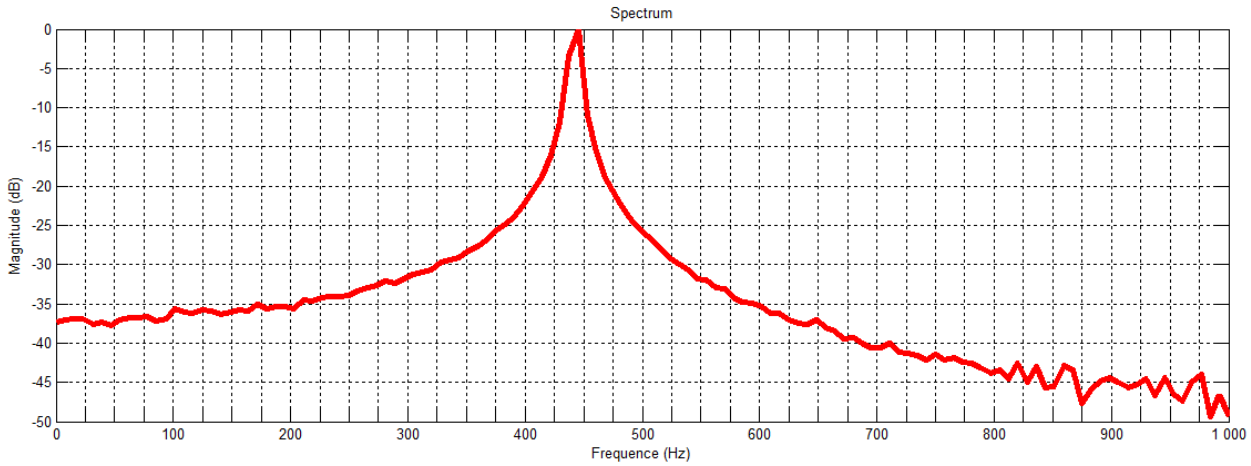
Для исследования эффектов квантования оценим величину ошибки, вносимую в результате округления промежуточных результатов. Для этого проведем

нормировку графиков спектров полученных в результате вычисления 128-, 256-, 512- и 1024-точечных БПФ. Полученные графики спектров представлены в децибелахна рисунках 29,30,31 и 32.



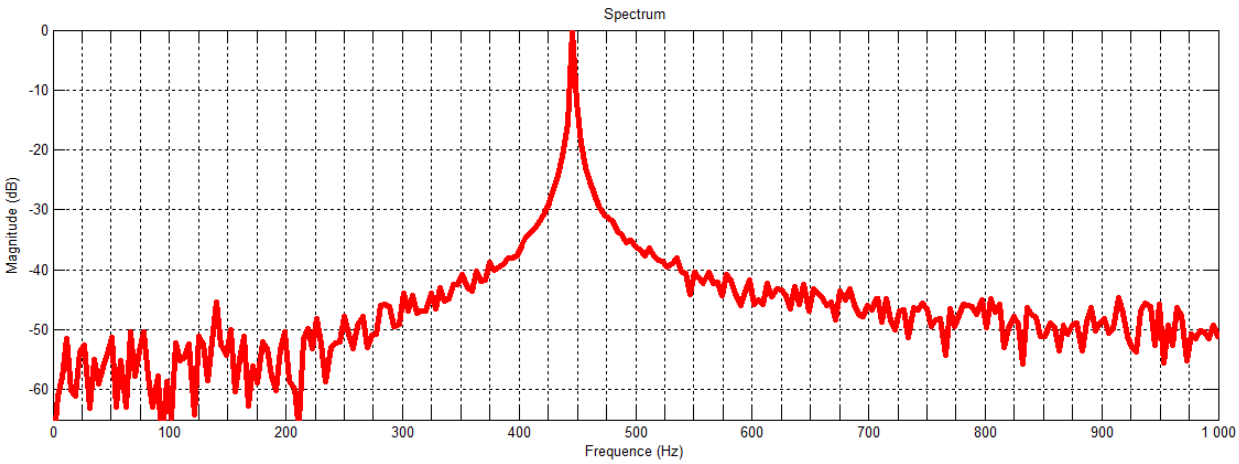
Спектр 29 – Спектр сигнала (128-БПФ)

Ошибка округления – 5 дБ



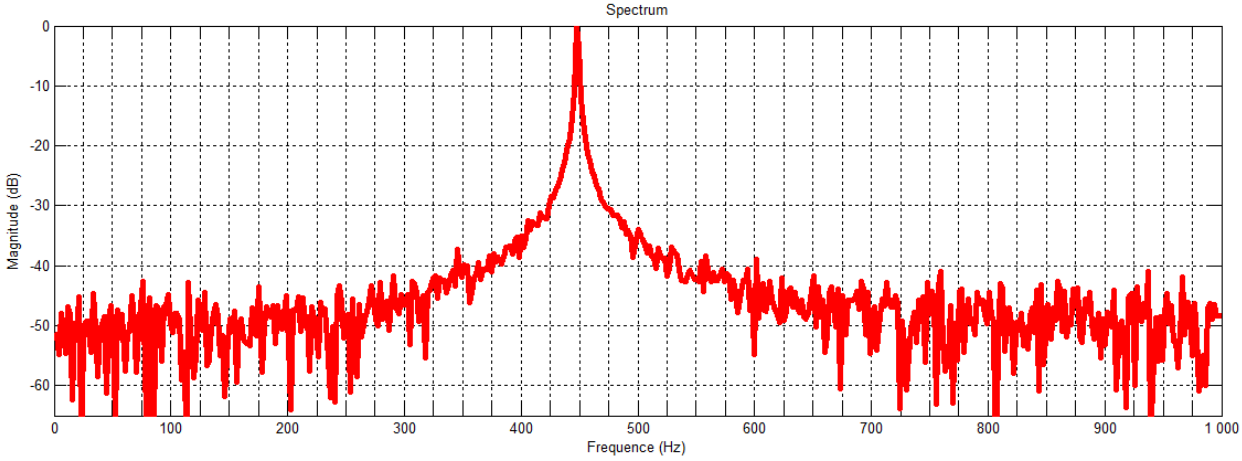
Спектр 30 – Спектр сигнала (256-БПФ)

Ошибка округления – 8 дБ



Спектр 31 – Спектр сигнала (512-БПФ)

Ошибка округления – 15 дБ



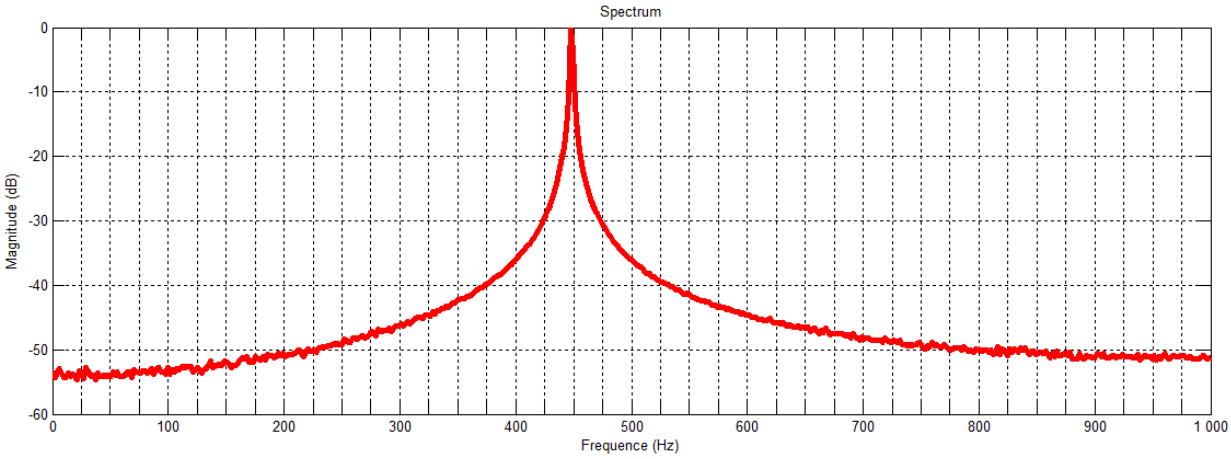
Спектр 32 – Спектр сигнала (1024-БПФ)

Из графика на рисунке 32 шум округления составляет около 7 дБ. С увеличением числа точек БПФ увеличивается шум, в результате накопления ошибки квантования.

На рисунке 33 показана усредненная ошибка квантования в зависимости от числа отсчетов сигнала.

Рисунок 33 – Зависимость ошибки квантования от числа точек БПФ

При исследовании алгоритма БПФ на сигнальном процессоре ADSP-TS-201 ошибка округления в исследуемом диапазоне незначительна и составляет около двух децибел (рисунок 34).



Спектр 34 – Спектр сигнала (1024-БПФ)

ЗАКЛЮЧЕНИЕ

В рамках данного исследования удалось найти и исследовать оптимальные алгоритмы вычислительных графов быстрого преобразования Фурье для сигнальных процессоровADSP-BF-537 и ADSP-TS-201.

В ходе исследования были реализованы следующие задачи:

– изучены и проанализированы статьи и литература по данной теме, рассмотрены методы оптимизации алгоритма БПФ;

– смоделированы исследуемые вычислительные графы БПФ средствами Matlab и VisualDSP;

– произведена оценка времени вычисления алгоритма в зависимости от количества отсчетов сигнала для процессора ADSP-BF-537 (Рисунок 25)и ADSP-TS-201 (Рисунок 27);

– так же произведена оценка времени вычисления различных алгоритмов с фиксированным количеством отсчетов сигнала;

– рассмотрено влияние шумов квантования.

На основании выполнения этих задач, можно с уверенностью сказать что цель исследования достигнута.

Как и было предположено в начале исследования, различные графы БПФ имеютразную производительность на различных устройствах.

Подтвердилась так же практическая значимость работы: полученные результаты можно использовать при проектировании систем цифровой обработки сигналов, а так же устройств передачи и приема сигналов, что поможет повысить производительность всей системы.

На основании, проведенного исследования можно сделать следующие выводы по производительности.

а. Все графы показали экспоненциальный рост процессорных циклов для вычисления БПФ в зависимости от числа точек быстрого преобразования Фурье;

б. При исследовании алгоритмов БПФ на сигнальном процессоре ADSP-BF-537удалось получить выигрыш в производительности в 22491 цикл (22,5%) – стандартный алгоритм в сравнении с алгоритмом без использования бит-реверсивной перестановки. Причиной этого является отсутствие необходимости переставлять отсчеты после выполнения вычисления, особенности самого вычислительного графа, и более простая программная реализация циклов при вычислении БПФ с помощью изучаемого графа.

Выигрыш в 8065 процессорных циклов (8%) показал алгоритм с постоянной структурой. Выигрыш в данном случае обеспечивается постоянной структурой графа на каждом вычислительном этапе, что упрощает адресацию данных и программную реализацию алгоритма.

в. В ходе исследования алгоритмов БПФ на сигнальном процессоре ADSP-TS-201был получен выигрыш в производительности в 2178 цикл (17,9%) –алгоритм Кули-Тьюки по сравнению с алгоритмом без использования бит-реверсивной перестановки. Это достигается за счет особенностей данного алгоритма, таких как прямой порядок отсчетов на входе и выходе преобразования, относительно простая программная реализация циклов при вычислении БПФ.

Выигрыш в 508 процессорных циклов (4,2%) показал алгоритм с постоянной структурой. Выигрыш в данном случае обеспечивается постоянной структурой графа на каждом вычислительном этапе, что упрощает адресацию данных, за счет того, что нумерация отсчетов сигнала на всех этапах не меняется, что позволяет упростить программную и аппаратную реализацию алгоритма.

г. Исследование производительности на сигнальных процессорах ADSP-BF-537 и ADSP-TS-201 показало кардинальный выигрыш процессора семейства TigerSharc. Это достигается благодаря высокопроизводительной архитектуреTigerSharc с плавающей точкой. Совмещая технологии VLIW и SIMD, поддерживает параллельное выполнение до 4 операций умножения 32-разрядных чисел с плавающей точкой, что вдвое больше, процессоров семейства BlackFin.

д. При проектировании алгоритмов на сигнальных микропроцессорах, ключевую роль играет программный код, с помощью которого реализуется исследуемый алгоритм. Использование ассемблера позволяет понять, как выполнятся программа, как хранятся переменные, как вызываются функции. Наглядно преимущество ассемблера перед C++ можно увидеть, сравнив результаты вычисления БПФ на рисунке 1 и рисунке 27.

е. Сравнив результаты работы программы на рисунках 26 и 28, можно сделать следующие выводы об эффектах квантования.

При работе с сигнальным процессором ADSP-BF-537 используется арифметика с фиксированной точкой, и во избежание переполнения необходимо урезать результат промежуточных вычислений, в связи с этим происходит накопление ошибки квантования. В нашем случае при вычислении 1024-точечного БПФ ошибка составила около 7 дБ.

Как видно из рисунка 28 сократить до минимума эффекты квантования можно с помощью использования арифметики с плавающей точкой, в нашем случае с помощью сигнального процессора ADSP-TS-201.

Данное исследование позволило выявить наиболее оптимальный алгоритм БПФ, ознакомится с современным программным обеспечением для цифровой обработки сигналов, изучить современные сигнальные процессоры и их язык ассемблер.

Немаловажное значение имеет исследование процессора ADSP-TS-201, аналог (сигнальный процессор 1967ВН028) которого был разработан в рамках программы по импортозамещению АО «ПКК Миландер». Рассмотренные алгоритмы можно использовать при работе с данным процессором.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рабинер, Л. Теория и применения цифровой обработки сигналов / Лоуренс Р. Рабинер, Бернард Гоулд; пер. с англ. А.Л. Зайцева, Э.Г. Назаренко, Н.Н. Тетёкина. – М.: Мир, 1978. – 848 с.

2. Куприянов, М.С. Цифровая обработка сигналов / М.С. Куприянов, Б.Д. Матюшкин. – 2-е изд., перераб. и доп. – СПб.: Политехника, 1999. 592 с.

3. Техническое обеспечение цифровой обработки сигналов: справочник / М.С. Куприянов, Б.Д. Матюшкин, В.Е. Иванова, Н.И. Матвиенко, Д.Ю. Усов. – СПб.: Наука и техника, 2000. – 752 с.

4. Безруков, А.В. Основы цифровой обработки сигналов: учебное пособие / А.В. Безруков. – СПб.: Изд-во БГТУ "Военмех" им. Д.Ф. Устинова, 2015. – 149 с.

5. Солонина, А.И. Алгоритмы и процессоры цифровой обработки сигналов / А.И. Солонина, Д.А. Улахович, Л.А. Яковлев. – СПб.: БХВ-Петербург, 2002. – 464 с.

6. Лайонс, Р. Цифровая обработка сигналов / Ричард Лайонс; пер. с англ. – 2-е изд. – М.: ООО «Бином-Пресс», 2006. – 656 с.

7. Айфичер, Э.С. Цифровая обработка сигналов: практический подход / Э.С. Айфичер, Б.У. Джервис; пер. с англ. – 2-е изд. – М.: Вильяме, 2004. – 992 с.

8. Гольденберг, Л.М. Цифровая обработка сигналов: учебное пособие для вузов /Л.М. Гольденберг, Б.Д. Матюшкин, М.Н. Поляк. – 2-е изд., перераб. и доп. – М.: Радио и связь, 1990. – 256 с.

9. Макклеплан, Д.Х. Применение теории чисел в цифровой обработке сигналов / Д.Х. Макклеплан, Ч.М. Рейдер; пер. с англ. – М.: Радио и связь, 1983. – 264 с.

10. Гольденберг, Л.М. Цифровая обработка сигналов: справочник / Л.М. Гольденберг, Б.Д. Матюшкин, М.Н. Поляк. – М.: Радио и связь, 1985. – 312 с.

11. Альтман, Е.А. Оптимизация вычислительной схемы быстрого преобразования Фурье / Е.А. Альтман // ОНВ. Информационные технологии. Радиотехника и связь. – 2008. – № 1 (64). – С. 149–151.

12. Ильин, А.А. Быстрые алгоритмы цифровой обработки сигналов: учебное пособие / А.А. Ильин, В.С. Титов, В.В. Евсюков. – Тула: Изд-во ТулГУ, 2004. – 125 с.

13. Ершова, О.В. Ошибки усечения результатов арифметических операций с фиксированной точкой в алгоритмах быстрого преобразования Фурье / О.В. Ершова, А.В. Чкан, Е.В. Кириченко, Е.А. Семерников // Известия ЮФУ. Технические науки. – 2014. – № 12 (161). – С. 138–148.

14. Галанина, Н.А. Моделирование оптимальных алгоритмов быстрого преобразования Фурье и их сравнительный анализ / Н.А. Галанина, А.Д. Ефимов // Вестник ЧГУ. – 2009. – № 2. – С. 200–209.

15. Галанина, Н.А. Методы и вычислительные устройства цифровой обработки сигналов в системе остаточных классов / Н.А. Галанина

16. Грицутенко, С.С. Эффективный алгоритм расчета быстрого преобразования Фурье для процессора NVCom-01 / С.С. Грицутенко, А.С. Сидоренко // ОНВ. Информационные технологии. – 2011. – № 3 (103). – С. 258–261.

17. Фот, Ж.А. Пути совершенствования процесса эскизного проектирования костюма в САПР / Ж.А. Фот // ОНВ. Информационные технологии. Радиотехника и связь. – 2008. – № 1 (64). – С. 151–154.

18. Марзук, С.Ф. Разработка процессоров быстрых спектральных преобразований для систем анализа геофизической информации / С.Ф. Марзук

19. Григорян, М.Б. Методы и устройства децимации сигнала с повышенным быстродействием для цифрового спектрального анализа / М.Б. Григорян

20. Цифровая обработка сигналов: учебное пособие по дисциплине «Цифровая обработка сигналов» / Ю.Н. Матвеев, К.К. Симончик, А.Ю. Тропченко, М.В. Хитров. – СПб.: СПбНИУ ИТМО, 2013. – 166 с.

21. Денисова, Л.А. Оптимизация системы регулирования с учетом чувствительности критерия качества / Л.А. Денисова // ОНВ. Информационные технологии. Радиотехника и связь. – 2008. – № 1 (64). – С. 145–149.

22. Тихонов, А.И. Алгоритм фильтрации, пример на основе БПФ: реферат / А.И. Тихонов. – Самара: 2010. – 41 с.

23. Петлин, А.А. Проектирование системы электроснабжения завода по производству электротермического оборудования, г.Бийск: дипломный проект / А.А. Петлин, рук-ль А.В. Кабышев. – Томск: ТПУ, 2016. – 99 с.

24. Калмыков, И.А. Теоретические основы вычислений в полиномиальной системе классов вычетов, ориентированных на построение отказоустойчивых систем: дис. ... д-ра техн. наук: / И.А. Калмыков. – Ставрополь, 2006. – 397 с.

25. Башкиров, А.В. Разработка моделей и алгоритмов верификации в САПР высокоразрядных СБИС спектрального анализа: дис. ... канд. техн. наук / А.В. Башкиров. – Воронеж, 2006. – 136 с.

26. Гусев, В.Ю. Методика фильтрации периодических помех цифровых изображений / В.Ю. Гусев, А.В. Крапивенко // Электронный журнал «Труды МАИ». – Вып. 50. – www.mai.ru/science/trudy/. – 13 с.

27. Простыми словами о преобразовании Фурье. – https://habrahabr.ru/post/196374/.

28. Солонина А.И., Клионский Д.М., Меркучева Т.В., Перов С.Н., Цифровая обработка сигналов и MATLAB, 2013г.

29. Стивен Смит Цифровая обработка сигналов. Практическое руководство для инженеров и научных работников. Додэка XXI, 2008. — 720 с. ISBN 978-5-94120-145-7, ISBN 0-750674-44-X

30. Юкио Сато Без паники! Цифровая обработка сигналов. Додэка XXI, 2010. — 176 с. ISBN 978-5-94120-251-5, ISBN 4-274-08674-7

31. СергиенкоА. Б. Цифровая обработка сигналов. — 2-е. — СПб.: Питер, 2007. — С. 751. — ISBN 5-469-00816-9.

32. Гольденберг Л. М. и др. Цифровая обработка сигналов. Справочник. — М.: Радио и связь, 1985. — 312 с.

33. Гольденберг Л. М. и др. Цифровая обработка сигналов. Учебное пособие для вузов. — М.: Радио и связь, 1990. — 256 с.

34. Оппенгейм А., Шафер Р. Цифровая обработка сигналов. Изд. 2-е, испр. — М.: Техносфера, 2007. — 856 с. ISBN 978-5-94836-135-2

35. Оппенгейм А. В., Шафер Р. В. Цифровая обработка сигналов. — М.: Связь, 1979. — 416 с.

36. Рабинер Л., Гоулд Б. Теория и применение цифровой обработки сигналов. — М.: Мир, 1978. — 848 с.

37. Глинченко А. С. Цифровая обработка сигналов. В 2 ч. — Красноярск: Изд-во КГТУ, 2001. — 383 с.

38. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов. — М.: Мир, 1989. — 448 с.

39. Даджион Д., Мерсеро Р. Цифровая обработка многомерных сигналов. — М.: Мир, 1988. — 488 с.

40. Макс Ж. Методы и техника обработки сигналов при физических измерениях. В 2-х тт. — М.: «Мир», 1983.

41. *Марпл-мл. С. Л.* Цифровой спектральный анализ и его приложения. — М.: МИР, 1990. — С. 584.

42. Хемминг Р. В. Цифровые фильтры. — М.: Недра, 1987. — 221 с.

43. Дьяконов В. П. MATLAB 6.5 SP1/7.0 + Simulink 5/6/ Обработка сигналов и проектирование фильтров. — М.: СОЛОН-Пресс, 2005. — 676 с.

44. Дьяконов В. П. Вейвлеты. От теории к практике. Изд.е 2-ое дополненное и переработанное. — М.: СОЛОН-Пресс, 2005. — 400 с.

45. Дьяконов В. П. Современная осциллография и осциллографы. — М.: СОЛОН-Пресс, 2004. — 320 с.

46. Афонский А. А., Дьяконов В. П. Измерительные приборы и массовые электронные измерения / Под ред. проф. В. П. Дьяконова. — М.: СОЛОН-Пресс, 2007. — 544 с.

47. Афонский А. А., Дьяконов В. П. Цифровые анализаторы спектра, сигналов и логики / Под ред. проф. В. П. Дьяконова. — М.: СОЛОН-Пресс, 2009. — 248 с.

48. Захаров С. И. Повышение эффективности вибродиагностики механизмов с помощью экспертных систем. М: Машиностроение//Вестник машиностроения, 2008, № 6, стр.31-32.

49. Cooley James W. and Tukey John W. An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation, 1965 p. 297–301.

50. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов. Москва, Мир, 1989.

51. Нуссбаумер Г. Быстрое преобразование Фурье и алгоритмы вычисления сверток. Москва, Радио и связь, 1985.

52. Оппенгейм А., Шафер Р. Цифровая обработка сигналов. Москва, Техносфера, 2006.

53. Гольденберг Л.М., Матюшкин Б.Д., Поляк М.Н. Цифровая обработка сигналов. Москва, Радио и связь, 1985.

54. Сергиенко А.Б. Цифровая обработка сигналов. Санкт-Петербург, Питер, 2002.

55. The Scientist and Engineer's Guide to Digital Signal Processing By Steven W. Smith, Ph.D. Hard Cover, 2007 662 p. http://www.dspguide.com/errata.htm

ПРИЛОЖЕНИЕА

Фрагмент листинга программы, реализующей алгоритм Кули-Тьюки на процессоре ADSP-BF-537

```
// Loop for N/2, computing one butterfly per iteration
LSETUP(.stage1_ns_start, .stage1_ns_end) LC0 = P0 >> 1;
.stage1_ns_start:
        R0 = R0 + R2, R2 = R0 - R2 (S) || R3 = [I1++M0];
        R1 = R1 + R3, R3 = R1 - R3 (S)              || [I2++] = R0;
        R7.L = EXPADJ (R0, R7.L )                   || [I2++] = R1;
        R5.L = EXPADJ (R1, R5.L )     || R0 = [I0++M0] || [I2++] = R2;
        R7.L = EXPADJ (R2, R7.L )     || R2 = [I0++M0] || [I2++] = R3;
.stage1_ns_end:
        R5.L = EXPADJ (R3, R5.L )     || R1 = [I1++M0];
    JUMP .first_stage_finish;
.ALIGN 4
.first_stage_scaling:
    // Alternative code, apply scaling
    R5 = PACK( R7.H, R7.L )
    || R0 = [I0++M0]
    || I1 += 4;
    R0 = R0 >>> 1
    || R2 = [I0++M0];
    // Loop for N/2, computing one butterfly per iteration
LSETUP(.stage1_start, .stage1_end) LC0 = P0 >> 1;
.stage1_start:
        R2 = R2 >>> 1                || R1 = [I1++M0];
        R0 = R0 + R2, R2 = R0 - R2 (S) || R3 = [I1++M0];
        R1 >>>= 1;
        R3 >>>= 1;
        R1 = R1 + R3, R3 = R1 - R3 (S);
        R7.L = EXPADJ (R2, R7.L )                   || [I2++] = R0;
        R5.L = EXPADJ (R0, R5.L )                   || [I2++] = R1;
        R7.L = EXPADJ (R3, R7.L )     || R0 = [I0++M0] || [I2++] = R2;
        R5.L = EXPADJ (R1, R5.L )     || R2 = [I0++M0] || [I2++] = R3;
.stage1_end:
        R0 >>>= 1;
.ALIGN 4
.first_stage_finish:
    R7 = R5.L * R7.L (IS)         // If either R5 or R7 == 0 => R7 = 0
    || [SP+CURRENT_BLOCKEXPO] = R6;// Store block exponent
BITTGL( R7, 0 );              // IF EXPADJ == 0 => scale (R7 = 1)
                              //            otherwise don't (R7 = 0)
    R6 = [SP+(IN_SCALING+SIZE_TEMP_DATA)];
                              // Retrieve scaling method
    CC = R6 == 2;                // Test for dynamic scaling
    IF CC R6 = R7;               // R6 = 1 if scaling, 0 otherwise
  /* Second Stage */
    I0 = B0;                     // Read pointer &out[0].re
    I1 = B0;                     // Read pointer &out[1].re
                              // I2 = Write pointer out
```

```
                        //      set to &out[0]
    P0 = P0 >> 2;               // Set loop counter (=N/4)
    M0 = 12;                    // Stride real[i] ->real[i+2]
                        //      imag[i] ->imag[i+2]
    R7 = W[SP+CONST_VAL] (Z);
    CC = R6 == 1;
    IF CC JUMP .second_stage_scaling;
#if WA_05000428
    // Speculative read from L2 by Core B may cause a write to L2 to fail
    NOP;
    NOP;
    NOP;
#endif
    // Alternative code, no scaling
    R5 = PACK( R7.H, R7.L )
    || R0 = [I0++]
    || I1 += 4;
    R1 = [I0++M0]
    || I1 += 4;
    R2 = [I0++];
    R0 = R0 + R2, R2 = R0 - R2 (S) || R3 = [I0++M0];
    // Loop for N/4, compute two butterflies in parallel
LSETUP(.stage2_ns_start, .stage2_ns_end) LC0 = P0;
.stage2_ns_start:
      R1 = R1 + R3, R3 = R1 - R3 (S)              || [I2++] = R0;
      R7.L = EXPADJ (R0, R7.L )     || R0 = [I1++]    || [I2++M0] = R1;
      R5.L = EXPADJ (R1, R5.L )     || R1 = [I1++M0]  || [I2++] = R2;
      R7.L = EXPADJ (R2, R7.L )     || R2 = [I1++]    || [I2] = R3;
      R5.L = EXPADJ (R3, R5.L )     || R3 = [I1++M0]  || I2 -= M0;
      R0 = R0 + R3, R3 = R0 - R3 (S);
      R2 = R1 + R2, R1 = R1 - R2 (S)              || [I2++] = R0;
      R7.L = EXPADJ (R0, R7.L )                   || [I2++M0] = R1;
      R5.L = EXPADJ (R1, R5.L )     || R0 = [I0++]    || [I2++] = R3;
      R7.L = EXPADJ (R2, R7.L )     || R1 = [I0++M0]  || [I2++] = R2;
      R5.L = EXPADJ (R3, R5.L )     || R2 = [I0++];
.stage2_ns_end:
      R0 = R0 + R2, R2 = R0 - R2 (S) || R3 = [I0++M0];
    JUMP .second_stage_finish;
.ALIGN 4
.second_stage_scaling:
    // Alternative code, apply scaling
    R5 = PACK( R7.H, R7.L )
    || R0 = [I0++]
    || I1 += 4;
    R0 = R0 >>> 1
    || R1 = [I0++M0]
    || I1 += 4;
    R1 = R1 >>> 1
    || R2 = [I0++];
    // Loop for N/4, compute two butterflies in parallel
```

ПРИЛОЖЕНИЕ Б

Фрагмент листинга программы, реализующей алгоритм с постоянной структурой на процессоре ADSP-BF-537

```
LSETUP(.stage2_start, .stage2_end) LC0 = P0;
.stage2_start:
        R2 >>>= 1;
        R0 = R0 + R2, R2 = R0 - R2 (S) || R3 = [I0++M0];
        R3 >>>= 1;
        R1 = R1 + R3, R3 = R1 - R3 (S)              || [I2++] = R0;
        R7.L = EXPADJ (R0, R7.L )                   || [I2++M0] = R1;
        R5.L = EXPADJ (R1, R5.L )     || R0 = [I1++]   || [I2++] = R2;
        R7.L = EXPADJ (R2, R7.L )     || R1 = [I1++M0] || [I2] = R3;
        R5.L = EXPADJ (R3, R5.L )     || R2 = [I1++]   || I2 -= M0;
        R0 = R0 >>> 1                 || R3 = [I1++M0];
        R1 >>>= 1;
        R2 >>>= 1;
        R3 >>>= 1;
        R0 = R0 + R3, R3 = R0 - R3 (S);
        R2 = R1 + R2, R1 = R1 - R2 (S);
        R7.L = EXPADJ (R0, R7.L )                   || [I2++] = R0;
        R5.L = EXPADJ (R1, R5.L )     || R0 = [I0++]   || [I2++M0] = R1;
        R7.L = EXPADJ (R2, R7.L )     || R1 = [I0++M0] || [I2++] = R3;
        R5.L = EXPADJ (R3, R5.L )     || R2 = [I0++]   || [I2++] = R2;
        R0 >>>= 1;
.stage2_end:
        R1 >>>= 1;
.ALIGN 4
.second_stage_finish:
      R7 = R5.L * R7.L (IS)         // If either R5 or R7 == 0 => R7 = 0
      || R5 = [SP+CURRENT_BLOCKEXPO];// Retrieve block exponent
      R5 = R5 + R6 (S)              // Increment block exponent
      || R6 = [SP+(IN_SCALING+SIZE_TEMP_DATA)];
                                    // Retrieve scaling method
BITTGL( R7, 0 );                    // IF EXPADJ == 0 => scale (R7 = 1)
                                    //            otherwise don't (R7 = 0)
      CC = R6 == 2;                 // Test for dynamic scaling
      IF CC R6 = R7;                // R6 = 1 if scaling, 0 otherwise
      [SP+CURRENT_BLOCKEXPO] = R5;   // Store updated block exponent

   /* Remaining Stages */
      P1 = 4;                       // Set loop counter butterflies
      R3 = [SP+(IN_FFTSIZE+SIZE_TEMP_DATA)];
      R4 = [SP+(IN_STRIDE+SIZE_TEMP_DATA)];
      R3 *= R4;                     // Twiddle stride = (((N/2)/4)*8)*stride
      P5 = R3;
      R4 = R3 << 2
      || R7 = [SP+CONST_VAL];
      I3 = B3;
      L3 = R4;                      // Twiddle table as circular buffer
      I0 = B0;                      // Read/Write pointer upper half butterfly
```

```
    I1 = B0;                    // Read pointer lower half butterfly
                            // I2 = Write pointer lower half butterfly
                            //      set to &out[0]
    P2 =  1;                    // Const value
    P4 = 32;                    // Offset real[i] ->real[next butter]
    P3 = P0 << 2;               // Loop counter scanning (==N)
.ALIGN 4
.iter_stages:
        // Loop for log2(fft_size) - 2
        M2 = P5;                // Update twiddle stride
        P0 = P0 >> 1;           // Set loop counter stages
        M1 = P4;                // Update offset
        P1 += -1;               // First butterfly unrolled
        CC = R6 == 1;
        IF CC JUMP .groups_with_scaling;
#if WA_05000428
        // Speculative read from L2 by Core B may cause a write to L2 to fail
        NOP;
        NOP;
#endif
        // Loop for ngroups
        // where ngroups = (fft_size)/8   (first iteration .iter_stages)
        //               = (fft_size)/16  (second iteration .iter_stages)
        //               = (fft_size)/32  (third iteration .iter_stages)
        //               ...
        //               = 1              (last iteration .iter_stages)
LSETUP(.groups_ns_start, .groups_ns_end) LC0 = P0;
.groups_ns_start:
          // First butterfly unrolled
          R0 = [I0++] || I1 += M1;
          R2 = [I1++] || I2 += M1;
          R0 = R0 + R2, R2 = R0 - R2 (S) || R1 = [I0--] || I3 += M2;
          R3 = [I1++] || [I2++] = R2;
          R1 = R1 + R3, R0 = R1 - R3 (S) || R4 = [I3++] || [I0++] = R0;
          R2 = [I1++] || R5 = [I3--];
          // Loop for nbutters=  3  (first iteration .iter_stages)
          //                =  7  (second iteration .iter_stages)
          //                = 15  (third iteration .iter_stages)
          //                = (fft_size / 2) - 1}
          //                     (last iteration .iter_stages)
LSETUP(.butterflies_ns_start, .butterflies_ns_end) LC1 = P1;
.butterflies_ns_start:
            // Real part of butterfly
            A1 = R4.H * R2.L (M)        || R3 = [I1++] || [I0++] = R1;
#if defined(__ADSPLPBLACKFIN__)
            R1 = -R3 (S);
#else
            R1 = R1 -|- R1 (S);
            R1 = R1 - R3 (S);#endif
            A1 -= R5.H * R3.L (M);
            A1 += R1.H * R5.L (M), A0  = R1.L * R5.L (FU);
            A1 += R7.H * R7.H (M), A0 += R7.H * R7.H (FU);
            A1 += R2.H * R4.L (M), A0 += R2.L * R4.L (FU);
```

ПРИЛОЖЕНИЕВ

Фрагмент листинга программы, реализующей алгоритм без использования бит-реверсивной перестановки на процессоре ADSP-BF-537

```
LSETUP(.groups_ns_start, .groups_ns_end) LC0 = P0;
.groups_ns_start:
        // First butterfly unrolled
        R1 = [I0]     || I1 += M1;
        R0 = [I1++M0] || I2 += M1;
        R1 = R1 + R0, R0 = R1 - R0 (S) || R2 = [I1++]   || I3 += M2;
        R1 = R1 -|- R1 || R3 = [I1--]  || [I0++] = R1;
        R0 = R0 -|- R0 || R4 = [I3++]  || [I2++] = R0;
        // Loop for nbutters=  3  (first iteration .iter_stages)
        //                =  7  (second iteration .iter_stages)
        //                = 15  (third iteration .iter_stages)
        //                ...
        //                = (fft_size / 2) - 1}
        //                     (last iteration .iter_stages)
LSETUP(.butterflies_ns_start, .butterflies_ns_end) LC1 = P1;
.butterflies_ns_start:            // Real part of butterfly
#if defined(__ADSPLPBLACKFIN__)
R1  = -R3 (S)             || R5 = [I3--]   || [I0++] = R1;
#else
          R1 = R1 -|- R1 (S)         || R5 = [I3--]   || [I0++] = R1;
          R1 = R1 - R3 (S);
#endif
A1  = R4.H * R2.L (M);
          A1 += R1.H * R5.L (M), A0  = R1.L * R5.L (FU);
          A1 += R7.H * R7.H (M), A0 += R7.H * R7.H (FU);
          A1 -= R5.H * R3.L (M);
          A1 += R2.H * R4.L (M), A0 += R2.L * R4.L (FU);
          A0 = A0 >> 16;
          A0 += A1;
          A0 = A0 >>> 15            || R1 = [I0]     || [I2++] = R0;
          A0 += R2.H * R4.H;
          R0 = (A0 -= R3.H * R5.H);
          R1 = R1 + R0, R0 = R1 - R0 (S);
          // Imaginary part of butterfly
A1  = R5.H * R2.L (M);
          A1 += R7.H * R7.H (M), A0  = R7.H * R7.H;
          A1 += R2.H * R5.L (M), A0 += R2.L * R5.L (FU);
A0 = A0 >> 16;
          A0 += A1                  || [I0++] = R1   || I1 += M0;
          A0 = A0 >>> 15            || R1 = [I0]     || I3 += M2;
          A0 += R2.H * R5.H         || R2 = [I1++]   || [I2++] = R0;
          R0 = (A0 += R3.H * R4.H)    || R3 = [I1--]   || R4 = [I3++];
.butterflies_ns_end:
          R1 = R1 + R0, R0 = R1 - R0 (S);
        [I0++M1] = R1 || I3 -= 4;
.groups_ns_end:
      JUMP .groups_finish;
```

ПРИЛОЖЕНИЕГ

Фрагмент листинга программы, реализующей алгоритм Кули-Тьюки на процессореADSP-TS-201

```
.align_code 4;
_Stages1and2Loop:
                r1:0=q[k0+k6]; q[j2+=4]=yr23:20; fr16=r0+r4, fr24=r0-r4;;
                r3:2=q[k2+k6]; q[j3+=4]=xr23:20; fr17=r2+r5, fr25=r2-r5;;
                r5:4=q[k1+k6]; q[j14+=4]=yr31:28; fr18=r18+r26, fr26=r18-r26;;
                r7:6=q[k3+k6]; q[j15+=4]=xr31:28; fr19=r19+r27, fr27=r19-r27;;

                k6=k6+k5 (br); q[j0+=4]=yr19:16; fr0=r0+r2, fr20=r0-r2;;
                r9:8=q[k0+k6]; q[j1+=4]=xr19:16; fr2=r1+r3, fr29=r1-r3;;
                r11:10=q[k2+k6]; q[j12+=4]=yr27:24; fr4=r4+r6, fr21=r4-r6;;
                r13:12=q[k1+k6]; q[j13+=4]=xr27:24; fr5=r5+r7, fr28=r5-r7;;

                r15:14=q[k3+k6]; fr18=r8+r10, fr22=r8-r10;;
                k6=k6+k5 (br); fr19=r9+r11, fr31=r9-r11;;
                fr26=r12+r14, fr23=r12-r14;;
                fr27=r13+r15, fr30=r13-r15;;

                fr20=r20+r28, fr28=r20-r28;;
                fr29=r29+r21, fr21=r29-r21;;
                fr22=r22+r30, fr30=r22-r30;;
.align_code 4;
                if NLC0E, jump _Stages1and2Loop;
                fr31=r31+r23, fr23=r31-r23;;

        q[j2+=4]=yr23:20; fr16=r0+r4, fr24=r0-r4;;
        q[j3+=4]=xr23:20; fr17=r2+r5, fr25=r2-r5;;
        q[j14+=4]=yr31:28; fr18=r18+r26, fr26=r18-r26;;
        q[j15+=4]=xr31:28; fr19=r19+r27, fr27=r19-r27;;

        q[j0+=4]=yr19:16;;
        q[j1+=4]=xr19:16;;
        q[j12+=4]=yr27:24;;
        q[j13+=4]=xr27:24;;

//************************* Stages 3 to Log2(N)-1  ********************************

        j0=j31+j6; k5=k31+0;;

.align_code 4;
_StageLoop:
                yr3:0=q[j0+=4];   k3=k5 and k4;;                                        // F1,    K1
                xr3:0=q[j0+=4];   r5:4=l[k7+k3];;                                       // F2,    K2
                LC0=k10;          k5=k5+k9;;                                    //       K3,    M1

                yr11:8=q[j0+=4];  k3=k5 and k4;    fr6=r2*r4;;                                  // F1+,   K1+
                xr11:8=q[j0+=4];  r13:12=l[k7+k3]; fr7=r3*r5;;                                  // F2+,   K2+,   M2
                            fr14=r2*r5;;                                              //              M3
                j1=j31+j5;        k5=k5+k9;        fr6=r10*r12;  fr16=r6-r7;;                   //       K3+,   M1+,   A1

                yr23:20=q[j0+=4]; k3=k5 and k4;    fr15=r3*r4;;                                 // F1++,  K1++,  M4
                xr23:20=q[j0+=4]; r5:4=l[k7+k3];   fr7=r11*r13;;                                // F2++,  K2++,  M2+
                                  fr14=r10*r13; fr17=r14+r15;;                                  //             M3+,   A2
                j2=j1+j11;        k5=k5+k9;        fr6=r22*r4;   fr18=r6-r7;;                   //       K3++,  M1++,  A1+

                yr31:28=q[j0+=4]; k3=k5 and k4;    fr15=r11*r12; fr24=r0+r16, fr26=r0-r16;;                  // F1+++, K1+++, M4+,   A3
                xr31:28=q[j0+=4]; r13:12=l[k7+k3]; fr7=r23*r5;   fr25=r1+r17, fr27=r1-r17;;                  // F2+++, K2+++, M2++,  A4
q[j1+=4]=r25:24;            fr14=r22*r5;  fr19=r14+r15;;                                    // S1,          M3++,  A2+
                .align_code 4;
_BflyLoop:
                        q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r30*r12;  fr16=r6-r7;;                  // S2----,K3-,   M1-,   A1--

                        yr3:0=q[j0+=4];   k3=k5 and k4;    fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;  // F1,    K1,    M4--,  A3---
                        xr3:0=q[j0+=4];   r5:4=l[k7+k3];   fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;  // F2,    K2,    M2-,   A4---
                        q[j1+=4]=r25:24;                   fr14=r30*r13; fr17=r14+r15;;                  // S1---,        M3-,   A2--
                        q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r2*r4;    fr18=r6-r7;;                  // S2---, K3,    M1,    A1-

                        yr11:8=q[j0+=4];  k3=k5 and k4;    fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;  // F1+,   K1+,   M4-,   A3--
                        xr11:8=q[j0+=4];  r13:12=l[k7+k3]; fr7=r3*r5;    fr25=r21+r17, fr27=r21-r17;;  // F2+,   K2+,   M2,    A4--
q[j1+=4]=r25:24;                fr14=r2*r5;   fr19=r14+r15;;                    // S1--,         M3,    A2-
                        q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r10*r12;  fr16=r6-r7;;                  // S2--,  K3+,   M1+,   A1

                        yr23:20=q[j0+=4]; k3=k5 and k4;    fr15=r3*r4;   fr24=r28+r18, fr26=r28-r18;;  // F1++,  K1++,  M4,    A3-
                        xr23:20=q[j0+=4]; r5:4=l[k7+k3];   fr7=r11*r13;  fr25=r29+r19, fr27=r29-r19;;  // F2++,  K2++,  M2+,   A4-
                        q[j1+=4]=r25:24;                   fr14=r10*r13; fr17=r14+r15;;                  // S1-,          M3+,   A2
                        q[j2+=4]=r27:26;  k5=k5+k9;        fr6=r22*r4;   fr18=r6-r7;;                  // S2-,   K3++,  M1++,  A1+

                        yr31:28=q[j0+=4]; k3=k5 and k4;    fr15=r11*r12; fr24=r0+r16,  fr26=r0-r16;;  // F1+++, K1+++, M4+,   A3
                        xr31:28=q[j0+=4]; r13:12=l[k7+k3]; fr7=r23*r5;   fr25=r1+r17,  fr27=r1-r17;;  // F2+++, K2+++, M2++,  A4
.align_code 4;
```

```
        if NLC0E, jump _BflyLoop;
        q[j1+=4]=r25:24; fr14=r22*r5;  fr19=r14+r15;;                                   // S1,         M3++,  A2+

                q[j2+=4]=r27:26;                fr6=r30*r12;  fr16=r6-r7;;              // S2----,     M1-, A1--

                j0=j31+j5;                  fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;   //          M4--,  A3---, swap ping-pong pointers
                j5=j31+j6;                  fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;   //          M2-,   A4---
                q[j1+=4]=r25:24;                fr14=r30*r13; fr17=r14+r15;;            // S1---,      M3-,   A2--
                q[j2+=4]=r27:26;                        fr18=r6-r7;;                    // S2---,          A1-

                j6=j31+j0;                  fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;  //          M4-,   A3--
                                                fr25=r21+r17, fr27=r21-r17;;            //                 A4--
q[j1+=4]=r25:24;                                        fr19=r14+r15;;                  // S1--,           A2-
                q[j2+=4]=r27:26;                fr24=r28+r18, fr22=r28-r18;;            // S2--            A3-

                j0=j31+j6;                  fr25=r29+r19, fr23=r29-r19;;                //                 A4-
                q[j1+=4]=r25:24; k5=k31+0;;                                             // S1-
.align_code 4;
                if NLC1E, jump _StageLoop;
                q[j2+=4]=r23:22; k4=ashiftr k4;;                                        // S2-, shift the mask

//******************************** Last stage **********************************
  k9 = ashiftr k9;;//in this manner any MAX_FFT_SIZE can be used

        yr3:0=q[j0+=4]; yr5:4 = l[k7+=k9];;                                     // F1,

        xr3:0=q[j0+=4];   xr5:4=l[k7+=k9];;                                     // F2,    K2
        j1=j31+j7;        fr6=r2*r4; LC0=k10;;                          //          M1


        yr11:8=q[j0+=4];  yr13:12=l[k7+=k9];;                           // F1+
        xr11:8=q[j0+=4];  xr13:12=l[k7+=k9]; fr7=r3*r5;;                        // F2+,   K2+,   M2
  j2=j1+j11;                                    fr14=r2*r5;;            //          M3
                        fr6=r10*r12;  fr16=r6-r7;;                              //          M1+,   A1

        yr23:20=q[j0+=4]; yr5:4=l[k7+=k9];  fr15=r3*r4;;                        // F1++,       M4
        xr23:20=q[j0+=4]; xr5:4=l[k7+=k9];   fr7=r11*r13;;                      // F2++,  K2++,  M2+
                        fr14=r10*r13; fr17=r14+r15;;                            //          M3+,   A2
                        fr6=r22*r4;   fr18=r6-r7;;                      //          M1++,  A1+

        yr31:28=q[j0+=4]; yr13:12=l[k7+=k9];fr15=r11*r12; fr24=r0+r16, fr26=r0-r16;;           // F1+++,      M4+,   A3
        xr31:28=q[j0+=4]; xr13:12=l[k7+=k9]; fr7=r23*r5;   fr25=r1+r17, fr27=r1-r17;;          // F2+++, K2+++, M2++,  A4
q[j1+=4]=r25:24;                fr14=r22*r5;  fr19=r14+r15;;                            // S1,         M3++,  A2+

.align_code 4;
_BflyLastLoop:
                q[j2+=4]=r27:26;                fr6=r30*r12;  fr16=r6-r7;;              // S2----,     M1-,   A1--

                yr3:0=q[j0+=4];   yr5:4=l[k7+=k9];  fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;  // F1,         M4--,  A3---
                xr3:0=q[j0+=4];   xr5:4=l[k7+=k9];  fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;  // F2,    K2,    M2-,   A4---
                q[j1+=4]=r25:24;                fr14=r30*r13; fr17=r14+r15;;            // S1---,      M3-,   A2--
                q[j2+=4]=r27:26;                fr6=r2*r4;    fr18=r6-r7;;              // S2---,      M1,    A1-

                yr11:8=q[j0+=4];  yr13:12=l[k7+=k9];fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;  // F1+,        M4-,   A3--
                xr11:8=q[j0+=4];  xr13:12=l[k7+=k9];fr7=r3*r5;    fr25=r21+r17, fr27=r21-r17;;  // F2+,   K2+,   M2,    A4--
q[j1+=4]=r25:24;                        fr14=r2*r5;   fr19=r14+r15;;                    // S1--,       M3,    A2-
                q[j2+=4]=r27:26;                fr6=r10*r12;  fr16=r6-r7;;              // S2--,       M1+,   A1

                yr23:20=q[j0+=4]; yr5:4=l[k7+=k9];  fr15=r3*r4;   fr24=r28+r18, fr26=r28-r18;;  // F1++,       M4,    A3-
                xr23:20=q[j0+=4]; xr5:4=l[k7+=k9];  fr7=r11*r13;  fr25=r29+r19, fr27=r29-r19;;  // F2++,  K2++,  M2+,   A4-
                q[j1+=4]=r25:24;                fr14=r10*r13; fr17=r14+r15;;            // S1-,        M3+,   A2
                q[j2+=4]=r27:26;                fr6=r22*r4;   fr18=r6-r7;;              // S2-,        M1++,  A1+

                yr31:28=q[j0+=4];yr13:12=l[k7+=k9]; fr15=r11*r12; fr24=r0+r16,  fr26=r0-r16;;  // F1+++,      M4+,   A3
                xr31:28=q[j0+=4];xr13:12=l[k7+=k9]; fr7=r23*r5;   fr25=r1+r17,  fr27=r1-r17;;  // F2+++, K2+++, M2++,  A4
.align_code 4;
        if NLC0E, jump _BflyLastLoop;
        q[j1+=4]=r25:24;  fr14=r22*r5;  fr19=r14+r15;;                                  // S1,         M3++,  A2+

        q[j2+=4]=r27:26;                fr6=r30*r12;  fr16=r6-r7;;                      // S2----,     M1-, A1--

                        fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;                       //          M4--,  A3---
                        fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;                       //          M2-,   A4---
        q[j1+=4]=r25:24;                fr14=r30*r13; fr17=r14+r15;;                    // S1---,      M3-,   A2--
        q[j2+=4]=r27:26;                        fr18=r6-r7;;                            // S2---,          A1-

                        fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;                      //          M4-,   A3--
                                fr25=r21+r17, fr27=r21-r17;;                            //                 A4--
q[j1+=4]=r25:24;                                        fr19=r14+r15;;                  // S1--,           A2-
        q[j2+=4]=r27:26;;                                                               // S2--

                                fr24=r28+r18, fr26=r28-r18;;                            //                 A3-
                                fr25=r29+r19, fr27=r29-r19;;                            //                 A4-

        comp(j11,COMPLEX);;                                                                                                     //
        ifjeq, jump _FFTEpilogue;;
// IfComplex, done
```

ПРИЛОЖЕНИЕ Д

Фрагмент листинга программы, реализующей алгоритм с постоянной структурой на процессоре ADSP-TS-201

```
//************************************ Stages ************************************
    a k10 = k31 + N;; // twiddles stride
            k11 = k31 + N/2;; // butterflies/group
            j12 = j31 + 1;; // groups
            j10 = j31 + 2;; // width of butterfly
            j11 = j31 + 4;; // butterfly stride
            k20 = k31 + STAGES;;
    _stages_loop:
            j0 = j31 + j29;; // j0 ->internal_buff
            k0 = k31 + k30;; // k0 -> twiddles
            j13 = j31 + 0;;
            LC1 = j12;;
                    _group_loop:
                            xr1:0 = L[k0 += k10];; // xr0=cos, xr1=-sin
                            j1 = j0 + j10;; // j1 -> second input to butterfly
                            LC0 = k11;;
                                    _butterfly_loop:
                                            xr3:2 = L[j0 += 0];;            // xr2=Re1, xr3=Im1
                                            xr5:4 = L[j1 += 0];;            // xr4=Re2, xr5=Im2
                                            xfr6 = r4 * r0;;                // xr6=Re2*cos
                                            xfr7 = r5 * r1;;                // xr7=Im2*sin
                                            xfr8 = r6 - r7;;                // xr8=Re(z2*twid)
                                            xfr9 = r4 * r1;;                // xr6=Re2*sin
                                            xfr10 = r5 * r0;;                // xr7=Im2*cos
                                            xfr11 = r9 + r10;;               // xr8=Im(z2*twid)
                                            xfr12 = r2 + r8, fr14 = r2 - r8;;          // Re(butterfly)
                                            xfr13 = r3 + r11, fr15 = r3 - r11;;        // Im(butterfly)
                    L[j0 += j11] = xr13:12;;
                    L[j1 += j11] = xr15:14;;
    if NLC0E, jump _butterfly_loop (NP);;
                    j13 = j13 + 2;;
                    j0 = j29 + j13;;                // offset for the next group
    if NLC1E, jump _group_loop (NP);;
            k10 = lshiftr k10;;             // twiddles stride
            k11 = lshiftr k11;;             // butterflies/group
            j12 = j12 + j12;;                       // groups
            j10 = j10 + j10;;                       // width of butterfly
            j11 = j11 + j11;;                       // butterfly stride
            k20 = k20 - 1;;
    if NKEQ, jump _stages_loop (NP);;
```

ПРИЛОЖЕНИЕ Е

Фрагмент листинга программы, реализующей алгоритм без использования бит-реверсивной перестановки на сигнальном процессоре ADSP-TS-201

```
_combine_stage:
fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];; // r16=Re(G(n+2)+conj(G(N/2-(n+2)))), r20=Re(G(n+2)-conj(G(N/2-(n+2))))
// next twiddles(n+2)
fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];; // r18=Re(G(n+3)+conj(G(N/2-(n+3)))), r22=Re(G(n+3)-conj(G(N/2-(n+3))))
// next twiddles(n+3)
fr13=r13+r17, fr31=r13-r17;; // r13=Im(F(n)), r31=Im(F(N/2-n))
fr15=r15+r19, fr29=r15-r19; l[j12+=2]=xr13:12;; // r15=Im(F(n+1)), r29=Im(F(N/2-(n+1))), store F(n+N/8)
fr24=r20*r9; fr21=r1+r7, fr17=r1-r7; q[j14+=-4]=yr31:28;;     // r24=s(n+2)*Re(G(n+2)-conj(G(N/2-(n+2))))
// r21=Im(G(n+2)+conj(G(N/2-(n+2)))), r17=Im(G(n+2)-conj(G(N/2-(n+2))))
// store F(N/2-n), F(N/2-(n+1))
fr26=r22*r11; fr23=r3+r5, fr19=r3-r5; xr3:0=DAB q[j2+=4];; // r26=s(n+3)*Re(G(n+3)-conj(G(N/2-(n+3))))
// r23=Im(G(n+3)+conj(G(N/2-(n+3)))), r19=Im(G(n+3)-conj(G(N/2-(n+3))))
// xr3:0=next G(n+4+N/8), G(n+5+N/8)
fr25=r21*r8; yr3:0=DAB q[j0+=4];; // r25=c(n+2)*Im(G(n+2)-conj(G(N/2-(n+2))))
// yr3:0=next G(n+4), G(n+5)
fr27=r23*r10; q[j13+=-4]=xr31:28;;// r27=c(n+3)*Im(G(n+3)-conj(G(N/2-(n+3))))
// store F(N/2-(n+N/8)), F(N/2-(n+1+N/8))
fr24=r24+r25; fr25=r21*r9; l[j10+=2]=yr13:12;; // r24=Re(-i*exp(2*pi*i*x)(G(n+2)-conj(G(N/2-(n+2)))))
// r25=s(n+2)*Im(G(n+2)-conj(G(N/2-(n+2)))), store F(n)
fr26=r26+r27; fr27=r23*r11; l[j10+=2]=yr15:14;; // r26=Re(-i*exp(2*pi*i*x)(G(n+3)-conj(G(N/2-(n+3)))))
// r27=s(n+3)*Im(G(n+3)-conj(G(N/2-(n+3)))), store F(n+1)
fr13=r20*r8; fr12=r16+r24, fr30=r16-r24; l[j12+=2]=xr15:14;;// r13=cos(n+2)*Re(G(n+2)-conj(G(N/2-(n+2))))
// r12=Re(F(n+2)), r30=Re(F(N/2-(n+2))), store F(n+1+N/8)
fr15=r22*r10; fr14=r18+r26, fr28=r18-r26; xr7:4=q[j3+=-4];;  // r15=cos(n+3)*Re(G(n+3)-conj(G(N/2-(n+3))))
// r14=Re(F(n+3)), r28=Re(F(N/2-(n+3)))
// xr7:4=next G(N/2-(n+4+N/8)), G(N/2-(n+5+N/8))
fr13=r25-r13; xr9:8=l[k12+=k9]; yr7:4=q[j1+=-4];;  // r13=Im(-i*exp(2*pi*i*x)(G(n+2)-conj(G(N/2-(n+2)))))
// next twiddles(n+4+N/8)
// yr7:4=next G(N/2-(n+4)), G(N/2-(n+5))
.align_code 4;
        if NLC0E, jump _combine_stage(P); fr15=r27-r15; xr11:10=l[k12+=k9];;// r15=Im(-i*exp(2*pi*i*x)(G(n+3)-
conj(G(N/2-(n+3)))))
// next twiddles(n+5+N/8)
fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];; // r16=Re(G(n+4)+conj(G(N/2-(n+4)))), r20=Re(G(n+4)-conj(G(N/2-(n+4))))
// next twiddles(n+4)
fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];; // r18=Re(G(n+5)+conj(G(N/2-(n+5)))), r22=Re(G(n+5)-conj(G(N/2-(n+5))))
// next twiddles(n+5)
fr13=r13+r17, fr31=r13-r17;; // r13=Im(F(n+2)), r31=Im(F(N/2-(n+2)))
fr15=r15+r19, fr29=r15-r19;; // r15=Im(F(n+3)), r29=Im(F(N/2-(n+3)))
fr24=r20*r9; fr21=r1+r7, fr17=r1-r7; yr1:0=l[j31+j7];; // r24=s(n+4)*Re(G(n+4)-conj(G(N/2-(n+4))))
// r21=Im(G(n+4)+conj(G(N/2-(n+4)))), r17=Im(G(n+4)-conj(G(N/2-(n+4))))
// yr0=Re(G(0)), yr1=Im(G(0))
fr26=r22*r11; fr23=r3+r5, fr19=r3-r5;; // r26=s(n+5)*Re(G(n+5)-conj(G(N/2-(n+5))))
// r23=Im(G(n+5)+conj(G(N/2-(n+5)))), r19=Im(G(n+5)-conj(G(N/2-(n+5))))
fr25=r21*r8;;                                   // r25=cos(x)*Im(G(n)-conj(G(N/2-n)))
fr27=r23*r10; l[j12+=2]=xr13:12;;          // r27=cos(x)*Im(G(n)-conj(G(N/2-n)))
// store F(n+2+N/8)
```

```
yfr0=r1+r0; yr1=lshift r1 by -32; q[j14+=-4]=yr31:28;;// yr0=Re(G(0))+Im(G(0)), yr1=0=Im(F(0))
    // store F(N/2-(n+2)), F(N/2-(n+3))
    yfr0=r0+r0; q[j13+=-4]=xr31:28;;                                        // yr0=Re(F(0))
    // store F(N/2-(n+2+N/8)), F(N/2-(n+3+N/8))
    fr24=r24+r25; fr25=r21*r9; l[j10+=2]=yr13:12;; // r24=Re(-i*exp(2*pi*i*x)(G(n+4)-conj(G(N/2-(n+4)))))
                                                   // r25=s(n+4)*Im(G(n+4)-conj(G(N/2-(n+4)))), store F(n+2)
    fr26=r26+r27; fr27=r23*r11; l[j10+=2]=yr15:14;;// r26=Re(-i*exp(2*pi*i*x)(G(n+5)-conj(G(N/2-(n+5)))))
                                                   // r27=s(n+5)*Im(G(n+5)-conj(G(N/2-(n+5)))), store F(n+3)
    fr13=r20*r8; fr12=r16+r24, fr30=r16-r24; l[j12+=2]=xr15:14;;        // r13=c(n+4)*Re(G(n+4)-conj(G(N/2-(n+4))))
                                                   // r12=Re(F(n+4)), r30=Re(F(N/2-(n+4))), store F(n+3+N/8)
    fr15=r22*r10; fr14=r18+r26, fr28=r18-r26; l[j31+j7]=yr1:0;;                // r15=c(n+5)*Re(G(n+5)-conj(G(N/2-(n+5))))
                                   // r14=Re(F(n+5)), r28=Re(F(N/2-(n+5)))
                        // store F(0)
    fr13=r25-r13; l[j7+j17]=k11:10;;                // r13=Im(-i*exp(2*pi*i*x)(G(n+4)-conj(G(N/2-(n+4)))))

    // store F(N/4)
    fr15=r27-r15;                                  // r15=Im(-i*exp(2*pi*i*x)(G(n+5)-conj(G(N/2-(n+5)))))
    fr13=r13+r17, fr31=r13-r17;;                   // r13=Im(F(n+4)), r31=Im(F(N/2-(n+4)))
    fr15=r15+r19, fr29=r15-r19; l[j12+=2]=xr13:12;;  // r15=Im(F(n+5)), r29=Im(F(N/2-(n+5))), store F(n+4+N/8)
    q[j14+=-4]=yr31:28;;
    // store F(N/2-(n+4)), F(N/2-(n+5))
    q[j13+=-4]=xr31:28;;
    // store F(N/2-(n+4+N/8)), F(N/2-(n+5+N/8))
    l[j10+=2]=yr13:12;;
    // store F(n+4)
    l[j10+=2]=yr15:14;;
    // store F(n+5)
    l[j12+=2]=xr15:14;;
    // store F(n+5+N/8)
```