В.Ф. Гузик, В.Е. Золотовский,
Д.А. Беспалов, Е.В. Ляпунцова

Проектирование проблемно-ориентированных вычислительных систем

МИНИСТЕРТВО ОБРАЗОВАНИЙ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Южный федеральный университет»

В.Ф. Гузик, В.Е. Золотовский,
Д.А. Беспалов, Е.В. Ляпунцова

Проектирование проблемно-ориентированных вычислительных систем

Учебное пособие

Таганрог 2012 г.

УДК 004.3: 004.4

Рецензент:

Доктор технических наук, профессор, кафедры информатики ГОУВПО «Таганрогского государственного педагогического института имени А.П. Чехова» **Витиска Н.И.**

Гузик В.Ф., Золотовский В.Е., Беспалов Д.А., Ляпунцова Е.В.
Проектирование проблемно-ориентированных вычислительных систем. Учебное пособие. − Таганрог: Изд-во ТТИ ЮФУ, 2012. − 174с.

Данное учебное пособие представляет собой сборник руководств и заданий к выполнению лабораторных работ по курсу «Проектирование проблемно-ориентированных вычислительных систем». Целью является получение студентами навыков проектирования и реализации современных систем цифровой обработки сигналов на базе систем-на-кристалле (ПЛИС) и практическая реализация цифровых линейных фильтров, быстрых вычислительных алгоритмов Фурье-преобразований цифровых сигналов.
Учебное пособие рекомендовано студентам специальности 230101 всех форм обучения.

Табл. 2. Ил. 67. Библиограф. 16 назв.

© ТТИ ЮФУ, 2012
© Гузик В.Ф.,
Золотовский В.Е.,
Беспалов Д.А.,
Ляпунцова Е.В., 2012

СОДЕРЖАНИЕ

1. ЦИФРОВЫЕ СИГНАЛЫ И СИСТЕМЫ

Прежде чем приступить к рассмотрению способов анализа и обработки сигналов, следует выделить некоторые классы самих сигналов и определить их некоторые свойства, которые будут часто встречаться в дальнейшем. Это важно по многим причинам. Во-первых, сама природа сигнала и ее определение является частью анализа. Во-вторых принадлежность сигнала к тому или иному классу определяет структуру и назначение проблемно-ориентированной системы, проектируемой для его обработки. В-третьих, для анализа и обработки различных сигналов требуются разные подходы и разные аппаратно-программные комплексы.

1.1. Основные характеристики цифровых сигналов

С математической точки зрения, сигнал является *функцией*. Чаще всего рассматривается зависимость от времени, хотя это не является обязательным, так как для решения некоторых задач удобнее использовать другие функциональные пространства и другие переменные (частота, координаты, масштаб и т.п.).

Физическая природа сигналов также может быть весьма различной. В электронике, например, это напряжения, токи или сопротивление. Возможны и другие физические величины.

В данном учебном пособии, если это не оговаривается отдельно, мы будем подразумевать базовое представление сигналов как зависимость напряжения от времени.

В зависимости от того, известен ли нам сигнал заранее точно будем разделять *детерминированные* и *случайные* сигналы. Детерминированный сигнал, как следует из названия, известен нам заранее и его значение можно определить в любой момент времени точно. Случайный сигнал представляет собой случайную величину в любой момент времени, принимающую конкретные значения с некоторой вероятностью.

Специфические виды сигналов, в том числе и случайные, в данном пособии рассматриваться не будут, а основные методы анализа сигналов будут обсуждаться применительно к детерминированным сигналам.

Детерминированные сигналы являются сигналами с *интегрируемым квадратом*, т.е. *с ограниченной энергией*. Для таких сигналов $s(t)$ выполняется отношение:

$$\int_{-\infty}^{\infty} s^2(t)dt < \infty .$$

Многие важные соотношения теории цифровой обработки сигналов получены именно в предположении о конечности энергии анализируемых сигналов. Если это условие не выполняется, то приходится применять другие подходы, типа корреляционных функций для сигналов с конечной и бесконечной энергией или аппарата обобщенных функций [5].

Следующим важным признаком классификации сигналов является их *периодичность*. Здесь понимается повторение значений сигнала во времени через определенные промежутки (периоды T). Для таких сигналов выполняется соотношение

$$s(t + nT) = s(t) \text{ для любого } t,$$

где n – любое целое число.

Значения периода в этом случае будут кратными: *2T, 3T, …, nT.*

В некоторых случаях удобнее использовать понятие *частоты повторения сигнала f=1/T* или *круговой частоты* $\omega = 2\pi f$, измеряемой в радианах в секунду.

Любой периодический сигнал за исключением нулевого имеет бесконечную энергию, поэтому в реальных ситуациях используются *финитные* сигналы, т.е. существующие на конечном интервале времени.

Такие сигналы имеют конечную энергию, равны нулю вне границ интервала существования, при условии, что они не имеют разрывов второго рода с уходящими в бесконечность ветвями.

Очень важную роль в теории цифровой обработки сигналов играют *гармонические колебания*, также представляющие класс сигналов. В общем виде, гармонический сигнал имеет следующий вид:

$$s(t) = A\cos(\varpi t + \varphi).$$

В данном случае гармонический сигнал полностью определяется тремя числовыми параметрами: *амплитуда A, фаза* φ и *частота* ϖ.

Гармонический сигнал часто используется для тестирования систем обработки сигналов, для анализа характеристик цепей, а также для проверки математического аппарата.

Нередко гармонические сигналы определяют как *многокомпонентные*, т.е. составляемые из нескольких гармоник (суммы синусоид или косинусоид), как показано на Рис. 1.1.

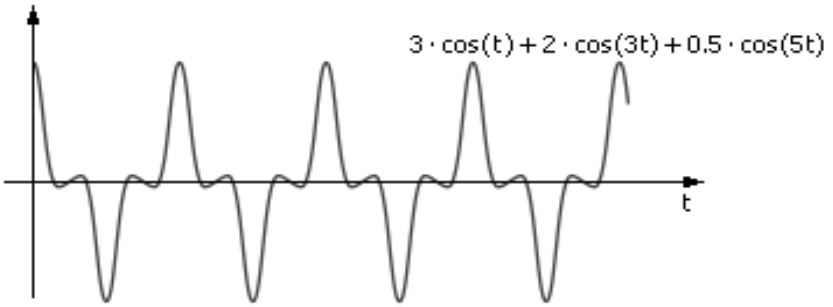


Рис. 1.1. Гармонический сигнал

Наряду с ним применяется *дельта-функция* $\delta(t)$ или *функция Дирака*, представляющая собой бесконечно узкий импульс с бесконечной амплитудой, расположенный в нулевом значении аргумента функции (например, в начале координат) (Рис. 1.2). Площадь функции равна единице.

$$\delta(t) = \begin{cases} 0, & t \neq 0, \\ \infty, & t = 0, \end{cases} \qquad \int_{-\infty}^{\infty} \delta(t)dt = 1.$$

Естественно, что дельта-функцию невозможно реализовать физически, однако в теории она достаточно часто используется.

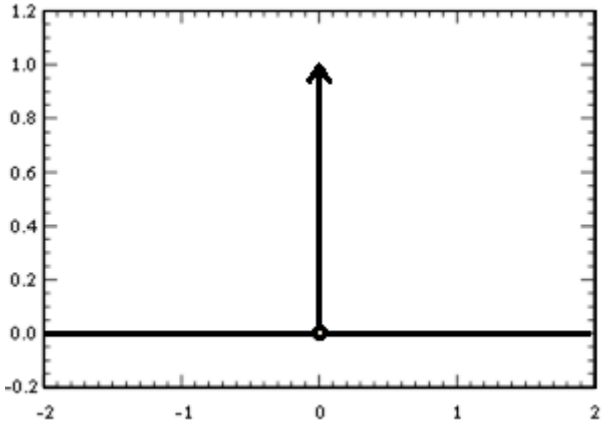


Рис. 1.2. Дельта-функция Дирака

Исходя из того, что интеграл от дельта-функции дает безразмерную единицу, можно утверждать, что размерность самой дельта-функции обратна размерности ее аргумента. Например, если дельта-функция времени имеет размерность 1/с, то она имеет размерность частоты.

Аналогичной по сфере применения функцией является *функция единичного скачка* $\sigma(t)$ или *функция Хевисайда*, она же *функция включения*:

$$\sigma(t) = \begin{cases} 0, & t < 0, \\ \frac{1}{2}, & t = 0, \\ 1, & t > 0. \end{cases}$$

При всех отрицательных значениях аргумента она равна нулю, а при всех положительных – равна единице. При нулевом значении аргумента, функцию принимают равной ½ либо считают неопределенной (Рис. 1.3).

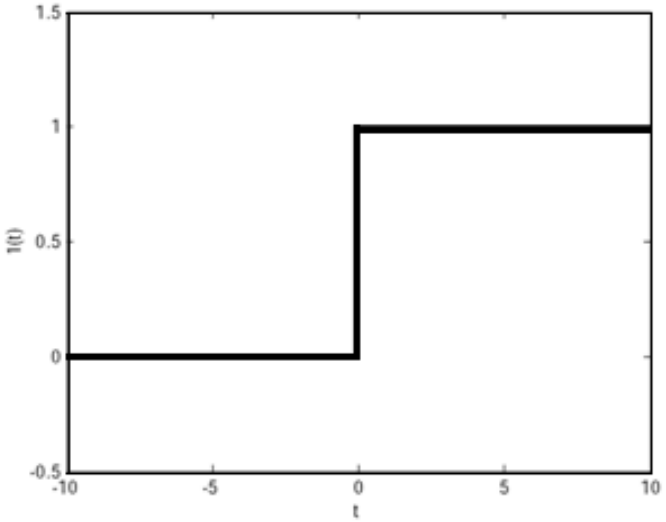


Рис. 1.3. Единичный скачок

Функцию единичного скачка удобно использовать при создании аналити-ческого аппарата для сигналов конечной длительности.

Далее нам необходимо рассмотреть некоторые свойства сигналов в виде их количественных (числовых) параметров. На практике наиболее часто применяют понятия *энергии* и *мощности* сигналов:

- энергия сигнала: $E = \int_0^T s^2(t)dt$;
- мгновенная мощность сигнала: $p(t) = s^2(t)$;
- средняя мощность сигнала: $\mathrm{P_{ср}} = \frac{1}{T}\int_0^T s^2(t)dt$.

Также дополнительно определяют:

- динамический диапазон, как отношение наибольшей мгновенной мощности сигнала к наименьшей $\mathrm{D} = 10\lg \mathrm{P_{max}} / \mathrm{P_{min}}$;
- ширину спектра канала F, как полосу частот, в пределах которой СОСРЕДОТОЧЕНА основная энергия сигнала;
- базу сигнала, как произведение длительности сигнала на ширину его спектра *B=TF*;
- отношение сигнал/шум, как отношение мощности полезного сигнала к мощности шума;
- объем передаваемой информации, выраженный в пропускной способности канала связи, необходимой для передачи сигнала и определяемый как произведение ширины спектра сигнала на его длительности и динамический диапазон: *V=FTD.*

Энергия сигнала может быть конечной или бесконечной. Любой сигнал конечной длительности, не содержащий дельта-функции или разрывов второго рода, имеет конечную энергию. Любой бесконечный периодический сигнал имеет, напротив, бесконечную энергию.

Если энергия сигнала бесконечна, то определяют его среднюю мощность на всей временной оси путем предельного перехода с устремление интервала усреднения в бесконечность:

$$\mathrm{P_{ср}} = \lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s^2(t)dt\,.$$

Квадратурный корень из средней мощности дает *среднеквадратичное* (*действующее*) значение сигнала [5]:

$$\sigma_s = \sqrt{\mathrm{P_{ср}}} = \sqrt{\lim_{T\to\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s^2(t)dt\,}\,.$$

Рассмотрим далее примеры типовых цифровых сигналов, которые часто используются для проверки и исследования методов ЦОС.

1.2. Примеры цифровых сигналов

1.2.1. Последовательность прямоугольных импульсов

Одним из простейших примеров цифровых сигналов является последовательность прямоугольных импульсов. Наиболее часто встречающаяся реализация – потенциальный код для передачи данных в цифровых системах и вычислительных машинах.

В данном случае, сигнал формируется как серия значений амплитуды A с длительностью τ и периодом повторения T. Начало отсчета времени полагаем в середине импульса, как показано на рис. 1.4.

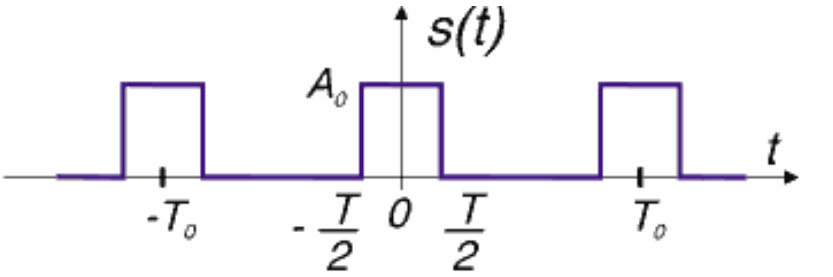


Рис. 1.4. Последовательность прямоугольных импульсов

1.2.2. Меандр

Меандр является частным случаем последовательности прямоугольных импульсов со скважностью 2. То есть длительность импульсов равна длительности промежутков между ними.

Пример такого сигнала показан на рис. 1.5.

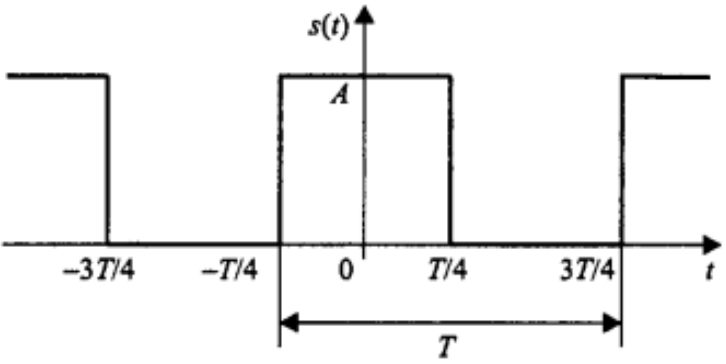


Рис. 1.5. Меандр

1.2.3. Пилообразный сигнал

Пилообразный сигнал – это линейная функция, которая в пределах периода описывается следующим образом:

$$s(t) = \frac{2A}{T}(t - kT), \quad \left(k - \frac{1}{2}\right)T < t \leq \left(k + \frac{1}{2}\right)T.$$

Пример пилообразного сигнала приведен на рис. 1.6.

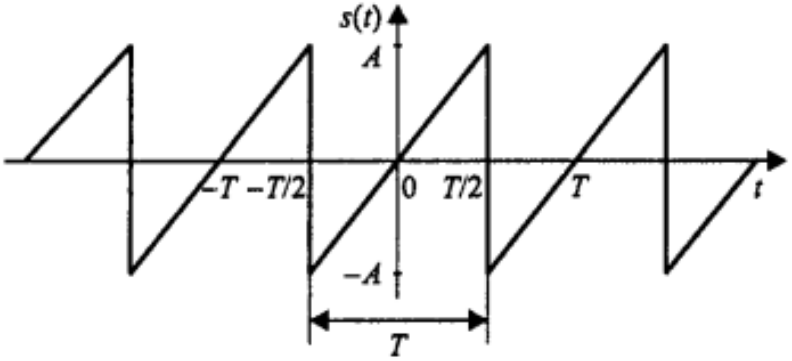


Рис. 1.6. Пилообразный сигнал

Как и прямоугольные импульсы, пилообразный сигнал характеризуется амплитудой импульсов A и частотой следования (периодом). Самое известное применение – это генераторы разверток

обычных телевизоров и осциллографов с применением кинескопов (электровакуумных трубок).

1.2.4. Последовательность треугольных импульсов

Периодическая последовательность треугольных импульсов похожа на пилообразный сигнал, однако имеет симметричную форму, как показано на рис. 1.7.

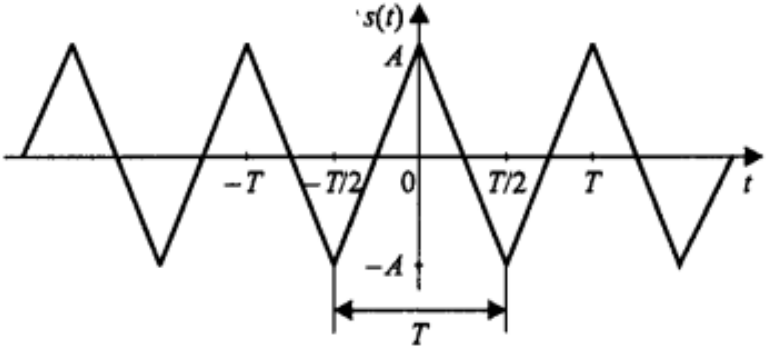


Рис. 1.7. Последовательность треугольных импульсов

Последовательность треугольных импульсов имеет следующее аналити-ческое выражение:

$$s(t) = A\left(1 - 4\frac{|t - kT|}{T}\right), \quad \left(k - \frac{1}{2}\right)T \le t < \left(k + \frac{1}{2}\right)T.$$

Рассмотрим далее принципы представления сигналов в цифровой области.

1.3. Квантование и дискретизация сигналов

Исходный сигнал, являющийся выражением некоторой физической величины, априори имеет аналоговую природу, т.е. определяется как *непрерывная* функция по времени. Такая функция известна во все моменты времени в интервале наблюдения.

Для обработки сигналов в вычислительных машинах и проблемно-ориентированных вычислительных системах используются последовательности чисел, представляющие отдельные отсчеты сигнала и называющиеся *дискретным рядом*. Такие данные не могут

полностью соответствовать исходному аналоговому сигналу, однако являются его достаточным приближением для цифровых вычислительных машин.

Отсчеты цифрового сигнала берутся из аналогового сигнала через определенные промежутки времени, имеющие название *период дискретизации* T (а также шаг или интервал дискретизации). Существует обратная величина периода дискретизации – *частота дискретизации* $f_д = \frac{1}{T_д}$. Ей также соответствует производная величина *круговая частота* $\varpi = \frac{2\pi}{T}$.

Пример дискретизации гармонического сигнала приведен на рис. 1.8.

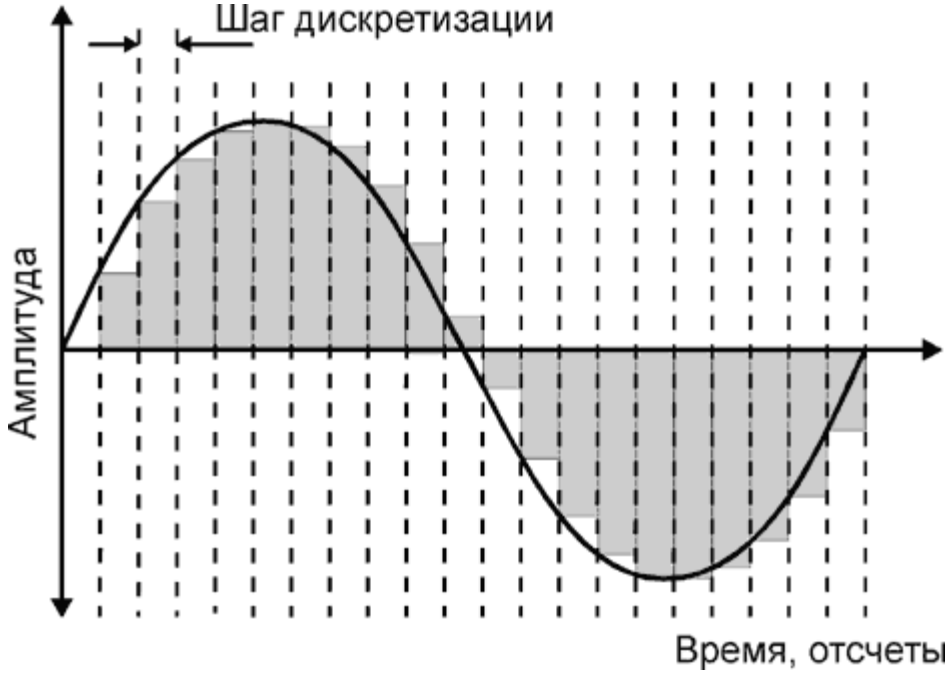


Рис. 1.8. Пример процедуры дискретизации сигнала

Естественно, что дискретное представление информации приводит к потерям данных по сравнению с аналоговым сигналом. Однако существует класс аналоговых сигналов, которые могут быть

точно восстановлены по своим дискретным значениям, если соблюдаются условия так называемой *Теоремы Котельникова*.

Потери информации при дискретизации происходят не только по тому, что информация берется в отдельные моменты времени, отстоящие друг от друга на некотором расстоянии. Второй причиной потерь является то, что данные о значениях обрабатываемого сигнала представляются с конечной точностью в виде переменных с фиксированной разрядностью. Это неизбежно приводит к *округлению* значений, а следовательно и к потерям.

Процесс преобразования значений физической величины в отдельных отсчетах в числа называется *квантованием по уровню*, а возникающие при этом ошибки округления называются *шумом квантования*.

Пример квантования гармонического сигнала приведен на рис. 1.9.

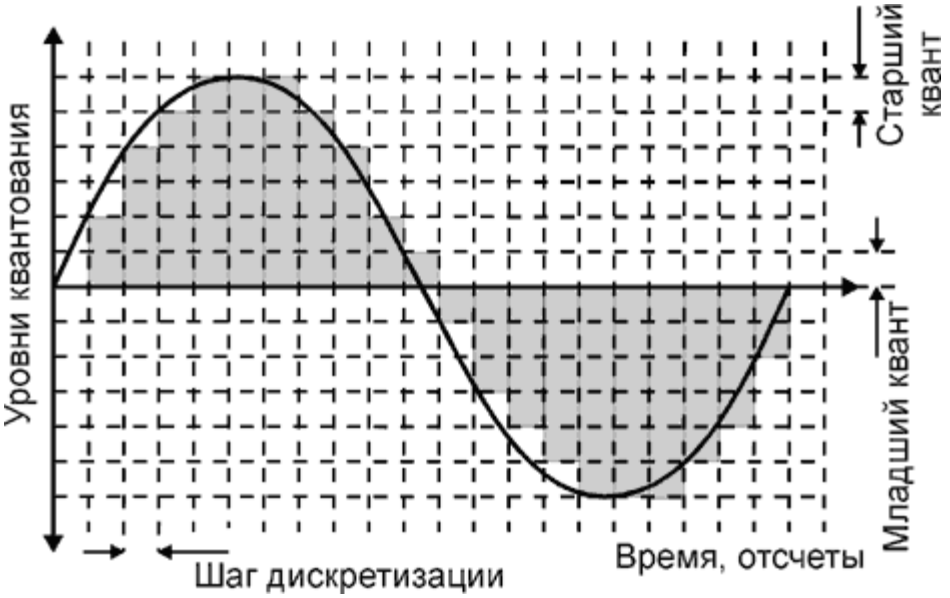


Рис. 1.9. Пример процедуры квантования сигнала

Таким образом, цифровой сигнал является сигналом, *дискретным по времени и квантованным по уровню*.

Вычислительные устройства, обрабатывающие сигналы, оперируют только цифровыми сигналами. Однако у них могут быть аналоговые источники. В этом случае входной аналоговый сигнал преобразуется в свой цифровой эквивалент при помощи *аналого-цифрового преобразования (АЦП)*.

Аналого-цифровой преобразование имеет и обратную операцию – *цифро-аналоговое преобразование (ЦАП)*. Данная операция восстанавливает непрерыв-ную форму сигнала (изменение физической величины) по цифровым данным.

В этом случае, обобщенная структура полной системы цифровой обработки сигналов приведена на рис. 1.10. На вход гибридной системы поступает аналоговый сигнал $s_{\text{вх}}(t)$. Далее в аналого-цифровом преобразователе проводится его дискретизация по времени и квантование по уровню. Выходным сигналом АЦП является последовательность чисел, поступающая на устройство обработки сигналов, например на сигнальный цифровой процессор (ЦП), выполняющий заложенный алгоритм обработки данных. Результатом его работы является последовательность чисел, представляющая собой отсчеты выходного сигнала. Результирующий аналоговый сигнал $s_{\text{вых}}(t)$ восстанавливается при помощи ЦАП.

Результирующий аналоговый сигнал имеет ступенчатую форму вследствие дискретности своего источника. Поэтому он может быть преобразован в плавно меняющийся при помощи сглаживающего фильтра Ф.

Гармонический сигнал может быть представлен дискретными данными, только если его частота не превышает частоты дискретизации, т.е. частоты, с которой производится выборка значений сигнала.

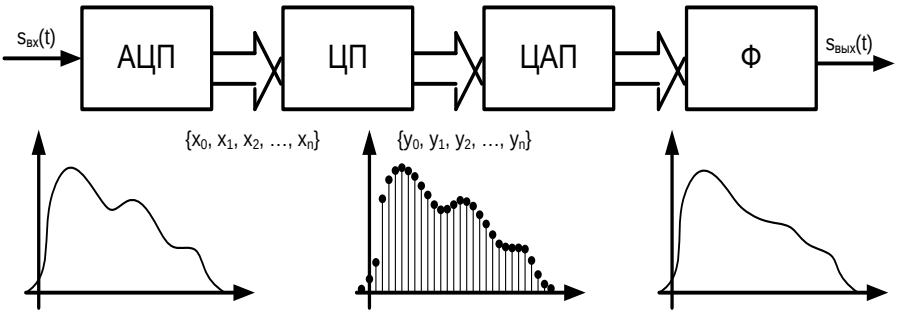


Рис. 1.10. Структура системы цифровой обработки сигналов

Эта частота носит название *частоты Найквеста* и равна

$$f_N = \frac{f_д}{2} = \frac{1}{2T},$$
$$\varpi_N = \frac{\varpi_д}{2} = \frac{\pi}{T}.$$

Между частотой Найквеста и частотой дискретизации возможно три типа соотношений:

1. Если частота гармонического сигнала меньше частоты Найквеста, то дискретные данные позволяют корректно восстановить исходный аналоговый сигнал.
2. Если частота Найквеста соразмерна по значению с частотой Найквеста, то дискретные отсчеты позволяют восстановить исходный аналоговый сигнал с той же самой частотой, однако в этом случае искаженными могут отказаться амплитуда сигнала и его фаза. В самом худшем случае, все дискретные отсчеты отдельных гармоник могут оказаться равными нулю из за смещения фазы.
3. Если частоты некоторых гармоник сигнала больше частоты Найквеста, то при восстановлении изменится сама частота аналогового сигнала. Этот эффект носит название: *появление ложных частот*.

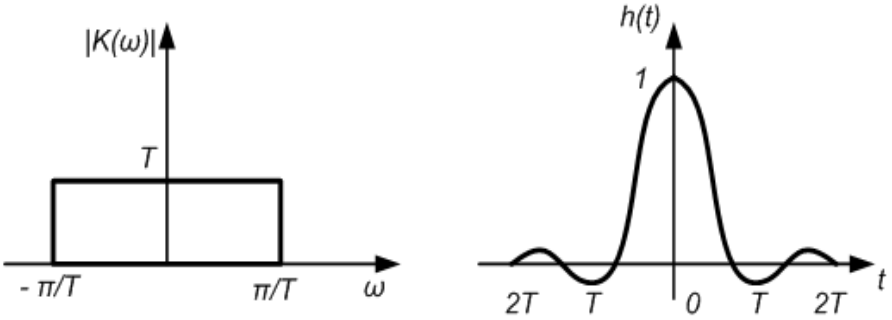
Спектр дискретизированного сигнала представляет собой сумму сдвинутых копий спектра исходного сигнала. При этом шаг сдвига равен частоте дискретизации ϖ_N.

Если в спектра аналогового сигнал анне содержится составляющих с частотами, превышающими частоту Найквеста (т.е. частоты только меньше $\varpi_N/2$), то сдвинутые копии спектра не будут

перекрываться и можно выделить один рабочий спектр для гармонического анализа сигнала.

Выделение основного спектра, сосредоточенного в окрестностях нулевой частоты, возможно при использовании идеального фильтра низких частот (ФНЧ) с прямоугольной амплитудно-частотной характеристикой (АЧХ). Восстановление точной копии исходного сигнала может быть выполнено именно с помощью этого основного спектра.

Пример идеального фильтра для восстановления аналогового сигнала приведен на рис. 1.11.



а) б)

Рис. 1.11. Амплитудно-частотная (а) и импульсная характеристики (б) идеального восстанавливающего фильтра

Коэффициент передачи в полосе пропускания равен T. Импульсная характеристика данного фильтра равна

$$h(t) = \frac{\sin\left(\pi \frac{t}{T}\right)}{\pi \frac{t}{T}} .$$

Дискретизированный сигнал фактически представляет собой сумму дельта-функций. При этом прохождение такого сигнала через восстанавливающий ФНЧ приводит к порождению на выходе соответствующим образом сдвинутую и масштабированную копию импульсной характеристики фильтра. Выходной сигнал с бесконечной точностью аппроксимирует исходный аналоговый сигнал и представляет собой сумму сдвинутых и умноженных на отсчеты сигнала копий импульсных характеристик идеального ФНЧ:

$$s(t) = \sum_{k=-\infty}^{\infty} s(kT) \frac{\sin\left(\pi \frac{t-kT}{T}\right)}{\pi \frac{t-kT}{T}} .$$

Резюмируя вышесказанное, можно сформулировать *теорему Котельникова* следующим образом: любой сигнал $s(t)$, спектр которого не содержит составляющих с частотами выше некоторого значения $\varpi_Д = 2\pi f_Д$ может быть без потерь информации представлен своими дискретный отсчетами $\{s(kT)\}$, взятыми с интервалом T, удовлетворяющим следующему неравенству: $T \leq \frac{1}{2f_Д} = \frac{\pi}{\varpi_Д}$.

1.4. Z-преобразование

В современной теории цифровой обработки сигналов часто используются методы анализа дискретных последовательностей, построенные на базе *z-преобразования*.

Смысл этого преобразования заключается в том, что последовательности чисел $\{x(k)\}$ ставятся в соответствие функции комплексной переменной z, определяемой следующим образом:

$$X(z) = \sum_{k=-\infty}^{\infty} x(k) z^{-k} .$$

Z-преобразование играет для дискретных сигналов такую же роль, что и преобразование Лапласа для сигналов.

Рассмотрим несколько примеров z-преобразования часто встречающихся на практике дискретных сигналов.

1.4.1. Единичная импульсная функция

Единичная импульсная функция является дискретным аналогом дельта-функции и представляет собой единичный отсчет с единичным значением:

$$x_0(k) = \begin{cases} 1, & k = 0, \\ 0, & k = 1. \end{cases}$$

Ее z-преобразование имеет простую форму

$$X_0(z) = \sum_{k=-\infty}^{\infty} x_0(k) z^{-k} = 1 * z^{-0} = 1.$$

1.4.2. Единичный скачок

Единичный скачок в дискретной области по смыслу соответствует своему прообразу в аналоговой форме

$$x(k) = \begin{cases} 1, & k < 0, \\ 0, & k \geq 0. \end{cases}$$

Используя определение z-преобразования можно получить

$$X(z) = \sum_{k=-\infty}^{\infty} x(k) z^{-k} = \sum_{k=0}^{\infty} 1 * z^{-k}.$$

Данный ряд является суммой бесконечной геометрической прогрессии с первым членом $1 * z^{-0} = 1$ и знаменателем z^{-1}. Как известно, такой ряд сходится при $\left|z^{-k}\right| < 1$, т.е. при $|z| > 1$ и его сумма равна

$$X(z) = \frac{1}{1 - z^{-1}}.$$

1.4.3. Дискретная экспоненциальная функция

Экспоненциальная зависимость физической величины от времени в дискретной области определяется следующим образом:

$$x(k) = \begin{cases} 1, & k < 0, \\ a^k, & k \geq 0. \end{cases}$$

Для вычисления z-преобразования нужно вычислить сумму следующего ряда:

$$X(z) = \sum_{k=-\infty}^{\infty} x(k) z^{-k} = \sum_{k=0}^{\infty} a^k * z^{-k} = \sum_{k=0}^{\infty} (a^{-1} z)^{-k}.$$

Этот ряд представляет собой сумму геометрической прогрессии. Первый член ряда 1, а знаменатель az^{-k}. Ряд сходится при $\left|az^{-k}\right| < 1$, т.е. при $|z| > |a|$. Его сумма равна

$$X(z) = \frac{1}{1 - az^{-1}}.$$

1.4.4. Дискретная синусоида с экспоненциальной амплитудой

Дискретная версия синусоидального сигнала с произвольной частотой, начальной фазой и экспоненциально меняющейся амплитудой представляется следующим образом:

$$x(k) = a^k \cos(\omega k + \phi). \quad (1.22)$$

Для вычисления z-преобразования можно представить косинус как полу-сумму двух комплексных компонент по формуле Эйлера:

$$X(z) = \frac{\cos(\varphi) - \mathrm{a}\cos(\varpi - \varphi)z^{-1}}{1 - 2a\cos(\varpi)z^{-1} + z^{-2}}. \quad (1.23)$$

Данный ряд сходится при $|z| > |a|$.

1.4.5. Свойства z-преобразования

Z-преобразование тесно связано с преобразование Фурье и с преобразованием Лапласа, следовательно они подобны друг другу по форме. Однако существует некоторая специфика, которая возникает вследствие дискретного характера рассматриваемых сигналов.

Линейность

Z-пребразование является линейной комбинацией отсчетов последовательности, представляющей исходный сигнал и подчиняется принципу суперпозиции:

если $\{x_1(k)\} \leftrightarrow X_1(z)$ и $\{x_2(k)\} \leftrightarrow X_2(z)$,

то $\{ax_1(k) + bx_2(k)\} \leftrightarrow aX_1(z) + bX_2(z)$.

Задержка

Если z-преобразование последовательности $\{x(k)\}$ равно $X(z)$, то z-преобразование последовательности, задержанной на k_0 тактов (т.е. $y(k) =$
$= x(k - k_0)$, будет равно

$$Y(z) = \sum_{k=-\infty}^{\infty} y(k)z^{-k} = \sum_{k=-\infty}^{\infty} x(k - k_0)z^{-k} =$$

$$= z^{-k_0} \sum_{k=-\infty}^{\infty} x(k - k_0)z^{-(k-k_0)} = z^{-k_0} \sum_{n=-\infty}^{\infty} x(n)z^{-n} = X(z)z^{-k_0}.$$

Таким образом, при задержке последовательности на k_0 тактов, необходимо умножить ее z-преобразование на z^{-k_0}. Последний множитель является *оператором задержки* последовательности на k_0 тактов.

Свертка

Свертка двух бесконечных дискретных последовательностей $\{x_1(k)\}$ и $\{x_2(k)\}$ определяется следующим образом:

$$y(k) = \sum_{n=-\infty}^{\infty} x_1(n)x_2(k-n).$$

Результат вычисления z-преобразования будет следующим:

$$Y(z) = \sum_{k=-\infty}^{\infty} y(k)z^{-k} = \sum_{k=-\infty}^{\infty}\left[\sum_{n=-\infty}^{\infty} x_1(n)x_2(k-n)z^{-k}\right] =$$

$$= \sum_{k=-\infty}^{\infty}\left[\sum_{n=-\infty}^{\infty} x_1(n)x_2(k-n)z^{-n}z^{-(k-n)}\right] =$$

$$= \sum_{n=-\infty}^{\infty}\left[x_1(n)z^{-n}\sum_{k=-\infty}^{\infty} x_2(k-n)z^{-(k-n)}\right] =$$

$$= X_2(z)\sum_{n=-\infty}^{\infty} x_1(n)z^{-n} = X_1(z)X_2(z).$$

Таким образом, линейная свертка двух дискретных последовательностей соответствует произведению их z-преобразований.

Обратное z-преобразование

Дискретную последовательность отсчетов сигнала можно восстановить из формы представления его в виде z-преобразования при помощи так называемого обратного z-преобразования. Формально эта процедура записывается следующим образом:

$$x(k) = \frac{1}{2\pi j}\oint X(z)z^{k-1}dz\,. \qquad (1.25)$$

Интеграл в данном случае берется по произвольному замкнутому контуру, расположенному в области сходимости функции $X(z)$ и охватывающему все ее полюсы.

В простейшем случае вычисление обратного z-преобразования сводится к разложению функции $X(z)$ на простые дроби.

Пусть

$$X(z) = \frac{1}{\frac{1}{2}z^{-2} - \frac{3}{2}z^{-1} + 1} =$$

$$= \frac{1}{(1 - z^{-1})\left(1 - \frac{1}{2}z^{-1}\right)} = \frac{2}{1 - z^{-1}} - \frac{1}{1 - \frac{1}{2}z^{-1}} .$$

Тогда, глядя на (1.18) и (1.21), можно заметить, что первое слагаемое соответствует скачку с амплитудой, равной 2, а второе – дискретной показательной функции $-2^{-k}, k \geq 0$. Тогда искомая последовательность имеет вид $x(k) = \begin{cases} 2 - 2^k, & k \geq 0, \\ 0, & k < 0. \end{cases}$

Z-преобразование тесно связано с преобразованием Лапласа и преобра-зованием Фурье. Рассмотрим этот факт подробнее.

Пусть нам задан некоторый дискретный сигнал, определенный для $k \geq 0$. Сопоставим этой последовательности чисел временный сигнал в виде набора дельта-функций:

$$s(t) = \sum_{k=0}^{\infty} x(k)\delta(t - kT) ,$$

где T - это интервал дискретизации.

Преобразование Лапласа для такого сигнала будет равно:

$$S(p) = \int_0^{\infty} s(t)e^{-pt}\,dt = \int_0^{\infty} \sum_{k=0}^{\infty} s(k)\,\delta(t - kT)e^{pt}\,dt =$$

$$= \sum_{k=0}^{\infty} x(k) \int_0^{\infty} \delta(t - kT)e^{-pt}\,dt.$$

Воспользовавшись схожестью дельта функции с линейным фильтром, можно получить:

$$S(p) = \sum_{л=0}^{\infty} x(k)e^{-pkT} .$$

Эта формула переходит в выражение (1.13) при замене $z = e^{pT}$.

Таким образом, можно однозначно определить связь:

между z-преобразованием и преобразованиями Фурье:

$$X(z) = S\left(\frac{1}{T}\ln z\right),$$

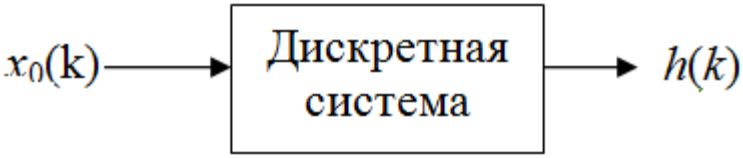
$$S(p) = X(e^{pT}),$$

между z-преобразованием и преобразованием Лапласа:

$$X(z) = \dot{S}\left(\frac{1}{jT}\ln z\right),$$
$$\dot{S}(p) = X\left(e^{j\omega T}\right).$$

1.5. Дискретные линейные системы

Под *дискретной линейной системой* будем понимать некоторую систему «черный ящик», внутри которой по некоторому закону будет обрабатываться (изменяться) входной дискретный сигнал.



$$y_0(k) = F\left(x_0(k)\right) = h(k)$$

Рис. 1.12. Дискретная линейная система

Чаще всего под дискретными линейными системами понимают *линейные фильтры* или *дискретные фильтры*. Суть их работы проста – они *пропускают* одни частоты и *задерживают* другие.

Однако линейные системы не ограничиваются только таким применением. Они могут выполнять и иные сложные задачи кроме выделения из сигнала определенных полос частот.

Рассмотрим сущность линейной обработки.

Любая линейная дискретная система – это произвольная система обработки дискретной информации, обладающая набором свойств:

1. *Аддитивность* в данном случае означает то, что выходная реакция системы на сумму сигналов равна сумме реакций на эти сигналы, поданные на вход по отдельности: $x(k) = \sum_{k=-\infty}^{\infty} x_0(k-m)x(m)$.
2. Реакцией на единичную импульсную функцию $x_0(k)$ является импульсная характеристика дискретной системы $h(k)$.
3. *Инвариантность во времени* определяет, что воздействию, задержанному на m отсчетов, соответствует реакция,

задержанная на такое же количество времени: $x_0(k-m) => h(k-m)$.

4. *Однородность* системы означает, что умноженному на константу $x(m)$ воздействию соответствует реакция, умноженная на ту же константу: $x_0(k-m)x(m) => h(k-m)x(m)$.
5. *Стационарность* в этом случае означает то, что задержка входного сигнала приводит лишь к такой же задержке сигнала на выходе системы без изменения его формы.

Любая линейная система, выполняющая функции фильтра обладает определенной частотной характеристикой. Кроме того, в них выходной сигнал $y(k)$ зависит не только от неизменяемых внутренних коэффициентов, используемых при обработки сигналов, но и от нескольких предыдущих значений входного сигнала или даже от состояния своего же собственного выхода с некоторым запозданием. То есть такая система обладает некоторой *памятью* и по разному пропускает на выход сигналы различных частот.

Дискретные линейные системы, как и аналоговые, имеют несколько способов описания. Благодаря наличию и свойствам z-преобразования, способы описания аналоговых и дискретных сигналов похожи друг на друга.

Рассмотрим их последовательно.

1.5.1. Импульсная характеристика линейной системы

В том случае, когда используется линейная система с постоянными параметрами, для анализа прохождения любого сигнала достаточно знать результат обработки элементарного импульса. Таким импульсом является дельта-функция, а точнее единичная импульсная функция $x_0(k)$.

В этом случае реакция на единичный импульс $x_0(k)$ называется *импульсной характеристикой* дискретной линейной системы и обозначается как $h(k)$.

Импульсная характеристика линейной системы является ее основным свойством, при помощи которого можно определить реакцию на любое внешнее воздействие.

Произвольный входной сигнал $\{x(k)\}$ может быть представлен линейной комбинацией единичных отсчетов с соответствующей поправкой:

$$x(k) = \sum_{m=-\infty}^{\infty} x(m)x_0(k-m).$$

Исходя из соображений линейности и стационарности анализируемых систем, входной сигнал должен представлять собой линейную комбинацию импульсных характеристик:

$$y(k) = \sum_{m=-\infty}^{\infty} x(m)h(k-m).$$

Последнее выражение (1.31) называют *дискретной сверткой* или *линейной сверткой*. Для реальных физически реализуемых систем верхний предел суммирования изменяется на *k*:

$$y(k) = \sum_{m=-\infty}^{k} x(m)h(k-m).$$

Такое ограничение сверху означает то, что линейная дискретная система при вычислении очередного значения выходного сигнала может ориентироваться только на свою предысторию и ничего не знает о будущем.

1.5.2. Передаточная функция линейной системы

Если провести над уравнением дискретной свертки (1.31) z-преобразование, то, согласно свойствам последнего, результатом будет являться произведение z-преобразований обоих компонент, то есть:

$$Y(z) = X(z) * H(z)\ .$$

Компонента дискретной свертки (1.29) $H(z)$ равна отношению z-преобразований входного и выходного сигналов и является z-преобразованием импульсной характеристики системы – т.е. *передаточной функцией системы*:

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{k=0}^{\infty} h(k)z^{-k}.$$

1.5.3. Частотная характеристика линейной системы

Для того чтобы получить комплексный коэффициент передачи (частотную характеристику) дискретной системы, можно воспользоваться формулой (1.29), описывающей связь z-преобразования с преобразованием Фурье:

$$\dot{K}(\varpi) = H\left(e^{j\varpi T}\right) = \sum_{k=\infty}^{\infty} h(k)e^{-j\varpi kT} \ .$$

Из (1.35) видно, что частотная характеристика дискретной системы является периодической функцией частоты с периодом, равным частоте дискретизации $\omega_{\text{д}} = 2\pi/T$ так же, как и спектр дискретизированного сигнала.

1.5.4. Дискретная свертка линейной системы

Линейное уравнение (1.32) называется дискретной сверткой системы.

Дискретная свертка позволяет описать взаимосвязь выхода дискретной системы с ее входом при помощи импульсной характеристики.

Эта формула может быть записана в следующем виде:

$$y(k) = h(k) * x(k).$$

Пределы суммирования в (1.32) меняются в соответствии с входными параметрами сигнала и свойствами системы. Для любой физически реализуемой дискретной линейной системы можно использовать одно из представлений дискретной свертки:

$$y(k) = \sum_{m=0}^{k} x(k)h(k-m) \ ,$$

$$y(k) = \sum_{m=0}^{k} h(m)x(k-m) \ .$$

Ограничение пределов тут означает, что линейная система использует только предыдущие значения отсчетов входного воздействия и не имеет информации о последующих.

1.5.5. Разностное уравнение линейной системы

Наряду с описанием дискретных линейных систем с помощью формулы свертки входного сигнала $x(k)$ с импульсной характеристикой дискретной системы $h(k)$ также используется описание соотношения вход/выход линейной системы с помощью *линейный разностных уравнений.*

Разностные уравнения имеют следующий вид:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) + \ \ldots + b_n x(k-n) - \\ -a_1 y(k-1) - a_2 y(k-2) - \ \ldots - a_m y(k-m) = \\ = \sum_{i=0}^{n} b_i x(k-i) - \sum_{j=1}^{m} a_j y(k-j)\,.$$

В (1.39) параметры b_i и a_j это вещественные коэффициенты (константы), являющиеся внутренними параметрами линейной системы, а $x(k)$ и $y(k)$ − это входное воздействие и выходная реакция соответственно.

Линейная система, полностью описываемая уравнением (1.39), является физически реализуемой, так как при нулевых начальных условиях ее реакция не может возникнуть раньше воздействия, потому что значение реакции $y(k)$ зависит от текущего и предыдущих значений воздействия и не зависит от последующих.

Заключение

В данном учебном пособии понятие дискретной линейной системы неразрывно связано с понятием цифрового фильтра и используется для его представления.

В следующей главе и соответствующей лабораторной работе предлагается рассмотреть свойства конкретной линейной системы (цифрового фильтра) и обработать при помощи нее некоторое входной воздействие (дискретный сигнал).

2. РАЗРАБОТКА И ИССЛЕДОВАНИЕ ЛИНЕЙНЫХ ФИЛЬТРОВ

2.1. Основные положения

Цифровой фильтр является линейной системой, оказывающей влияние на цифровой сигнал, пропускаемый через него [6].

Фильтры являются основой для большинства приложений обработки сигналов. Типичное назначение − это извлечение или вырезка области спектра входного сигнала или определенной частоты. Используемые для кондиционирования сигналов фильтры нередко называются частотно-селектирующими, поскольку обычно разрабатываются на основе требований к частотной характеристике.

В большинстве случаев цифровая фильтрация применяется при необходимости убрать некоторые компоненты из обрабатываемых данных, считаемые шумовыми.

Классификация цифровых фильтров (ЦФ) была предложена в докладе Д.А. Губанова и В.Б. Сташенко (см. Рис. 2.1). В ее основу положен функциональный признак (т.е. используемые алгоритмы цифровой фильтрации, а не схемотехнические решения), согласно которому ЦФ подразделяются на 4 группы: фильтры частотной селекции, оптимальные (квазиоптимальные), адаптивные и эвристические.

Наиболее изученными и опробованными на практике являются ЦФ частотной селекции. Они почти всегда представляют собой реализованные на новой элементной базе традиционные аналоговые фильтры частотной селекции. Развитие цифровых средств частотной селекции происходит в следующих направлениях:

1) создание пакетов прикладных программ для структурного синтеза, анализа качества фильтрации, обеспечения схемотехнической реализации и тестирования устройств;

2) совершенствование существующих методов оптимального проектирования многоступенчатых структур с целью их полной формализации и включения в состав ПО;

3) разработка новых подходов к проектированию ЦФ частотной селекции с улучшенными качественными показателями.

Новейшие технические реализации как традиционных, так и нетрадиционных задач цифровой обработки сигналов чаще всего

используют разные схемотехнические решения. Наибольшее внимание участников секции цифровой фильтрации привлекли алгоритмы многоскоростной обработки сложных сигналов, которые содержат как быстро, так и медленно меняющиеся составляющие.

Рис. 2.1. Классификация цифровых фильтров

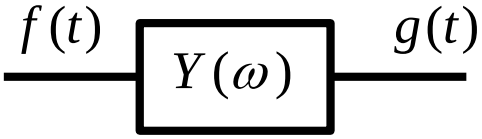
Такой алгоритм должен предусматривать предварительное разделение «быстрых» и «медленных» компонентов с понижением частоты дискретизации «медленных» составляющих и последующей их обработкой. Современные вычислительные средства позволяют решать в режиме реального времени и задачи многомерной фильтрации, существенно более сложные, чем цифровая фильтрация

одномерных сигналов, выполняемая с помощью сигнальных процессоров или многопроцессорных систем.

При использовании цифровых фильтров в задачах обработки сигналов ставится цель «удаления» одной или нескольких компонентов, называемых шумовыми, из исходного сигнала. Такое положение вещей вполне описывается следующим уравнением, где исходный сигнал до обработки $f(t)$ представляется в виде аддитивной комбинации полезной $x(t)$ и побочной $\eta(t)$ компонентов сигнала

$$f(t) = x(t) + \eta(t).$$

Задача фильтра - так обработать $f(t)$, чтобы получить по мере возможности истинный сигнал $x(t)$. В некоторых случаях операция сглаживания объединяется с операцией предсказания (экстраполяции), т.е. необходимо не только получить хорошее приближение к $x(t)$, но и к тому, что $x(t)$ будет представлять собой в будущем, скажем через τ секунд $x(t+\tau)$. Если $\tau < 0$, то ищется, каким был сигнал τ секунд назад. Схематично это можно представить в таком виде



Здесь $g(t)$ – приближение для $x(t)$.

Таким образом, задача сводится к нахождению вида обработки *Y(* ω*)*. Нахождение этой функции базируется на следующих трех предположениях.

Временные последовательности, представляющие сигнал $x(t)$ и шум $\eta(t)$, стационарны, т.е. статистические свойства шума и сигнала не изменяются во времени.

В качестве критерия ошибки приближения взято среднеквадратичное отклонение между действительным сигналом и откликом системы. Для нашего случая это означает:

$$\overline{\left[g(t) - x(t+\alpha)\right]^2} = \min,$$

где черта над операцией означает усреднение по значениям погрешности на интервале и по всем возможным функциям сигнала и шума, взвешенным соответственно вероятностям их появления (среднее по ансамблю).

Операция сглаживания и предсказания предполагается линейной операцией, т.е. искомое звено должно быть линейным физически осуществимым фильтром.

Любой линейный фильтр может быть описан несколькими способами. Наиболее распространенное описание – это описание в терминах комплексного коэффициента передачи $Y(\omega)$, в терминах его импульсной характеристики $h(k)$ и передаточной функции $H(z)$. Если чистая синусоида частоты ω и амплитуды E воздействует на вход фильтра, то откликом является синусоида частотой ω и амплитудой $|Y(\varpi)|E$. Фаза на выходе определяется фазой фильтра на этой частоте. Так как $Y(\omega)$ есть комплексный коэффициент передачи, то его удобно записывать в форме $Y(\varpi) = e^{A(\varpi)}e^{iB(\varpi)}$, где $A(\varpi) = \ln|Y(\varpi)|$ – усиление, а $B(\varpi)$ – угол $|Y(\varpi)|$, являющийся фазой. Коэффициент передачи $|Y(\varpi)|$ всегда может быть выбран таким, чтобы достичь требуемой амплитуды A.

Можно выделить несколько основных классов фильтров, используемых на практике.

2.1.1. Фильтр низких частот

Фильтр низких частот (ФНЧ) является одним из самых широко распространенных видов аналоговых или цифровых фильтров, эффективно обрабатывающий сигнал с частотным спектром, в котором частоты, расположенные ниже частоты среза, пропускаются, а частоты, расположенные выше этой частоты, подавляются. Степень подавления зависит от типа и характеристик фильтра.

Простейший фильтр высоких частот состоит из конденсатора и резистора, соединенных по схеме, представленной на рис. 2.2.

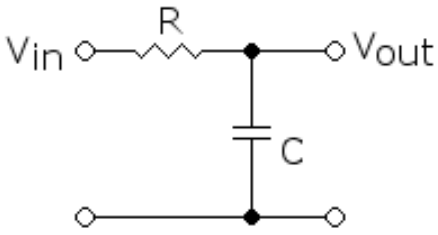


Рис. 2.2. Аналоговый фильтр низких частот

Идеальный фильтр низких частот полностью подавляет все частоты входного сигнала со значениями выше частоты среза и пропускают без изменения все гармоники сигнала, выше частоты среза. Для идеального ФНЧ не существует переходной зоны между частотами *полосы подавления* и *полосы пропускания*. Естественно, что такой фильтр на практике не может быть реализован. Он существует только теоретически и реализуется при помощи умножения входного сигнала на прямоугольную функцию в частотной области или же при свертке сигнала во временной области с sinc-функцией.

Sinc-функция обозначается как $sinc(x)$ и называется *кардинальным синусом*. Имеет два определения нормированной и ненормированной sinc-функции. Нормированная функция применяется в обработке сигналов и имеет вид

$$sinc(x) = \begin{cases} \sin\frac{(\pi x)}{\pi x}, & x \neq 0 \\ 1, & x = 0 \end{cases}.$$

Ненормированная sinc-функция в математике определяется как

$$sinc(x) = \begin{cases} \sin\frac{(x)}{x}, & x \neq 0 \\ 1, & x = 0 \end{cases}.$$

В обоих случаях ее значение в особой точке $x = 0$ явным образом задается равным 1.

Частотный и временной домен этой функции связаны через преобразование Фурье. Ее непрерывное преобразование Фурье как раз и образует для единичного интервала частот прямоугольную функцию $rect(f)$:

$$\int_{-\infty}^{\infty} sinc(t)e^{-2\pi ift}\,dt = rect(f).$$

Реальные фильтры могут лишь приближаться к идеальному варианту с той или иной степенью точности.

2.1.2. Фильтр высоких частот

Фильтр высоких частот или *фильтр верхних частот (ФВЧ)* − это также цифровой или аналоговый фильтр, пропускающий высокие частоты входного сигнала, подавляя, при этом частоты сигнала меньше, чем частота среза. Степень подавления также зависит от типа и характеристик фильтра.

Простейший фильтр высоких частот состоит из конденсатора и резистора, соединенных по схеме, представленной на рис. 2.3.

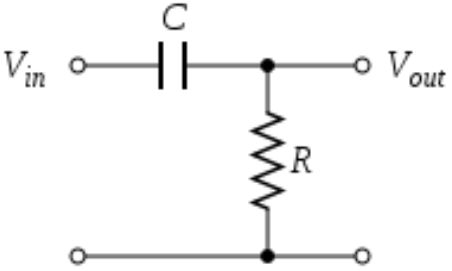


Рис. 2.3. Аналоговый фильтр высоких частот

Конденсатор пропускает только переменный ток, а выходное напряжение снимается с резистора. Произведение сопротивления на емкость является постоянной времени для такого фильтра, которая обратно пропорциональна частоте среза в герцах $f = 1/2\,\pi RC$.

2.1.3. Фазовый фильтр

Фазовый фильтр – это аналоговый или цифровой фильтр, пропускающий все частоты сигнала с равным усилением, однако меняющий фазу сигнала. Это происходит путем задержки пропускания по частотам. Обычно такой фильтр характеризуется основным параметром – частотой, на которой фазовый сдвиг равен 90^{o}.

2.1.4. Полосовой фильтр

Полосовой или *полосно-пропускающий фильтр* − это цифровой или аналоговый фильтр, который пропускает частоты, находящиеся в некотором диапазоне (полосе) частот.

Полосовой фильтр – это линейная система и может быть представлен в виде последовательности фильтра низких частот и фильтра высоких частот.

Аналоговый полосовой фильтр формируется конденсатором, катушкой и резистором, как показано на рис. 2.4.

Идеальные полосовые фильтры определяются двумя основными характеристиками:

1) нижняя частота среза ϖ_{C1};
2) верхняя частота среза ϖ_{C2}.

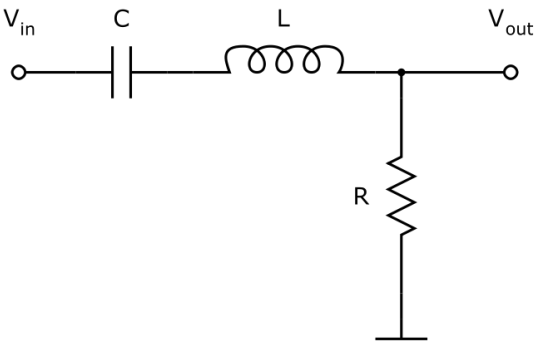


Рис. 2.4. Аналоговый полосовой фильтр

Кроме того, важными являются:

1) нижняя граница частоты пропускания ϖ_{P1};
2) верхняя граница частоты пропускания ϖ_{P2};
3) нижняя граница частоты задержания ϖ_{S1};
4) верхняя граница частоты задержания ϖ_{S2};
5) максимальное подавление в полосе пропускания;
6) минимальное подавление в полосе подавления.

2.1.5. Полосно-заграждающий фильтр

Полосно-заграждающий фильтр, также называемый *режекторным* – это цифровой или аналоговый фильтр, не пропускающий колебания некоторой определенной полосы частот и пропускающий колебания с частотами, не попадающими в эту полосу.

Эта полоса пропускания характеризуется шириной BW и расположена приблизительно вокруг некоторой центральной частоты ω_0. Для амплитудно-частотной характеристики существуют частоты ω_L и ω_U, представляющие собой нижнюю и верхнюю частоты среза.

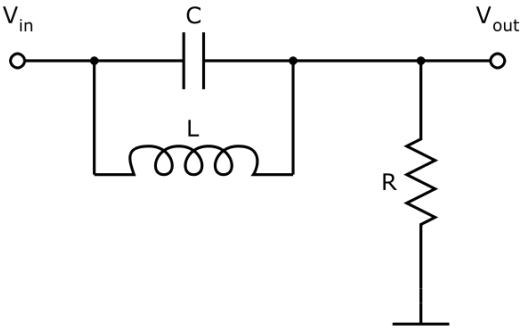


Рис. 2.5. Аналоговый режекторный фильтр

Заграждающий фильтр, предназначенный в основном для подавления одной определенной частоты, также называют *узкополосным заграждающим фильтром* или *фильтром-пробкой*.

2.2. Описание линейных фильтров

Для дискретных линейных фильтров справедливы такие же условия, что и для аналоговых фильтров, но записанные для дискретных сигналов [3,4]. Например, принцип суперпозиции формулируется в следующем виде. Если $y_{1,n} = x_{1,n}$ и $y_{2,n} = x_{2,n}$, то входной последовательности $x_n = a_1 x_{1,n} + a_2 x_{2,n}$ соответствует реакция $y_n = a_1 y_{1,n} + a_2 y_{2,n}$.

Дискретные фильтры в общем случае представляют собой аппроксимацию того или иного порядка соответствующего аналогового фильтра.

Аппроксимирующее уравнение или система уравнений может достаточно простыми преобразованиями сведена к разностному уравнению m-го порядка. Общая форма этого уравнения имеет вид

$$y_n = \sum_{l=0}^{r} x_{n-l} b_l - \sum_{j=1}^{m} y_{n-j} a_j ,$$

где *r, m* – постоянные числа, определяющие пределы фильтра, a_j и b_l – постоянные коэффициенты, а x_n и y_n – отсчеты входного и выходного сигнала.

Форма уравнения (2.2) подчеркивает итеративный характер вычислений, т.е. если известны *m* выходных значений *(yn-1, yn-2,..., yn-m,)* и *r* входных *(*X *n-1,* X *n-2,...,* X *n-r,)*, то можно вычислить новое

выходное значение. Используя Z – преобразование, можно получить общее решение для y_n как функции входной последовательности. Преобразуем (3.2) к виду

$$\sum_{l=0}^{m} a_l y_{n-l} = \sum_{j=1}^{r} b_j x_{n-j}, \quad n = 0,1,2,\ldots$$

и возьмем Z – преобразование от обеих частей равенства. В результате имеем

$$\sum_{l=0}^{m} a_l \sum_{n=0}^{\infty} y_{n-l} z^{-n} = \sum_{l=0}^{r} b_l \sum_{n-0}^{\infty} x_{n-l} z^{-n}.$$

Используя первую теорему смещения, получим

$$Y(z) \sum_{l=0}^{m} a_l z^{-l} = X(z) \sum_{l=0}^{r} b_l z^{-l},$$

или

$$Y(z) = X(z) H(z),$$

где

$$H(z) = \frac{\sum_{l=1}^{m} b_l z^{-l}}{1 + \sum_{l=1}^{m} a_l z^{-l}}.$$

Таким образом, изображение выходной функции $y(z)$ явно определяется как произведение Z - преобразования входной последовательности на функцию $H(z)$. Функция $H(z)$ является передаточной функции дискретного фильтра.

Передаточная функция $H(z)$ представляет собой рациональную дробь относительно переменной z^{-l}. Постоянные коэффициенты разностного уравнения являются постоянными коэффициентами передаточной функции. Выражение для выходного сигнала можно получить с помощью обратного $Z-$преобразования.

Если в качестве входной функции выбрать единичный импульс, т.е.

$$\mathrm{x_n} = \delta_\mathrm{n} \text{ для } n = 0,1,2,\ldots,$$

то его $Z-$изображение $X(z) = 1$. Откликом на такой сигнал будет обратное $Z-$преобразование функции $H(z)$. Последовательность $x_n = 1,0,0,\ldots$ в теории цифровых фильтров играет ту же роль, что и дельта импульс в теории аналоговых фильтров.

Передаточная функция определяет частотную избирательность фильтра. Действительно, пусть входной сигнал представляет собой комплексную функцию

$$x_n = X(nT) = e^{in\varpi T}.$$

Для некоторого входного сигнала выходной сигнал может быть записан в виде

$$y_n = Y(nT) = F\left(e^{i\varpi T}\right)e^{in\varpi T}.$$

Далее получим

$$F\left(e^{i\varpi T}\right) = \left(\sum_{l=0}^{r} b_l e^{il\varpi T}\right) - \sum_{l=0}^{m} a_l F\left(e^{i\varpi T}\right)e^{il\varpi T}.$$

Преобразуя, окончательно будем иметь

$$F\left(e^{i\varpi T}\right) = \frac{\sum_{l=0}^{r} b_l e^{-il\varpi T}}{1 + \sum_{l=0}^{m} a_l e^{-il\varpi T}} = H\left(e^{i\varpi T}\right).$$

Из выражения (2.10) можно определить и амплитудно-частотную и фазо-частотную характеристики.

Функция $H\left(e^{i\varpi T}\right)$ является комплексной функцией, и она может быть представлена в виде

$$H\left(e^{i\varpi T}\right) = H_R\left(e^{i\varpi T}\right) + iH_J\left(e^{i\varpi T}\right),$$

где $H_R\left(e^{i\varpi T}\right)$, $H_J\left(e^{i\varpi T}\right)$ – соответственно реальная и мнимая часть *H*.

Амплитудно-частотной характеристикой (АЧХ) выражения (2.11) есть $\left|H\left(e^{i\varpi T}\right)\right|$ или

$$A(\varpi T) = \left|H\left(e^{i\varpi T}\right)\right| = \sqrt{H_R^2(e^{i\varpi T}) + H_J^2(e^{i\varpi T})}.$$

Типовая АЧХ фильтра выглядит следующим образом:

$$A(\omega T) = \frac{\sqrt{\left(\sum_{k=0}^{p-1} b_k \cos \omega k T\right)^2 + \left(\sum_{k=0}^{p-1} b_k \sin \omega k T\right)^2}}{\sqrt{\left(1 + \sum_{m=1}^{n-1} a_m \cos \omega m T\right)^2 + \left(\sum_{m=1}^{n-1} a_m \sin \omega m T\right)^2}}.$$

Пример типовых характеристик фильтров показан на рис. 2.6.

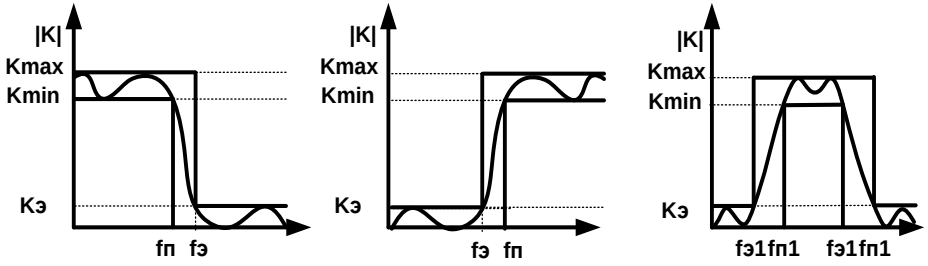


Рис. 2.6. Типовые характеристики фильтра низких частот (ФНЧ), фильтра верхних частот (ФВЧ) и полосового фильтра

Соответственно фазовой частотной характеристикой (ФЧХ) называется функция

$$\phi(\omega T) = arctg\left(\frac{H_R(e^{i\varpi T})}{H_J(e^{i\varpi T})}\right).$$

или

$$\phi(\omega T) = arctg\left(\frac{-\sum_{k=0}^{p-1} b_k sin\varpi kT}{\sum_{k=0}^{p-1} b_k cos\varpi kT}\right) - arctg\left(\frac{-\sum_{m=1}^{n-1} a_m sin\varpi mT}{1+\sum_{m=1}^{n-1} a_m cos\varpi mT}\right).$$

И амплитудная частотная и фазовая частотная характеристики являются периодическими функциями с периодом, связанным с частотой дискретизации $\varpi_{\text{d}} = \frac{2\pi}{\text{T}}$.

Для гармонической функции вида $x = \cos \omega T$ такой переход осуществляется следующим образом. Учитывая, что $\omega = 2\pi f$ и $T = \frac{1}{f_{\text{Д}}}$ и произведя замену соответствующих переменных, получим, что $x = \cos 2\pi \tilde{f}$, а $\tilde{f} = \frac{f}{f_{\text{Д}}}$.

Частота дискретизации выбирается так, чтобы было удовлетворено условие теоремы Котельникова, т.е. $f_{Д} \geq 2 f_{\max}$. Выбирая $f_{Д} = 2 f_{\max}$, получим

$$\tilde{f}_{\max} = \frac{f_{\max}}{f_{Д}} = 0{,}5.$$

Откуда следует, что переменная $\tilde{f}$ изменяется в пределах от 0 до 0,5.

Пример 2.1

В рассматриваемом нами примере по определению частотных свойств фильтра нижних частот разностное уравнение имеет вид

$$y_i + a_1 y_{i-1} + a_2 y_{i-2} = b_0 x_i + b_1 x_{i-1}.$$

Применяя Z-преобразование к разностному уравнению, получим

$$Y(z) = \frac{b_0 + b_1 Z^{-1}}{1 + a_1 Z^{-1} + a_2 Z^{-2}} X(z),$$

$$H(z) = \frac{b_0}{1 + a_1 Z^{-1} + a_2 Z^{-2}}.$$

Или, переходя в частотную область, запишем

$$H(i\omega T) = \frac{b_0 + b_1 e^{-i\omega T}}{1 + a_1 e^{-i\omega T} + a_2 e^{-i\omega 2T}}.$$

Амплитудно-частотная характеристика цепи равна

$$A(\omega T) = -\frac{\sqrt{(b_0 + b_1 \cos\omega T)^2 + (b_0 \sin\omega T)^2}}{\sqrt{(1 + a_1 \cos\omega T + a_2 \cos 2\omega T)^2 + (a_1 \sin\omega T + a_2 \sin 2\omega T)}}.$$

Пусть входной сигнал имеет $f_{max} = 200$ МГц, тогда $f_{Д}$ = 400 МГц. Найдём нормированную частоту $\tilde{f} = \frac{f_{вх}}{f_{Д}} = 0 \div 0{,}5.$

Используя нормированную частоту для определения амплитудно-частотной характеристики. Получим выражение:

$$A(\omega T) = \frac{\sqrt{\left(b_0 + b_1 \cos 2\pi\tilde{f}\right)^2 + \left(b_0 \sin 2\pi\tilde{f}\right)^2}}{\sqrt{\left(1 + a_1 \cos 2\pi\tilde{f} + a_2 \cos 4\pi\tilde{f}\right)^2 + \left(a_1 \sin 2\pi\tilde{f} + a_2 \sin 4\omega\tilde{f}\right)^2}}.$$

Пример 2.2

Коэффициенты фильтра разностного уравнения равны: $a_1 = -1{,}891$;

$a_2 = -0{,}9$; $b_0 = +0{,}9$; $b_1 = 0$. Таким образом, имеем

$$A(\omega T) = \frac{b_0}{\sqrt{\left(1 + a_1 \cos 2\pi\tilde{f} + a_2 \cos 4\pi\tilde{f}\right)^2 + \left(-a_1 \sin 2\pi\tilde{f} + a_2 \sin 4\omega\tilde{f}\right)^2}}.$$

Соответственно при $\tilde{f} = 0$ $A(0) = \frac{|b_0|}{|1 + a_1 + a_2|} = 1$,

$\tilde{f} = 0{,}001$ $A(0{,}001) \approx 0{,}9989$ ($\tilde{f}$ = 0,4 МГц),

$\tilde{f} = 0{,}005$ $A(0{,}005) \approx 0{,}9756$ ($\tilde{f}$ = 2,0 МГц),

$\tilde{f} = 0{,}01$ $A(0{,}01) \approx 0{,}9115$ ($\tilde{f}$ = 4,0 МГц),

$\tilde{f} = 0{,}02$ $A(0{,}02) \approx 0{,}7390$ ($\tilde{f}$ = 8,0 МГц),

$\tilde{f} = 0{,}0375$ $A(0{,}001) \approx 04930$ ($\tilde{f}$ = 15,0 МГц,)

$\tilde{f} = 0{,}05$ $A(0{,}05) \approx 0{,}3807$ ($\tilde{f}$ = 20,0 МГц),

$\tilde{f} = 0{,}1$ $A(0{,}1) \approx 0{,}1714$ ($\tilde{f}$ = 40,0 МГц).

График зависимости амплитуды от частоты приведен на рис 2.7. Из графика следует, что рассматриваемый фильтр отсекает частоты выше 40 МГц.

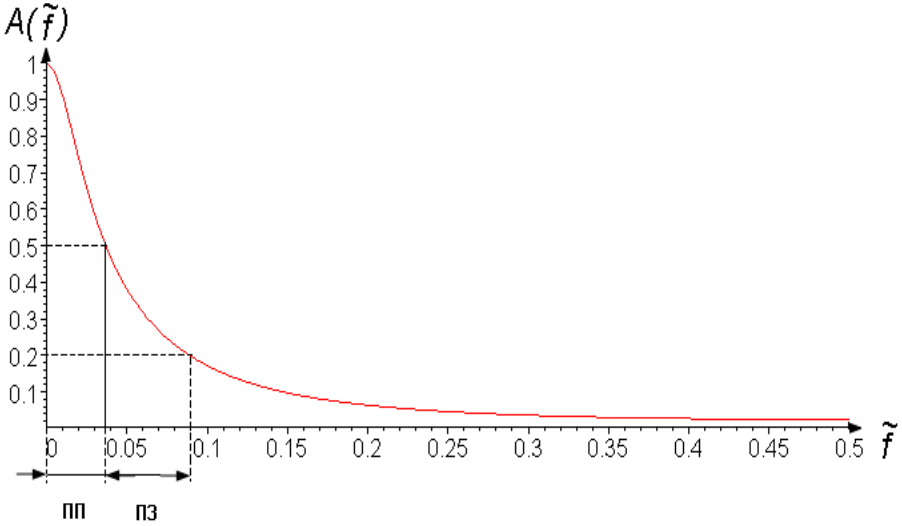


Рис. 2.7. График зависимости амплитуды от частоты для примера 2.2

2.3. Структуры линейных фильтров

Структура дискретного фильтра отражает структуру разностного уравнения и отражает все операции этого уравнения. Задержка сигнала на шаг дискретизации равносильна умножению на z-1. С учетом этого структура фильтра, описываемого разностным уравнением, представлена на рис. 2.8.

Изображенный на рис. 2.8 фильтр имеет *m*-й порядок.

Подбирая коэффициенты b_k и a_k *(k = 0, 1, 2…)*, можно построить фильтры, имеющие различные структуры с заданными характеристиками АЧХ и ФЧХ.

Устройство, реализующее цифровой фильтр, выполняет последовательное умножение входной и/или выходной выборки сигнала на коэффициенты фильтра и сложение полученных произведений.

Цифровые фильтры более высоких порядков можно реализовать несколь-кими способами. Наиболее простым способом является так называемая прямая форма реализации цифрового фильтра. В прямой форме цифровые фильтры реализуют в соответствии с формулой (2.6).

Другим способом реализации цифровых фильтров является каскадная форма. При реализации цифрового фильтра в каскадной

форме цифровой фильтр представляют в виде последовательности следующих друг за другом цифровых фильтров 1-го или 2-го порядка.

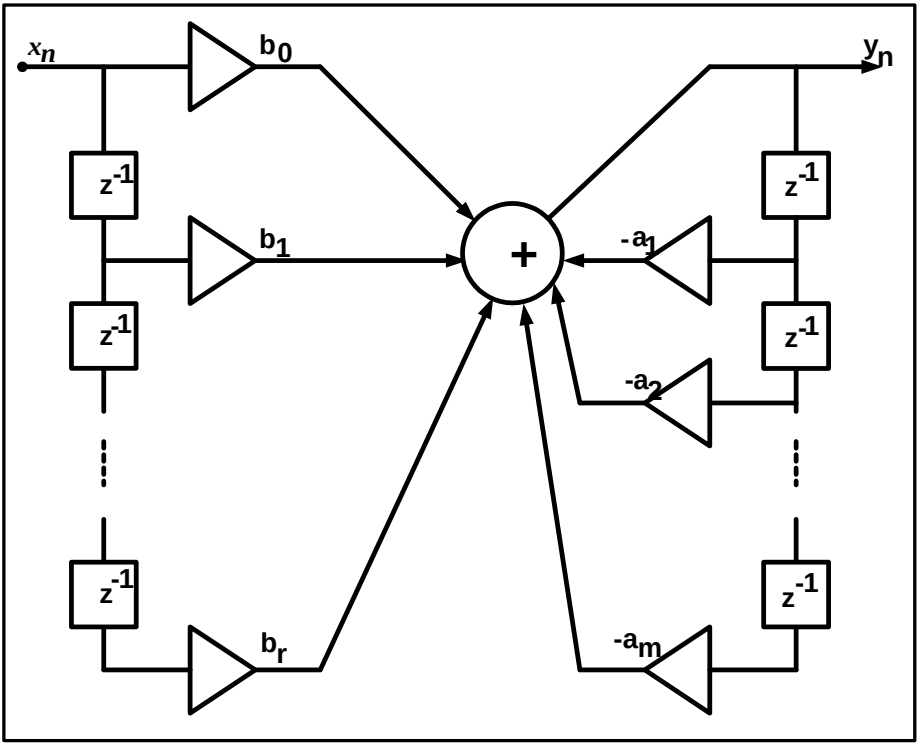


Рис. 2.8. Обобщенная структура КИХ-фильтра

Однако прямая реализация и расчёт характеристик фильтров высокого порядка является сложной задачей. В связи с этим, как будет показано далее, они могут быть преобразованы в некоторую совокупность более простых фильтров. Поэтому в технике цифровой обработки сигналов - фильтры большой размерности строятся как некоторая совокупность фильтров малой размерности. Рассмотрим некоторые базовые рекурсивные фильтры, используемые для построения сложных фильтров большой размерности.

2.3.1. Базовые фильтры 1-го порядка

Пример 2.3. Базовый фильтр 1-го порядка.

$$y_n = x_n - a_1 y_{n-1}, \qquad n = 0,1,2,...$$

Структура фильтра представлена на рис. 2.9.

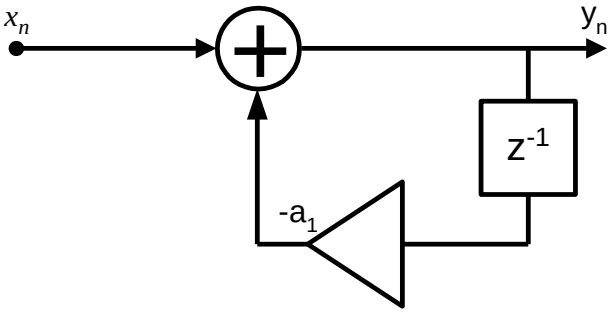


Рис. 2.9. Базовый фильтр 1-го порядка

Передаточная функция фильтра

$$H(z) = \frac{1}{1 + a_1 z^{-1}}.$$

Импульсная характеристика

$$h_n = (-a_1)^n, \qquad n = 0,1,2...$$

Частотная характеристика

$$H(e^{i\omega T}) = \frac{1}{1 + a_1 e^{-i\omega T}}.$$

Амплитудно-частотная характеристика

$$\left|H(e^{i\omega T})\right| = \frac{1}{\sqrt{1+2a_1\cos\omega T + a_1^2}}.$$

Фазо-частотная характеристика.

$$\varphi(\omega\mathrm{T}) = arctg\,\frac{a_1\sin\omega T}{1+a_1\cos\omega T}.$$

Пример 2.4. Рекурсивный фильтр 1-го порядка

$$y_n = b_0 x_n + b_1 x_{n-1} - a_1 y_{n-1}, \qquad \text{n = 0, 1, 2, …}$$

$$H(z) = \frac{b_0 + b_1 z^{-1}}{1+a_1 z^{-1}}.$$

Структура фильтра представлена на рис. 2.10.

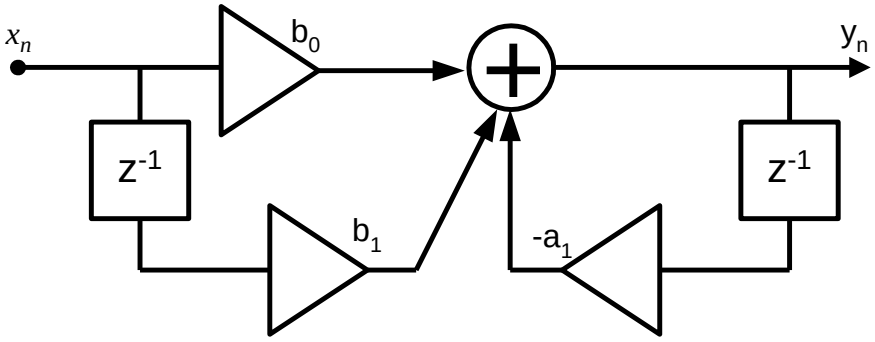


Рис. 2.10. Рекурсивный фильтр 1-го порядка

Импульсная характеристика фильтра

$$h_{0,n} = b_0(-a_1)^n, n = 0,1,2...;$$

$$h_{1,n} = b_1(-a_1)^n,\ n = 1,2...$$

Амплитудно-частотная характеристика

$$A(\omega T) = \frac{\sqrt{b_0^2 + 2b_0 b_1 \cos \omega T + b_1^2}}{\sqrt{1 + 2a_1 \cos \omega T + a_1^2}}.$$

Фазо-частотная характеристика

$$\varphi(\omega T) = -arctg \frac{-b_1 \sin \omega T}{b_0 + b_1 \cos \omega T} - arctg \frac{-a_1 \sin \omega T}{1 + a_1 \cos \omega T}.$$

2.3.2. Базовые фильтры 2-го порядка

Пример 2.5. Базовый рекурсивный фильтр 2-го порядка.

$$\left.\begin{aligned} &y_n = x_n - a_1 y_{n-1} - a_2 y_{n-2}, \qquad n = 0,1,2,... \\ &H(z) = \frac{1}{1 - a_1 z^{-1} - a_2 z^2}. \end{aligned}\right\}.$$

Амплитудно-частотная характеристика

$$A(\omega T) = \frac{1}{\sqrt{(1 - a_1 \cos \omega T + a_2 \cos 2\omega T)^2 + (1 - a_1 \sin \omega T + a_2 \sin 2\omega T)^2}}.$$

Фазо-частотная характеристика

$$\varphi(\omega T) = -arctg\,\frac{-(a_1 \sin \omega T + a_2 \sin 2\omega T)}{1 + a_1 \cos \omega T + a_2 \cos 2\omega T}.$$

Структура фильтра приведена на рис. 2.11.

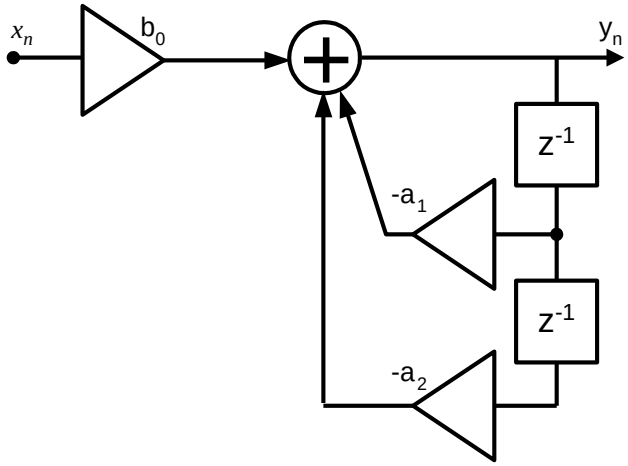


Рис. 2.11. Структура базового рекурсивного фильтра 2-го порядка

Пример 2.6. Рекурсивный фильтр 2-го порядка.

$$y_n = b_0 x_n + b_1 x_{n-1} - a_1 y_{n-1} - a_2 y_{n-2},$$

$$H(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}}.$$

Структура фильтра приведена на рис. 2.12.

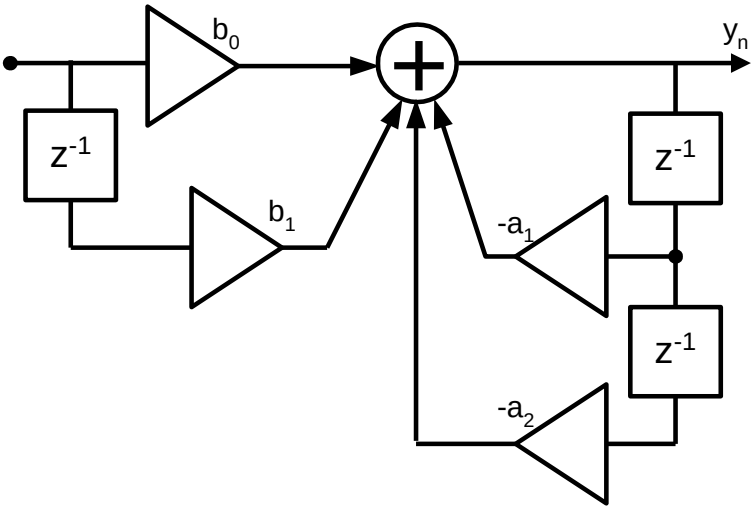


Рис. 2.12. Структура рекурсивного фильтра 2-го порядка

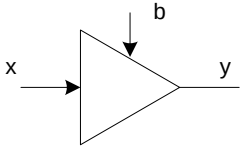
Основные характеристики фильтров определяются аналогично базовому рекурсивному фильтру 2-го порядка.

2.3.3. Базовые фильтры 3-го порядка

Рекурсивный фильтр 3-го порядка также является базовым для построения фильтров большой размерности, но в данном случае на его примере рассмотрены методы преобразования и реализации сложных фильтров через простые.

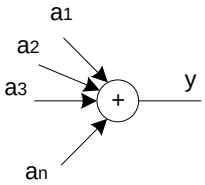
В соответствии с [3,4] можно выделить три основные формы реализации линейных цифровых фильтров рекурсивного типа: прямая, последовательная и параллельная.

Прямая форма реализации нами уже рассматривалась. Она основывается на том, что исходное уравнение фильтра непосредственно преобразуется в структуру. Для этого каждой операции ставится в соответствие свой блок. В нашем случае таких операция две: умножение



$$\left(y = x \cdot b\right)$$

и суммирование



$$\left(y = \sum_{i=1}^{n} a_i \right).$$

Разностное уравнение, описывающее фильтр 3-го порядка, приведено ниже

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} - a_1 y_{n-1} - a_2 y_{n-2} - a_3 y_{n-3}.$$

Передаточной функции соответствует выражение

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}.$$

Тогда структура в соответствии с уравнением фильтра его будет иметь вид, приведенный на рис. 2.13.

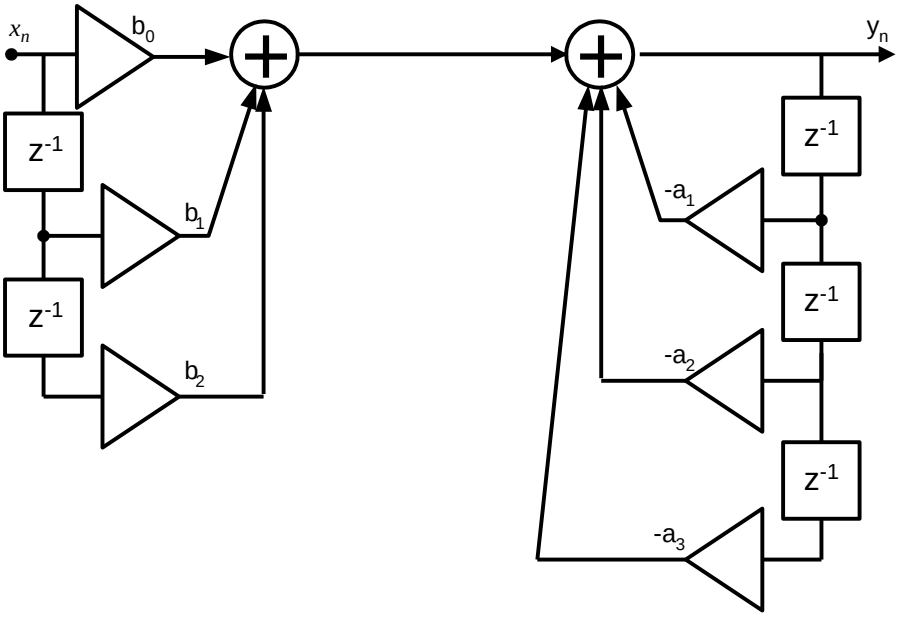


Рис. 2.13. Структура рекурсивного фильтра 3-го порядка

Рассматриваемые выше формы реализации линейных цифровых фильтров рекурсивного типа относятся к прямой реализации передаточной функции.

Рассмотрим теперь последовательную форму. В общем случае для получения последовательной структуры реализации линейного фильтра его передаточная функция должна быть представлена в виде

$$H(z) = \frac{\sum_{l=0}^{r} b_l z^{-1}}{1 + \sum_{l=1}^{m} a_l z^{-1}} = \prod_{i=1}^{R} H_i(z).$$

Если передаточная функция допускает запись в форме (2.22), то последовательная структура цифровой цепи предполагает следующую форму записи исходной системы:

$$\left.\begin{aligned} y_1(nT) &= -\sum_{k=1}^{m_1} a_{1k} y_1(nT-kT) + \sum_{k=1}^{r_1} b_{1k} x(nT-kT); \\ y_2(nT) &= -\sum_{k=1}^{m_2} a_{2k} y_2(nT-kT) + \sum_{k=0}^{r_2} b_{2k} y_1(nT-kT); \\ &\dots \\ y_R(nT) &= -\sum_{k=1}^{m_R} a_{Rk} y_R(n-kT) + \sum_{k=0}^{r_R} b_{Rk} y_{(R-1)}(nT-kT). \end{aligned}\right\}$$

Соответственно система передаточных функций будет иметь вид

$$H_1(z) = \frac{\sum_{k=0}^{r_1} b_{1k} z^{-k}}{1+\sum_{k=0}^{m_1} a_{1k} z^{-k}}; H_2(z) = \frac{\sum_{k=0}^{r_2} b_{2k} z^{-k}}{1+\sum_{k=0}^{m_2} a_{2k} z^{-k}}; \dots \quad H_R(z) = \frac{\sum_{k=0}^{r_R} b_{Rk} z^{-k}}{1+\sum_{k=0}^{m_R} a_{Rk} z^{-k}}.$$

Наиболее простые структуры получаются, если положить $b_{j0}=1$, а $b_{jk} = 0$ для всех $k \neq 0$ (j = 2, 3, 4, …, R).

Для фильтра 3-го порядка получим последовательную структуру, состоящую из фильтров 2-го и 1-го порядков (рис 2.14):

$$y_{1n} = b_{10}x_n + b_{11}x_{n-1} - a_{11}y_{1(n-1)} - a_{12}y_{1(n-2)},$$

$$y_{2n} = b_{20}y_{1n} + b_{21}y_{1(n-1)} - a_{21}y_{2(n-1)},$$

$$\left.\begin{aligned} H_1(z) &= \frac{b_{10}+b_{11}z^{-1}}{1+a_{11}z^{-1}+a_{12}z^{-2}}, \\ H_2(z) &= \frac{b_{20}+b_{21}z^{-1}}{1+a_{21}z^{-1}}. \end{aligned}\right\}.$$

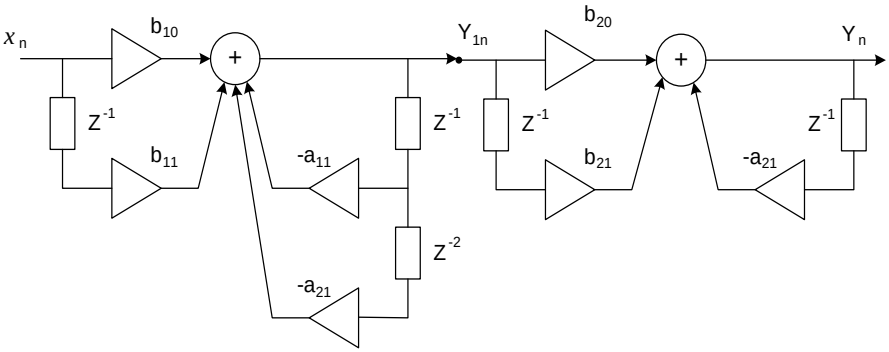


Рис. 2.14. Последовательная структура фильтра

Как видим, задержки внутри фильтра (рис. 2.14) выполняют одну и ту же функцию. Поэтому одна из них может быть исключена. Таким образом, последовательная форма реализации фильтров допускает минимизацию числа линий задержек. Действительно, представим передаточную функцию фильтра в виде произведения двух передаточных функций H1(z) и H2(z):

$$H(z) = H_1(z)H_2(z) = \frac{1}{A(z)}B(z)$$

Выше структура такого фильтра формировалась как последовательное соединение двух фильтров. Например, фильтру 3-го порядка и передаточным функциям вида (2.40) соответствуют следующие уравнения:

$$\left.\begin{aligned} y_{1n} &= x_n - a_1 y_{1(n-1)} - a_2 y_{1(n-2)} - a_3 y_{1(n-3)}; \\ y_n &= b_0 y_{1n} - b_1 y_{1(n-1)} - b_2 y_{1(n-2)} - b_3 y_{1(n-3)}. \end{aligned}\right\}$$

Структура фильтра, реализующая систему (2.26), показана ниже (рис. 2.15).

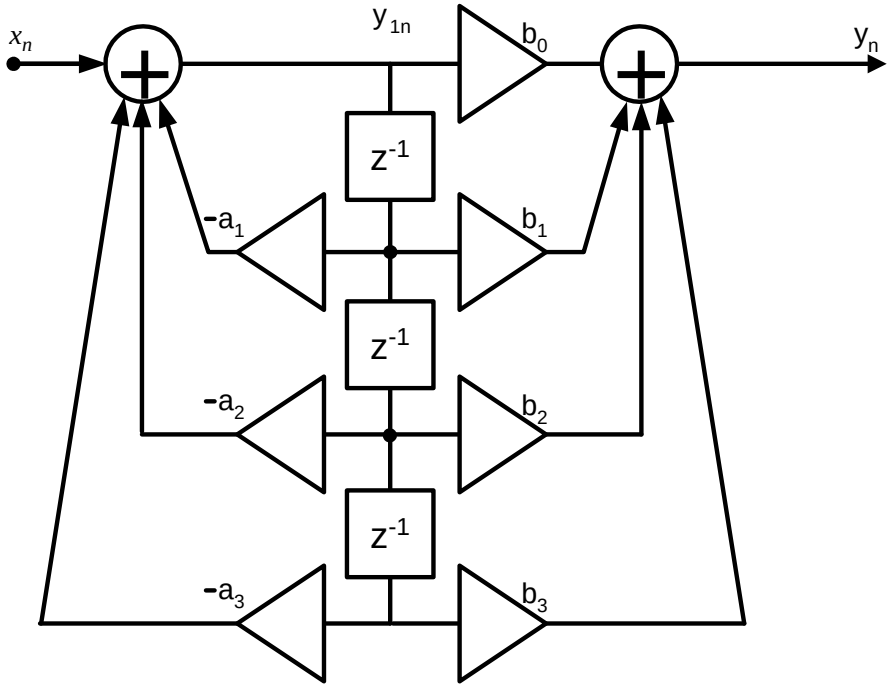


Рис. 2.15. Структура фильтра для системы (2.22)

Структуры рассматриваемого типа обладают минимальным набором линий задержек. Их число в два раза ниже, чем при прямой реализации. Используя этот прием, можно минимизировать все рассмотренные выше структуры.

После получения последовательного фильтра необходимо найти значения коэффициентов. Для их определения перемножим числители и знаменатели передаточных функций $H_i(z)$. Затем приравняем коэффициенты при одинаковых переменных z^{-j}. В результате получаем систему алгебраических уравнений для определения коэффициентов передаточных функций $H_i(z)$.

Получим следующую систему уравнений:

$$\left.\begin{array}{c} b_{10}\cdot b_{20}=b_0, \\ b_{10}\cdot b_{21}+b_{11}\cdot b_{20}=b_1, \\ b_{11}\cdot b_{21}=b_2. \end{array}\right\}$$

Данная система содержит 4 неизвестных и 3 уравнения. Поэтому одним коэффициентом необходимо задаться или ввести дополнительное условие. Например, положить, что $b_{10}=b_{20}$. Тогда $b_{10}=b_{20}=\sqrt{b_0}$. И система (2.27) становится разрешимой, т.е.

$$\left.\begin{array}{c} b_{21}+b_{11}=\dfrac{b_1}{\sqrt{b_0}}, \\ b_{11}\cdot b_{21}=b_2. \end{array}\right\}$$

Откуда получаем:

$$b_{21}^2-\frac{b_1}{\sqrt{b_0}}b_{21}+b_2=0.$$

Аналогично определяются коэффициенты a_{ij}:

$$\left.\begin{array}{c} a_{11}+a_{21}=a_1, \\ a_{12}\cdot a_{11}+a_{12}\cdot a_{11}=a_2, \\ a_{12}\cdot a_{21}=a_3. \end{array}\right\}$$

Эта система полная, т.е. число уравнений и неизвестных совпадает. После отыскания коэффициентов фильтр считается определенным.

Параллельная структура цифровой цепи предполагает следующую форму передаточной функции:

$$H(z)=H_1(z)+H_2(z)$$

Тогда

$$\left.\begin{aligned} Y_1(z) &= H_1(z)X(z), \\ Y_2(z) &= H_2(z)X(z), \\ Y(z) &= Y_1(z) + Y_2(z). \end{aligned}\right\}$$

Для фильтра 3-го порядка передаточная функция при параллельной форме записывается в виде двух выражений:

$$\left.\begin{aligned} H_1(z) &= \frac{b_{10} + b_{11}z^{-1} + b_{12}z^{-2}}{1 + a_{11}z^{-1} + a_{12}z^{-2}}, \\ H_2(z) &= \frac{b_{20} + b_{21}z^{-1} + b_{22}z^{-2}}{1 + a_{21}z^{-1} + a_{22}z^{-2}}. \end{aligned}\right\}$$

Коэффициенты b_{1k}, b_{2k} и a_{1k}, a_{2k} подбираются так, чтобы *H(z)* была равна передаточной функции исходного фильтра.

Таким образом, получаем два уравнения:

$$\left.\begin{aligned} y_{1n} &= b_{10}x_n + b_{11}x_{n-1} + b_{12}x_{n-2} - a_{11}y_{1(n-1)}a_{12}y_{1(n-2)}; \\ y_{2n} &= b_{20}x_n + b_{21}x_{n-1} + b_{22}x_{n-2} - a_{21}y_{2(n-1)}a_{22}y_{2(n-2)}; \\ y_n &= y_{1n} + y_{2n}. \end{aligned}\right\}$$

Структура фильтра показана на рис. 2.16.

Прямая структура рекурсивного фильтра всегда содержит задержки для входного и выходного сигналов. Это увеличивает число ячеек памяти при реализации цифрового линейного фильтра. Рассмотрим возможные пути уменьшения их числа.

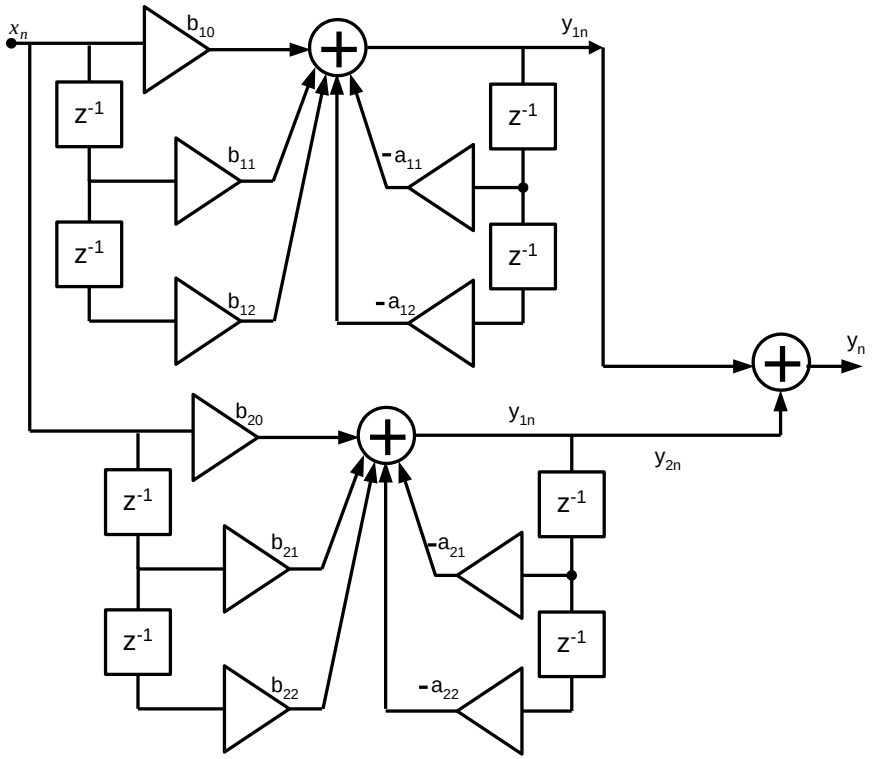


Рис. 2.16. Структура фильтра с параллельной реализацией

Пусть задана передаточная функция, описывающая рекурсивный фильтр 2-го порядка:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{B(z)}{A(z)}.$$

Существует еще один подход к построению цифровых фильтров. Он основан на представлении разностного уравнения порядка m системой уравнений первого порядка (аналог нормальной системы линейных дифференциальных уравнений с постоянными

коэффициентами). Возьмем за основу, как и в предыдущем случае, разностное уравнение 2-го порядка:

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} - a_1 y_{n-1} - a_2 y_{n-2}.$$

Введем вполне понятные обозначения:

$$\left.\begin{aligned} y_n &= b_0 x_n + y_{1(n-1)}, \\ y_{1n} &= b_1 x_n - a_1 y_n + y_{2(n-1)}, \\ y_{2n} &= b_2 x_n - a_2 y_n. \end{aligned}\right\} \quad (2.34)$$

Структура фильтра, реализующая систему (2.34), показана на рис. 2.17.

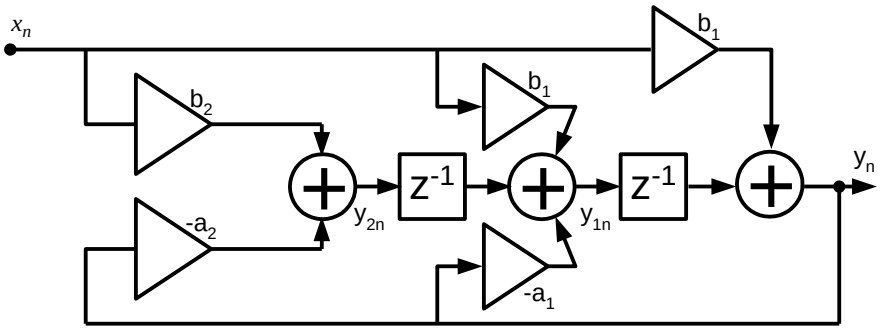


Рис. 2.17. Структура оптимизированного фильтра

Фильтр также является минимальным по числу элементов задержки.

Целесообразность использования прямой, последовательной или параллельной формы сложно строго обосновать, но можно сформулировать некоторые общие подходы к выбору типа структуры. Во-первых, в каждой эквивалентной структуре очень важно определить влияние погрешностей задания коэффициентов на точность формирования выходной функции. В прямой форме реализации это влияние оказывается наиболее ощутимым.

В последовательной или параллельной форме реализации фильтра погрешности коэффициентов оказывают меньшее влияние на выходную переменную. Однако в обоих случаях необходимо, чтобы существовала возможность представить знаменатель выражения для переходной функции в виде произведения.

При параллельной реализации фильтра суммирование выходных переменных различных фильтров, каждый из которых имеет свой динамический диапазон, приводит к проблеме вычитания больших чисел.

2.4. Расчет КИХ-фильтров

Основное отличие КИХ-фильтров от БИХ-фильтров заключается в том, что у КИХ- фильтров отсутствует обратная связь и импульсная характеристика у них всегда конечна.

Стандартный КИХ-фильтр описывается следующим разносным уравнением:

$$y_n = \sum_{k=0}^{r-1} b_k x_{n-k},$$

где n – длина дискретной последовательности x_n, $n = 0,1,2,\dots,N-1$, $b_k, k = 0,1,2,\dots,r-1$ – коэффициенты фильтра.

Передаточная функция КИХ-фильтров имеет вид

$$H(Z) = \sum_{k=0}^{r-1} b_k Z^{-k},$$

$$Y(Z) = H(Z) \cdot X(Z).$$

Импульсная характеристика КИХ-фильтра есть реакция фильтра на единичный импульс. Подставляя $X(Z) = 1$, получим отсчёты импульсной характеристики, которые будут равны:

$$h_0 = b_0,\ h_1 = b_1,\ h_{r-1} = b_{r-1},$$

Указанные отсчёты и являются обратным Z-преобразованием. Соответственно формула (2.35) есть прямое Z-преобразование.

Очень важным свойством КИХ-фильтра является возможность получения линейной фазовой характеристики.

Вторым положительным свойством КИХ-фильтра является простота реализации.

Фазовая характеристика линейна, если выполняется следующее условие:

$$\varphi(\varpi) = -\alpha(\varpi) \text{ или } \varphi(\varpi) = \beta - \alpha(\varpi).$$

Нетрудно показать, что для выполнения первого условия необходимо, чтобы коэффициенты фильтра обладали положительной симметрией

$$b_k = b_{(r-1-k)}, \begin{cases} k = 0,1,...,{(r-1)}/{2}, \ \text{r} - nechet; \\ k = 0,1,...\left(\dfrac{r-1}{2} - 1\right), \ \text{r} - chet. \end{cases},$$

Второе условие выполняется, когда коэффициенты фильтра имеют отрицательную симметрию:

$$b_k = -b_{(r-1-k)},$$
$$\alpha = \frac{r-1}{2}, \beta = \frac{\pi}{2}.$$

Покажем это для фильтра с положительной симметрией коэффициентов. Запишем передаточную функции по частоте H(ωT):

$$H(\omega T) = \sum_{k=0}^{r-1} b_k e^{-i\omega Tk}.$$

Предположим, что r – нечётное, переходя к нормированной частоте, произведём замену $\omega = 2\pi f$, а $T = 1/fd$.

Тогда $\tilde{f} = \frac{f}{f_d}$ и

$$H(e^{-i2f}) = e^{-i2\pi \cdot f\left(\frac{r-1}{2}\right)}\left(\sum_{k=0}^{\frac{r-1}{2}}\left(b_k \cdot e^{-i2\pi \cdot f \cdot k} - b_{r-1-k}\right)\cdot e^{-i2\pi \cdot f \cdot k}\right).$$

Так как $b_k = b_{(r-1-k)}$, из условия (2.37), то

$$H(e^{-i2\pi \cdot f}) = e^{-i2\pi \cdot f(r-1)} \cdot \sum_{k=0}^{r-1} d_k \cos 2\pi k,$$

где $d_0 = b_{\frac{r-1}{2}}, d_k = 2b_{(\frac{r-1}{2}-k)}$.

Откуда $A\left(\tilde{f}\right) = \left|\sum_{k=0}^{r-1/2} d_k \cos 2\pi k\right|$,

$\varphi\left(\tilde{f}\right) = -\pi \cdot rc \cdot tg\,\frac{\sin \pi\tilde{f}(r-1)}{\cos \pi\tilde{f}(r-1)} = -\pi\tilde{f}(r-1) + m\pi$, где m = *0, 1, 2 ...*

Амплитудно-частотная характеристика вычисляется аналогично БИХ-фильтру за исключением того, что в связи с отсутствием обратной связи знаменатель также отсутствует. Соответствующая формула приведена ниже. Здесь также следует перейти от круговой частоты к нормированной по правилам, рассмотренным при анализе БИХ-фильтров.

$$A(\omega T) = \sqrt{\left(\sum_{k=0}^{p-1} b_k \cos \omega \cdot kT\right)^2 + \left(\sum_{k=0}^{p-1} b_k \sin \omega \cdot kT\right)^2}$$

Тогда

$$A(\omega T) = \sqrt{\left(\sum_{k=0}^{p-1} b_k \cos 2\pi \cdot \tilde{f} \cdot kT\right)^2 + \left(\sum_{k=0}^{p-1} b_k \sin 2\pi \cdot \tilde{f} \cdot kT\right)^2},$$

где $\tilde{f}$ - нормированная частота.

Вид типовой АЧХ КИХ-фильтра низких частот показан на рис. 2.18.

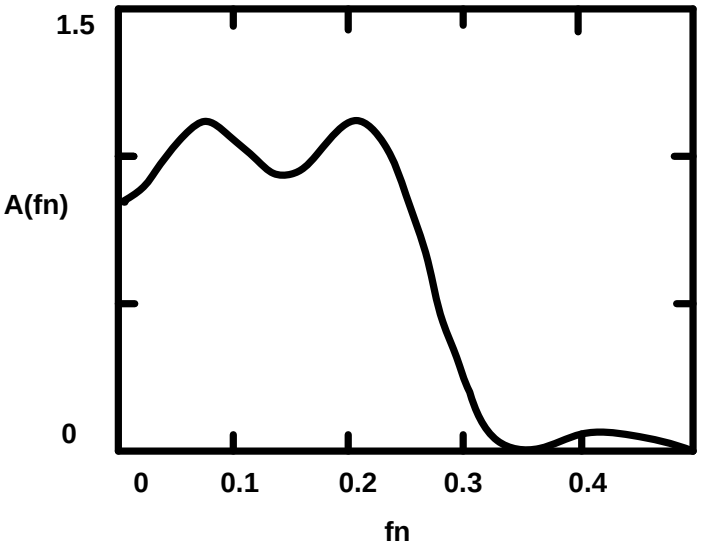


Рис. 2.18. Типовая АЧХ КИХ-фильтра низких частот

Фазо -частотная характеристика такого фильтра может быть представлена выражением

$$\phi(fn) := \operatorname{atan}\left[\frac{\sum_{j=0}^{15}\left(b_j \cdot \sin\left(2\cdot\pi\cdot fn\cdot j\right)\right)}{\sum_{j=0}^{15}\left(b_j \cdot \cos\left(2\cdot\pi\cdot fn\cdot j\right)\right)}\right],$$

и имеет вид, представленный на рис. 2.19.

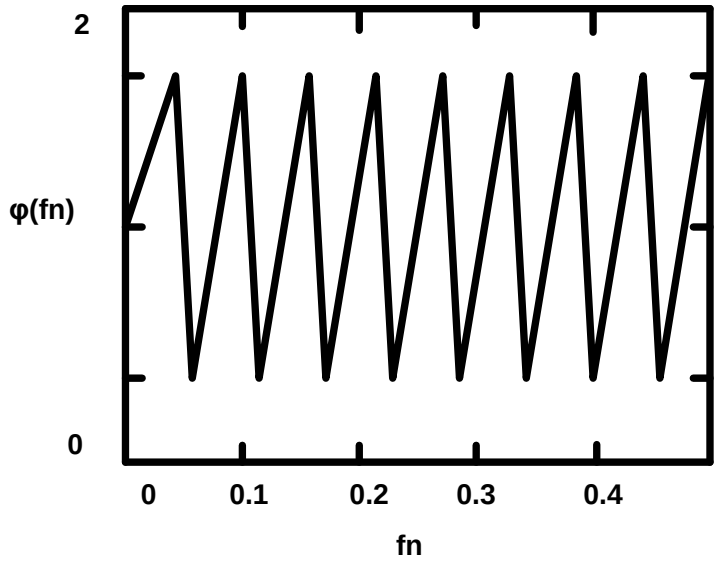


Рис. 2.19. Типовая ФЧХ КИХ-фильтра низких частот

Структура фильтра отражает реализацию его разносного уравнения, которое имеет вид:

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} + \cdots + b_{r-1} x_{n-r-1},$$

где *n = 0, 1, …, N – 1*.

Структура на основе схем задержки и умножителей (трансверсальная структура) приведена на рис. 2.20.

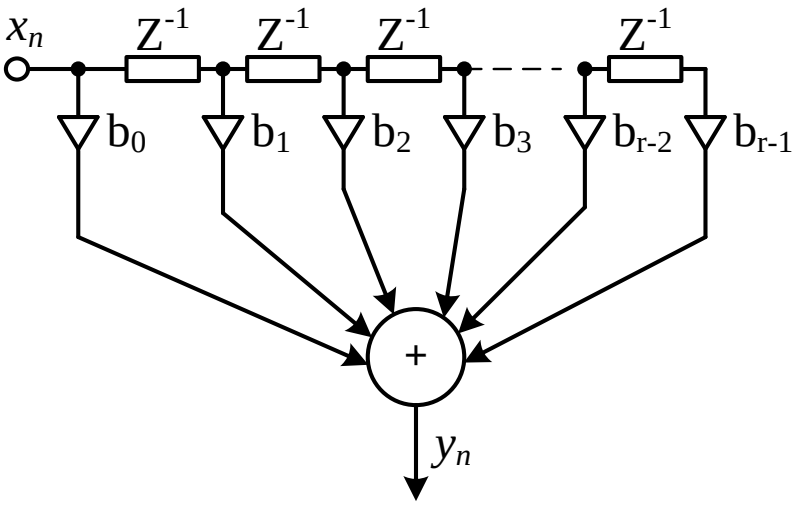


Рис. 2.20. Трансверсальная структура КИХ-фильтра

2.5. Аналитический расчет и моделирование цифровых фильтров

Рассмотрим пример расчета КИХ-фильтра в аналитическом виде при помощи программного комплекса MathCad.

Для простоты, зададим цифровой фильтр 16-го порядка в виде набора коэффициентов, определяющих его свойства (Рис. 2.21).

Так как данная линейная система является КИХ-фильтром, то в ней присутствует только одно множество коэффициентов, отвечающее за преобразование входных данных без обратной связи. Структурная схема такого фильтра представлена на Рис. 2.22.

В соответствии с представленным выше материалом, определим характеристики фильтра, а именно его амплитудной частотную (АЧХ) и фазовой частотной (ФЧХ) характеристики.

Так как разрабатываемая линейная система является цифровым КИХ-фильтром, то выражение АЧХ вырождается в следующее:

$$A(fn) := \left| \sqrt{\left[\sum_{j=0}^{15} \left(b_j \cdot \cos\left(2\pi \cdot fn \cdot j\right)\right)\right]^2 + \left[\sum_{j=0}^{15} \left(b_j \cdot \sin\left(2\pi \cdot fn \cdot j\right)\right)\right]^2} \right|$$

Данная характеристика будет иметь вид, представленный на следующем рис. 2.23.

$$b := \begin{pmatrix} -1.495023781 \cdot 10^{-2} \\ -4.918541221 \cdot 10^{-2} \\ -1.938521791 \cdot 10^{-2} \\ 5.715285601 \cdot 10^{-2} \\ -2.447212951 \cdot 10^{-2} \\ -1.151955091 \cdot 10^{-1} \\ 9.206225521 \cdot 10^{-2} \\ 4.947339961 \cdot 10^{-1} \\ 4.947339961 \cdot 10^{-1} \\ 9.206225521 \cdot 10^{-2} \\ -1.151955091 \cdot 10^{-1} \\ -2.447212951 \cdot 10^{-2} \\ 5.715285601 \cdot 10^{-2} \\ -1.938521791 \cdot 10^{-2} \\ -4.918541221 \cdot 10^{-2} \\ -1.495023781 \cdot 10^{-2} \end{pmatrix}$$

Рис. 2.21. Коэффициенты КИХ-фильтра

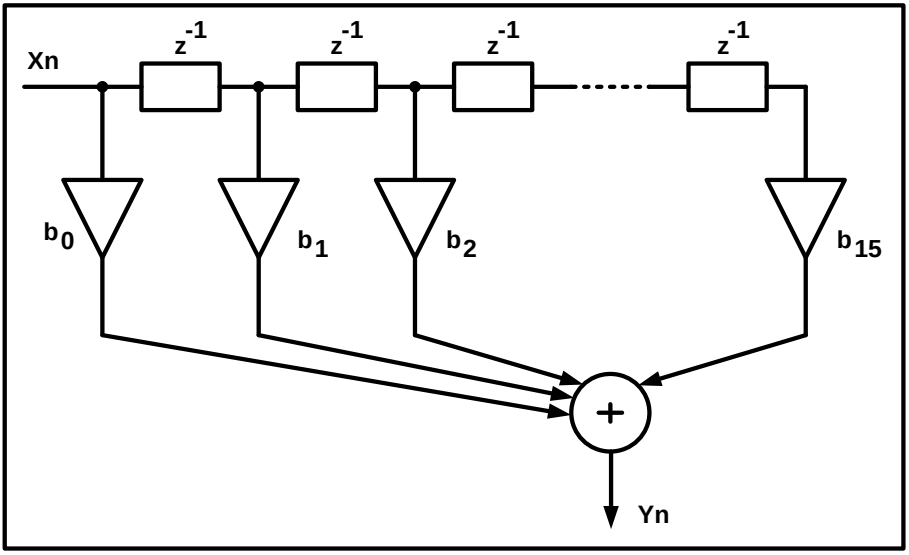


Рис. 2.22. Структурная схема реализации КИХ-фильтра

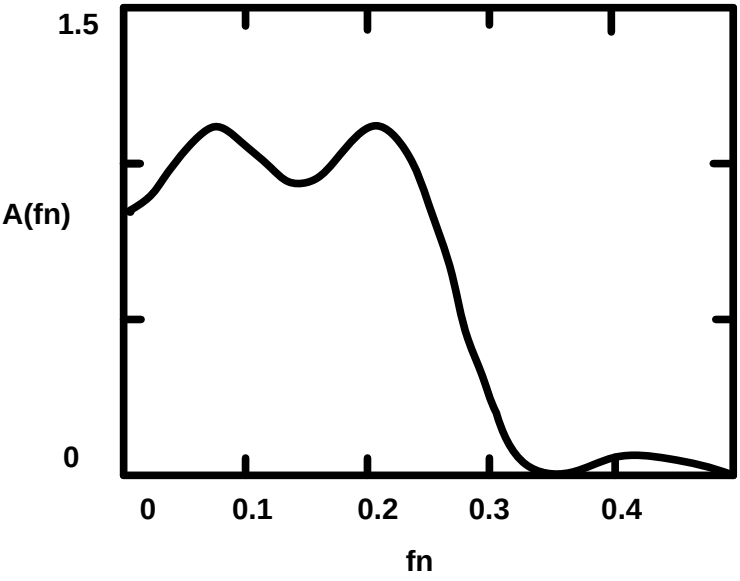


Рис. 2.23. АЧХ разрабатываемого КИХ-фильтра

Фазовая частотная характеристика фильтра, как следует из (3.19), может быть представлена в виде выражения

$$\phi(fn) := \operatorname{atan}\left[\frac{\sum_{j=0}^{15}\left(b_j \cdot \sin\left(2\cdot\pi\cdot fn\cdot j\right)\right)}{\sum_{j=0}^{15}\left(b_j \cdot \cos\left(2\cdot\pi\cdot fn\cdot j\right)\right)}\right],$$

и имеет вид, представленный на рис. 2.24.

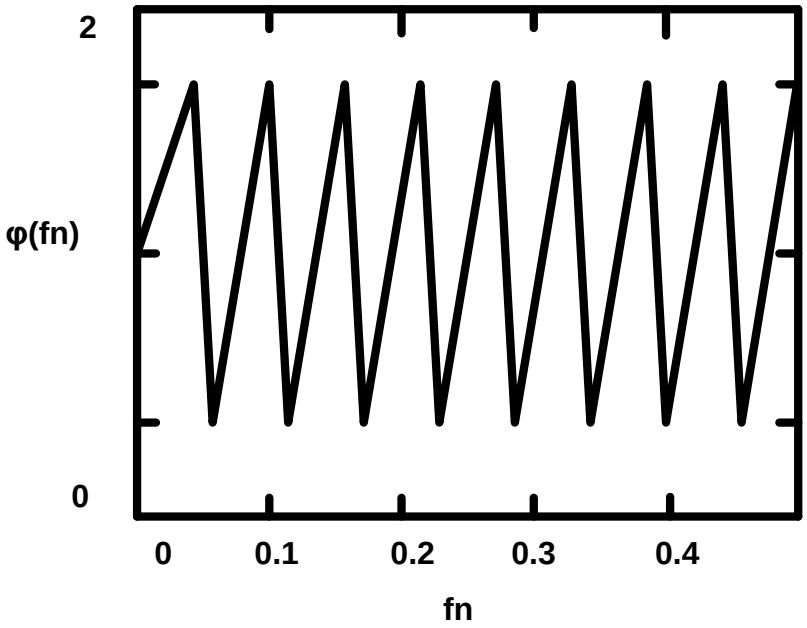


Рис. 2.24. ФЧХ разрабатываемого КИХ-фильтра

Обработка исходного сигнала производится посредством дискретной свертки последовательности отсчетов x_n входного сигнала с коэффициентами фильтра – элементами вектора b

$$Ymc_n := \sum_{j=0}^{15} \left(b_j \cdot X_{z(n,j)} \right) .$$

В качестве тестового сигнала возьмем сумму семи синусоид разной частоты f и сила (амплитуды) A (рис. 2.25).

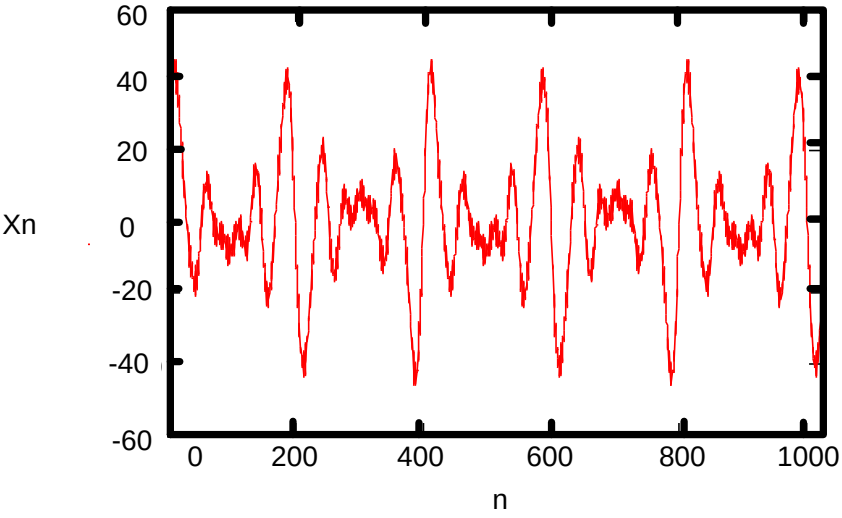
$$f := \begin{pmatrix} 300 \\ 500 \\ 700 \\ 900 \\ 1200 \\ 1500 \\ 18000 \end{pmatrix} \qquad Ampl := \begin{pmatrix} 7 \\ 9 \\ 18 \\ 12 \\ 2 \\ 4 \\ 4 \end{pmatrix}$$

Рис. 2.25. Параметры генерации тестового сигнала

Сам сигнал, в виде дискретной последовательности из 1024-х отсчетов, получается по следующей формуле

$$X_n := \sum_{i=0}^{6} \left(Ampl_i \cdot \sin\left(2 \cdot \pi \cdot f_i \cdot t_n \right) \right) .$$

Исходный сигнал имеет вид, представленный на рис. 2.26.

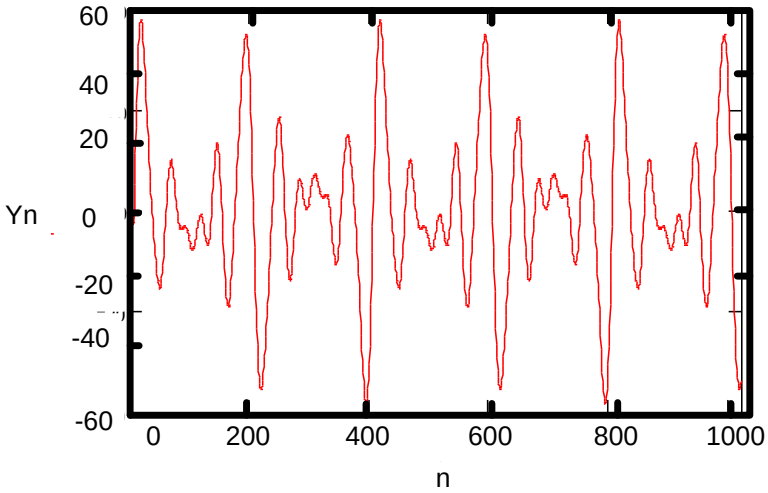


X =

	0
0	0
1	7.247
2	9.542
3	20.762
4	18.988
5	31.59
6	28.029
7	38.687
8	36.043
9	41.871
10	42.163
11	41.755
12	45.5
13	39.447
14	45.456
15	36.125

Рис. 2.26. Исходная форма сигнала

После обработки линейным фильтром, получилась очищенная форма сигнала, представленная на рис. 2.27.

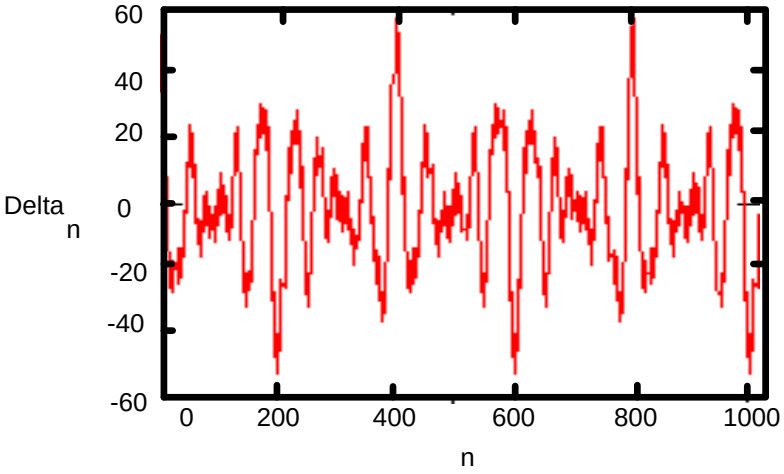


Ymc=

	0
0	0
1	-0.108
2	-0.499
3	-0.92
4	-1.076
5	-1.441
6	-2.223
7	-2.424
8	0.428
9	5.71
10	11.905
11	16.992
12	21.741
13	26.19
14	29.972
15	33.129

Рис. 2.27. Очищенная форма исходного сигнала

Разница между исходным и отфильтрованным сигналом составляет отсеянный шум, представленный на рис. 2.28.



DeltaYmcX=

	0
0	0
1	7.355
2	10.041
3	21.682
4	20.064
5	33.031
6	30.251
7	41.111
8	35.616
9	36.16
10	30.258
11	24.763
12	23.759
13	13.256
14	15.484
15	2.995

Рис. 2.28. Отсеянный шум

Совмещенные графики исходного и очищенного сигналов показаны на рис. 2.29.

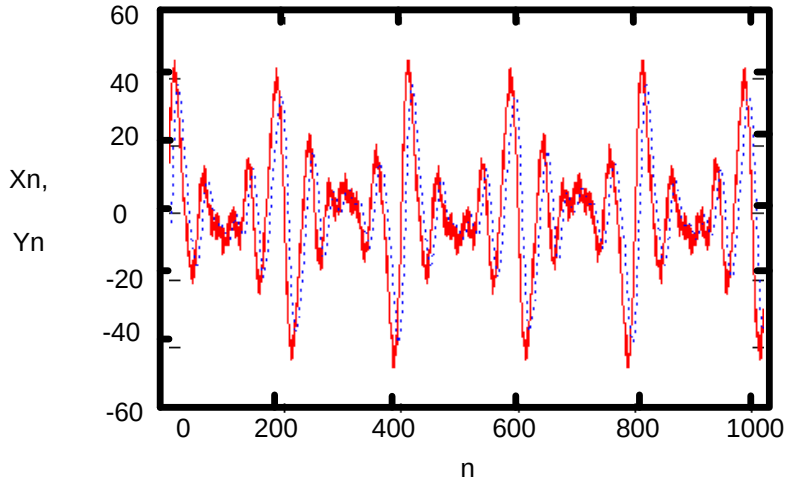


Рис. 2.29. Совмещенные формы зашумленного и очищенного сигналов

Получив результат в аналитической форме, необходимо реализовать вариант расчета фильтра для реализации на микропроцессорном уровне.

2.6. Программная реализация цифровых фильтров

Сложность вычислительной реализации фильтра определяется числом звеньев *N*. В цифровых процессорах для вычисления отсчета *y(n)* требуется приблизительно *N*n* команд (где *n* – количество тактов процессора для вычисления составной операции «умножение с накоплением», которое лежит в основе фильтра). Поэтому следует проверить возможность реализации такого числа операций за период дискретизации сигнала. Например, при частоте дискретизации сигнала 8 кГц период составляет 125 мкс. При частоте выполнения команд 50 МГц число команд, выполненных за период дискретизации, равно 6 250 и необходимо выполнить условие *N*n*<6250.

При программировании фильтра КИХ длиной *N* в памяти процессора организуются два буфера длиной *N* для хранения данных и коэффициентов фильтра. В буфер коэффициентов вводятся данные из соответствующего файла. Вычисление весовой суммы выполняется с

помощью многофункциональной команды, состоящей из умножения с накоплением в аккумуляторе и двух обращений к памяти.

Таким образом, для получения весовой суммы из *N*n* слагаемых выполняется цикл из *(N-1)*n* шага и дополнительное умножение с накоплением вне цикла.

Рассмотрим пример реализации КИХ-фильтра, написанный на языке Си для обработки 1024 отсчетов дискретизированного сигнала [9].

Пример 2.7. Процедура программной фильтрации цифровых данных.

```
float fir_filter(float input) {
     int i;
     static float sample[1024];
     float acc;
     float output;

     sample[0] = input;

     /* Аккумулятор */
     acc = 0.0f;

     /* Умножение с накоплением */
     for (i = 0; i < 1024; i++) {
       acc += (h[i] * sample[i]);
     }

     /* Выход */
     output = acc;

     /* Смещаем задержаный сигнал */
     for (i = 127; i > 0; i--)
       sample[i] = sample[i - 1];
     return output;
}
```

Рассмотрим далее упрощенную аппаратную реализацию цифрового фильтра в базисе систем на кристалле (ПЛИС).

2.7. Аппаратная реализация цифровых фильтров

Программная реализация цифровых фильтров является эффективной тогда, когда в проблемно-ориентированной системе присутствует процессор цифровой обработки сигналов или решение задачи производится на ПЭВМ. Во всех иных случаях фильтры применяются либо в аналоговой форме, либо в виде схемотехнического решения на базе системы на кристалле.

На базе ПЛИС есть несколько вариантов решения задачи цифровой фильтрации:

1. Использование пакетов математического моделирования для автома-тизированной генерации коэффициентов фильтров с заданными параметрами для последующей реализации в базисе ПЛИС [11].
2. Использование встроенных средств САПР Quartus для реализации цифрового фильтра при помощи MegaWizard Plugin Manager – FIR Compiler [13].
3. Генерация программного кода на языках описания аппаратуры или создание схемотехнического решения т.н. «с нуля».

Рассмотрим более подробно все перечисленные варианты решения.

2.7.1. Генерация фильтров с аппаратным решением в пакетах математического моделирования

Первый вариант использования пакетом математического моделирования логично рассмотреть в приложении к наиболее мощному в этом плане программному комплексу математического моделирования – MATLAB. Достоинством этого решения является не только возможность провести моделирование полученного фильтра средствами MATLAB и MATLAB Simulink, рассчитать коэффициенты фильтра с заданными параметрами и характеристиками, но и получить аппаратное решение для ПЛИС в виде кода на языке описания аппаратуры.

Встроенное приложение для расчета цифровых фильтров носит название Filter Design HDL Coder™ и позволяет генерировать эффективный и порти-руемый код на языках описания аппаратуры

VHDL и Verilog HDL. Кроме того, автоматически генерируются модулирующие коды (testbench) для проверки аппаратного решения в среде моделирования ALTERA ModelSim.

Внешний вид модуля в процессе генерации цифрового фильтра приведен на рис. 2.30.

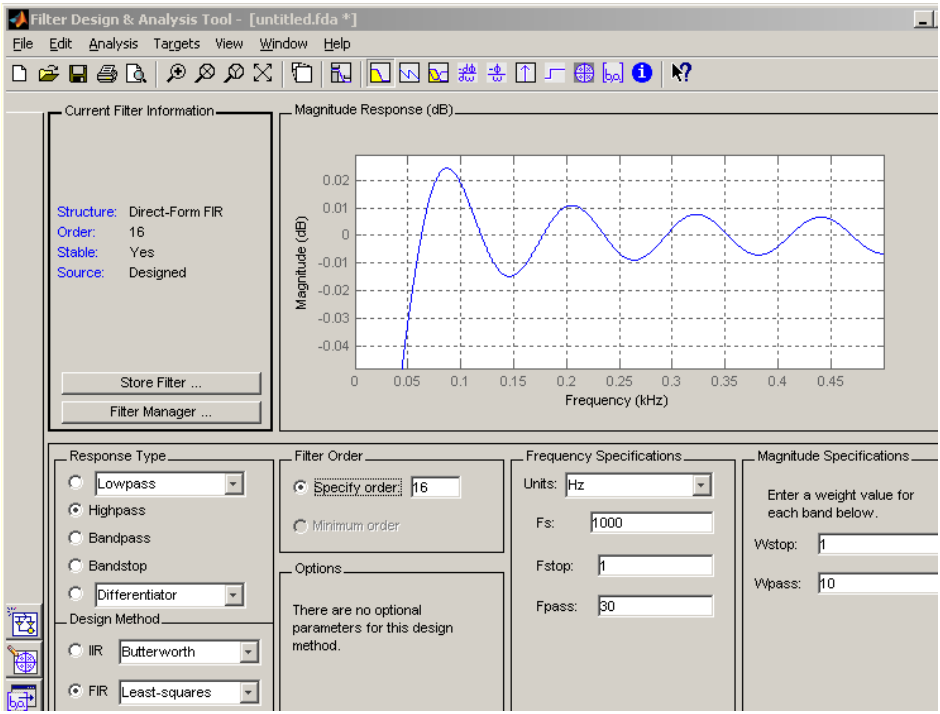


Рис. 2.30. Встроенный модуль MATLAB для автоматизированного расчета параметров цифровых фильтров

Данное решение является достаточно эффективным, но не является так называемым «естественным» для систем на кристалле. Альтернативой ему является использование встроенного в Quartus «нативного» решения FIR Compiler II / FIR Compiler MegaCore Function.

2.7.2. Использование встроенной в САПР Quartus II подпрограммы Altera® FIR Compiler MegaCore

Altera® FIR Compiler MegaCore® позволяет генерировать цифровые фильтры с конечной импульсной характеристикой, аппаратное решение которых без проблем поддерживается производимыми устройствами фирмы ALTERA. Кроме того, при генерации фильтра доступен инструмент IP Toolbench, который позволяет адаптировать структуру фильтра по необходимым критериям, включая требования к параллельному, последовательному и совмещенному решениям, а также к реализациям с фиксированной и плавающей точкой.

Внешний вид основного окна Altera® FIR Compiler показано на рис. 2.31.

Рассмотрим основные свойства FIR Compiler:

1. Обеспечивает полностью интегрированное в САПР Quartus программное решение.
2. Генерирует коэффициенты фильтра, необходимые по заявленным пользователем требованиям.
3. Рассчитывает модели фильтров на языках VHDL, Verilog HDL и MATLAB Script с заявленной битовой разрядностью и в заявленных ограничениях по времени выполнения алгоритма.
4. Автоматически генерирует код и графический символ синте-зированного блока фильтра, которые можно непосредственно использовать в пользовательском проекте.
5. Генерирует вектора данных для тестирования рассчитанного фильтра.
6. Генерирует проверочный код (testbench) на языке VHDL для синтезированного блока фильтра.

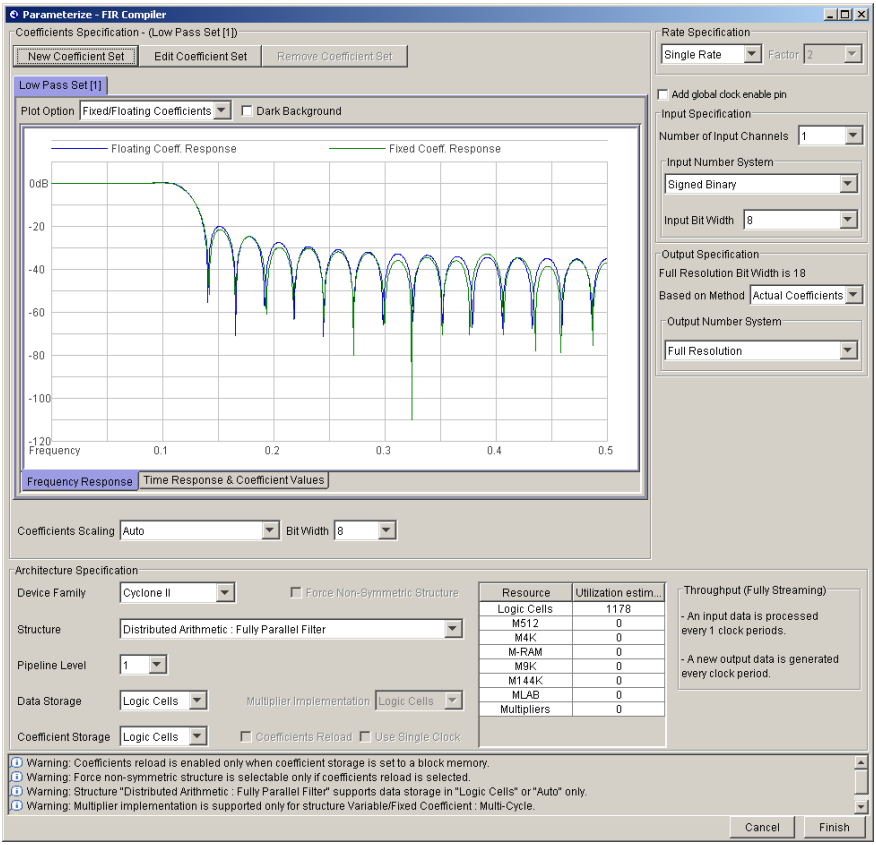


Рис. 2.31. Основное окно FIR Compiler

Окно программы для генерации коэффициентов фильтра показано на рис. 2.32.

В результате работы FIR Compiler получается мегафункция с заданными параметрами, которую можно использовать в проекте.

Условное графическое обозначение сгенерированного решения показано на рис. 2.33.

Основной заголовочный файл приведен в примере 2.8.

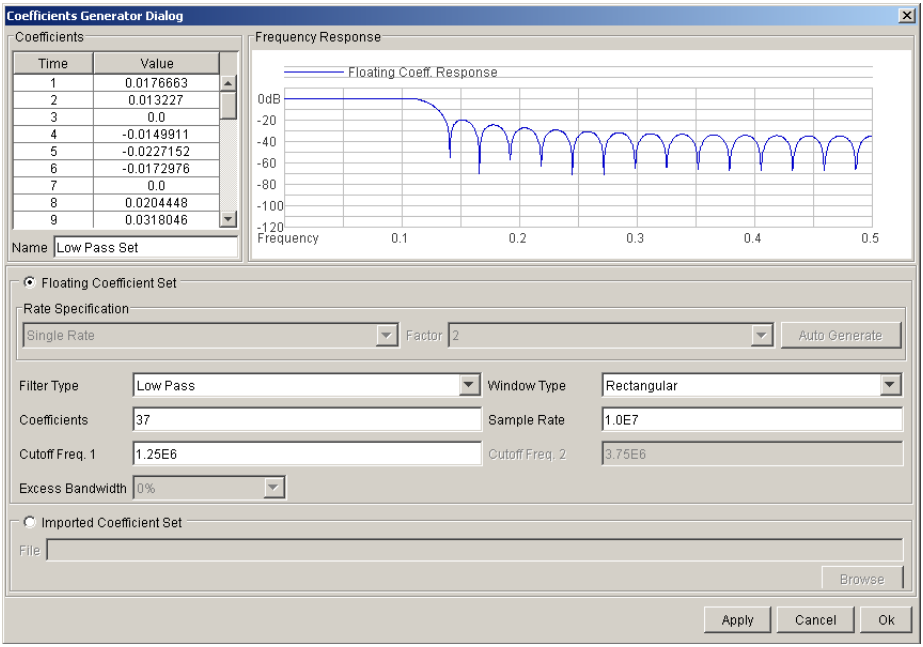


Рис. 2.32. Генерация коэффициентов фильтра

В дополнение к графическому элементу, в проекте генерируется набор заголовочных файлов для подключения к остальным программным компонентам.

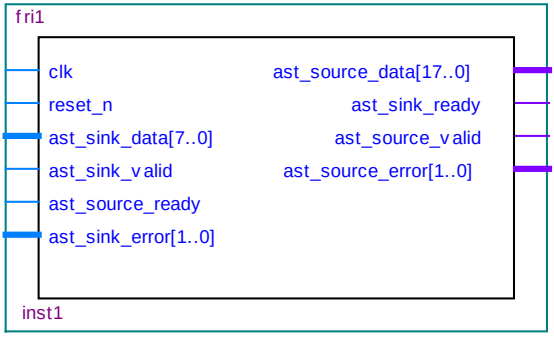


Рис. 2.33. УГО модуля цифрового КИХ-фильтра

Пример 2.8. Заголовочный файл для подключения синтезированного цифрового фильтра.

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY fri1 IS
PORT (
      clk                    : IN STD_LOGIC;
      reset_n                : IN STD_LOGIC;
      ast_sink_data          : IN STD_LOGIC_VECTOR (7
                              DOWNTO 0);
      ast_sink_valid         : IN STD_LOGIC;
      ast_source_ready: IN STD_LOGIC;
      ast_sink_error         : IN STD_LOGIC_VECTOR (1
                              DOWNTO 0);
      ast_source_data        :  OUT  STD_LOGIC_VECTOR
                              (17 DOWNTO 0);
      ast_sink_ready         : OUT STD_LOGIC;
      ast_source_valid       : OUT STD_LOGIC;
      ast_source_error       :  OUT  STD_LOGIC_VECTOR
                              (1 DOWNTO 0)
);
END fri1;

ARCHITECTURE SYN OF fri1 IS

COMPONENT fri1_ast
PORT (
      clk                    : IN STD_LOGIC;
      reset_n                : IN STD_LOGIC;
      ast_sink_data          : IN STD_LOGIC_VECTOR (7
                              DOWNTO 0);
      ast_sink_valid         : IN STD_LOGIC;
      ast_source_ready       : IN STD_LOGIC;
      ast_sink_error         : IN STD_LOGIC_VECTOR (1
                              DOWNTO 0);
      ast_source_data        :  OUT  STD_LOGIC_VECTOR
                              (17 DOWNTO 0);
```

```
      ast_sink_ready        : OUT STD_LOGIC;
      ast_source_valid      : OUT STD_LOGIC;
      ast_source_error      :  OUT  STD_LOGIC_VECTOR
                            (1 DOWNTO 0)
);
END COMPONENT;

BEGIN
fri1_ast_inst : fri1_ast
PORT MAP (
      clk              =>  clk,
      reset_n            =>  reset_n,
      ast_sink_data    =>  ast_sink_data,
      ast_source_data      =>  ast_source_data,
      ast_sink_valid   =>  ast_sink_valid,
      ast_sink_ready   =>  ast_sink_ready,
      ast_source_valid     =>  ast_source_valid,
      ast_source_ready     =>  ast_source_ready,
      ast_sink_error   =>  ast_sink_error,
      ast_source_error     =>  ast_source_error
);
END SYN;
```

Далее рассмотрим основы генерации аппаратных решений для цифровых фильтров средствами САПР Quartus II.

2.7.3. Генерация аппаратного решения средствами Quartus II

Как уже говорилось выше, основной операцией на аппаратном уровне для расчета цифровых фильтров является комплексная операция «умножение с накоплением».

Для ее реализации в идеале необходимы:

1. Память для хранения коэффициентов фильтра.
2. Память для хранения отсчетов входного сигнала.
3. Память для хранения аккумулятора.
4. Память для хранения результата вычисления.
5. Сумматор.

6. Умножитель.

Данные блоки можно реализовать как схемотехнически, так и на языках описания аппаратуры.

Рассмотрим пример реализации цифрового фильтра 4-го порядка на языке описания аппаратуры VHDL [14].

В данном цифровом фильтре используются 4 коэффициента, 4 отсчета входного сигнала и 3 линии задержки. Реализация последовательная. В качестве процедуры обработки данных используется классическое выражение

$$y[n] = \sum_{i=0}^{N} b_i x[n-i] \,.$$

Соединение элементов проводится на программном уровне путем импорта компонента задержки DFF в основной модуль.

Пример 2.9. Заголовочная часть реализации фильтра 4-го порядка.

Основной модуль

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity fir_4tap is
port (
      Clk   : in std_logic;
      Xin   : in signed(7 downto 0);
      Yout : out signed(15 downto 0)
);
end fir_4tap;
```

Полный текст модуля для примера 2.9 приведен в прил. В данного учебного пособия.

На основе данных кодов САПР Quartus II генерирует символ, который можно использовать для построения проекта большего уровня, содержащего цифровые фильтры. Условно-графическое обозначение модуля показано на рис. 2.34.

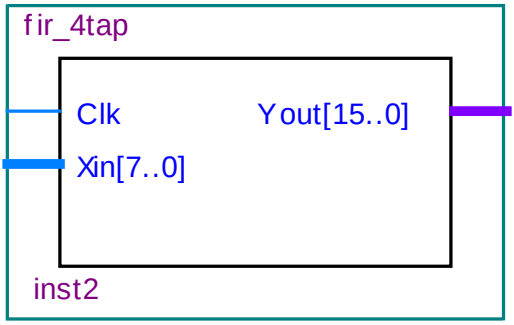


Рис. 2.34. УГО цифрового фильтра 4-го порядка

Заключение

Таким образом, в данной главе были освещены основные свойства, способы описания и представления линейных цифровых фильтров, а также варианты реализации на программном и аппаратном уровне. Кроме того, были рассмотрены базовые примеры и приведен весь необходимый для выполнения соответствующей лабораторной работы теоретический и практический базис.

Исходные данные и задания для выполнения соответствующей лабораторной работы № 1 приведены в прил. А данного учебного пособия.

3. ПРЕОБРАЗОВАНИЕ ФУРЬЕ И СПЕКТРАЛЬНЫЙ АНАЛИЗ

3.1. Основные положения

Данная форма анализа сигналов используется во многих областях науки и техники: в физике, математике, теории чисел, комбинаторике, цифровой обработке сигналов, теории вероятностей, статистике, криптографии, акустике, оптике и т.п. В общем случае, *преобразование Фурье* рассматривается как декомпозиция сигнала на частоты и амплитуды, т.е. обратимы переход из временного пространства в частотное пространство (представление сигнала).

Понятие преобразования Фурье часто используется совместно с понятием *спектра сигнала*. Под спектром будем понимать результат разложения сигнал анна более простые составляющие в базисе ортогональных функций. Базисом разложения в данном конкретном случае является как раз преобразование Фурье, хотя существуют спектры, полученные при разложении по функциям Уолша, Хаара, вейвлетам и т.п.

Разложению в ряд Фурье могут подвергаться периодические сигналы. При этом они фактически представляются в виде суммы гармонических функций или в виде суммы комплексных экспонент с частотами, образующими арифметическую прогрессию. Для того чтобы такое разложение существовало, необходимо соблюдение *условий Дирехле*:

- не должно быть разрывов второго рода;
- число разрывов первого рода (скачков) должно быть ограничено;
- число экстремумов должно быть конечным.

В зависимости от конкретной формы базисной функции и от сферы применения, различают несколько форм записи рядов Фурье [10].

3.2. Формы представления рядов Фурье

3.2.1. Синусно-косинусная форма представления ряда Фурье

В этом случае, ряд Фурье имеет вид

$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(k\varpi_1 t) + b_k \sin(k\varpi_1 t)).$$

Здесь $\varpi_1 = \frac{2\pi}{T}$ – круговая частота, соответствующая периоду повторения сигнала, равному T.

В формулу входят кратные ϖ_1 значения частот $k\varpi_1$, которые называются *гармониками*.

Коэффициенты ряда a_k и b_k считаются по формулам:

$$a_k = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s(t) \cos(k\varpi_1 t)\, dt,$$

$$b_k = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s(t) \sin(k\varpi_1 t)\, dt.$$

Константа a_0 может быть рассчитана по общей формуле для a_k, однако, по своей сути, это слагаемое представляет собой среднее значение сигнала на рассматриваемом периоде:

$$\frac{a_k}{2} = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} s(t)dt.$$

Пределы интегрирования, принятые в данном пособии равными $\left[-\frac{T}{2}; \frac{T}{2}\right]$, не являются обязательными. Интегрирование может проводится по любому интервалу времени, равному *T*, и результат от этого не изменится – интервал выбирается исходя из соображений удобства. Например могут быть использованы пределы от *0* до *T*.

3.2.2. Вещественная форма представления ряда Фурье

Неудобство косинусно-синусной формы представления рядов Фурье состоит в том, что для каждого значения индекса k (т.е. для каждой гармоники $k\varpi_1$), в выражении присутствуют два слагаемых: и косинус и синус.

Вследствие этого, часто преобразуют сумму этих двух слагаемых в косинус той же частоты, но с иной амплитудой и некоторой начальной фазой

$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} A_k \cos(k\varpi_1 t + \varphi_k).$$

3.2.3. Комплексная форма представления ряда Фурье

Комплексная форма представления рядов Фурье является наиболее часто употребляемой в радиотехнике. Она происходит из вещественной формы путем представления косинуса в виде полусуммы комплексных экспонент

$$cosx = \frac{1}{2}\left(e^{jx} + e^{-jx}\right),$$

получаемых из формулы Эйлера:

$$e^{jx} = cosx + jsinx.$$

Применим данное преобразование к вещественной форме ряда Фурье и получим суммы комплексных экспонент с положительными и отрицательными показателями:

$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \frac{A_k}{2} (\exp(\mathrm{j}k\varpi_1 t + j\varphi_k) + \exp(-\mathrm{j}k\varpi_1 t - j\varphi_k)).$$

Если трактовать экспоненты со знаком «минус» в показателе как члены ряда с отрицательными номерами, то можно получить модифицированную комплексную форму записи ряда Фурье:

$$s(t) = \sum_{k=-\infty}^{\infty} \dot{C}_k e^{-jk\varpi_1 t}.$$

Комплексные коэффициенты ряда связаны в этом случае с амплитудами A_k и фазами φ_k, которые используются в вещественной форме записи ряда Фурье и могут быть представлены как

$$\dot{C}_k = \frac{1}{2} A_k e^{j\varphi_k},\ A_k = 2\left|\dot{C}_k\right|,\ \varphi_k = \arg\left(\dot{C}_k\right).$$

Также можно показать связь данной формы представления ряда Фурье с коэффициентами a_k и b_k:

$$\dot{C}_k = \frac{a_k}{2} - \frac{b_k}{2},\ a_k = 2Re\left(\dot{C}_k\right),\ b_k = -2Im\left(\dot{C}_k\right).$$

Отсюда следует формула непосредственного расчета коэффициентов $\dot{C}_k$ ряда в комплексной форме

$$\dot{C}_k = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} s(t)exp(-jk\varpi_1 t)dt.$$

Если сигнал $s(t)$ является четной функцией, то коэффициенты ряда $\dot{C}_k$ будут чисто вещественными, если сигнал $s(t)$ - нечетная функция, то коэффициенты ряда $\dot{C}_k$ являются множеством чисто мнимых значений.

Совокупность амплитуд гармоник ряда Фурье называется *амплитудным спектром*, а совокупность фаз – *фазовым спектром*. Эти понятия отличаются от амплитудной и фазочастотной характеристик сигналов, так как последние относятся к цепям, а не к сигналам.

Так как анализируемые в данной работе сигналы являются вещественными, то амплитудный и фазовый спектры обладают симметрией

$$A_{-k} = -A_k, \quad \varphi_{-k} = -\varphi_k, \quad \dot{C}_{-k} = -\dot{C}_k.$$

3.3. Примеры разложения в ряд Фурье различных сигналов

Рассмотрим в данном пункте примеры разложения сигналов, проанализированных ранее в п. 1.3.

3.3.1. Фурье-преобразование последовательности прямоугольных импульсов

Пример последовательности прямоугольных импульсов показан в п. 1.3.1 на Рис. 1.4.

Данный сигнал является четной функцией, поэтому для его представления можно выбрать синусно-косинусную форму ряда Фурье, так как в ней будут присутствовать только косинусные составляющие ряда a_k, равные:

$$a_k = \frac{2}{T}\int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} A\cos\left(\frac{2\pi k}{T}t\right)dt = \frac{2A}{\pi k}\sin\left(\frac{\pi k\tau}{T}\right).$$

Следует также отметить, что такие параметры, как длительность импульса и период их чередования входят в формулу исключительно в виде отношения. Это отношение называется *скважностью* последовательности импульсов и обозначается как $q = \frac{T}{\tau}$. Аналогом этого понятия является *коэффициент заполнения*, равный $\frac{\tau}{T}$.

При введении этого параметра в формулу коэффициентов ряда Фурье и приведении к виду $\frac{\sin(x)}{x}$:

$$a_k = \frac{2A}{\pi k}\sin\left(\frac{\pi k}{q}\right) = \frac{2A}{q}\frac{\sin\left(\frac{\pi k}{q}\right)}{\frac{\pi k}{q}}.$$

Далее можно записать само представление последовательности прямоугольных импульсов (входного сигнала $s(t)$) в виде ряда Фурье:

$$s(t) = \frac{A}{q} + \sum_{k=1}^{\infty}\frac{2A}{\pi k}\sin\left(\frac{\pi k}{q}\right)\cos\left(\frac{2\pi k}{T}t\right).$$

На рис. 3.1 показаны коэффициенты ряда Фурье для последовательности прямоугольных импульсов.

Амплитуды гармонических слагаемых ряда зависят от номера гармоники по закону $\frac{\sin(x)}{x}$.

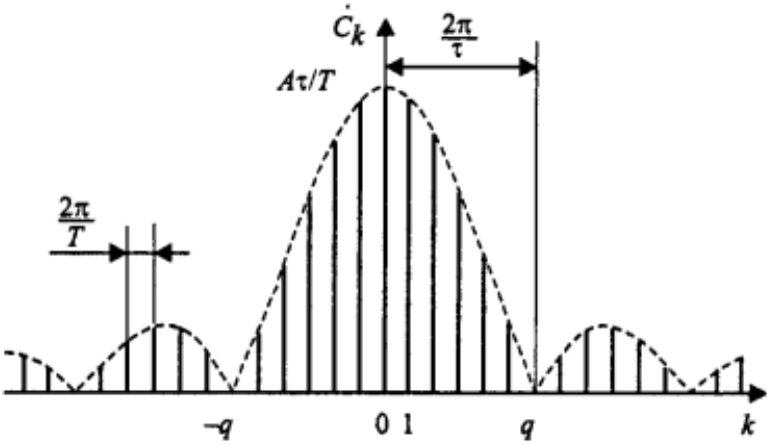


Рис. 3.1. Коэффициенты Фурье последовательности прямоугольных импульсов

Ширина лепестков, измеренная в количестве гармоник, равна скважности последовательности прямоугольных импульсов и обратно пропорциональна длительности импульсов. Причем гармоники с номерами кратными скважности имеют нулевые амплитуды.

Расстояние по частоте между соседними гармониками равно частоте следования импульсов, т.е. $\frac{2\pi}{T}$.

Из этого можно сделать вывод, что, чем короче сигнал – тем шире его спектр.

3.3.2. Фурье-преобразование меандра

Так как меандр является частным случаем последовательности прямоугольных импульсов, показанной в п. 1.3.2, то его спектр получается из спектра последовательности прямоугольных импульсов путем принятия *q=2*.

$$a_k = \frac{A\sin(\pi k/2)}{\pi k/2} = \begin{cases} A, & k = 0, \\ 0, & k = 2m, m \neq 0, \\ \dfrac{2A}{\pi k}, & k = 4m + 1, \\ -\dfrac{2A}{\pi k}, & k = 4m - 1, \end{cases}$$

где m – любое целое число.

В спектре меандра присутствуют только нечетные гармоники.

Меандр может быть сформирован в виде ряда Фурье с учетом выше изложенного:

$$s(t) = \frac{A}{2} + \frac{2A}{\pi}\left(cos\left(\frac{2\pi}{T}t\right) - \frac{1}{3}cos\left(3\frac{2\pi}{T}\ t\right) + \frac{1}{5}cos\left(5\frac{2\pi}{T}t\right) - \cdots\right).$$

Вообще говоря, и последовательность прямоугольных импульсов и меандр плохо подходят для представления рядом Фурье, так как содержат скачки. Поэтому на примыкающих к разрывам участках, сумма ряда Фурье дает пульсации, амплитуда которых не уменьшается с ростом числа суммируемых гармоник. Пульсации лишь сжимаются по горизонтали в приближении к точкам разрыва (см. Рис. 3.2).

Этот эффект присущ для рядом Фурье любых сигналов с разрывами первого рода (скачками) и называется *эффектом Гиббса*.

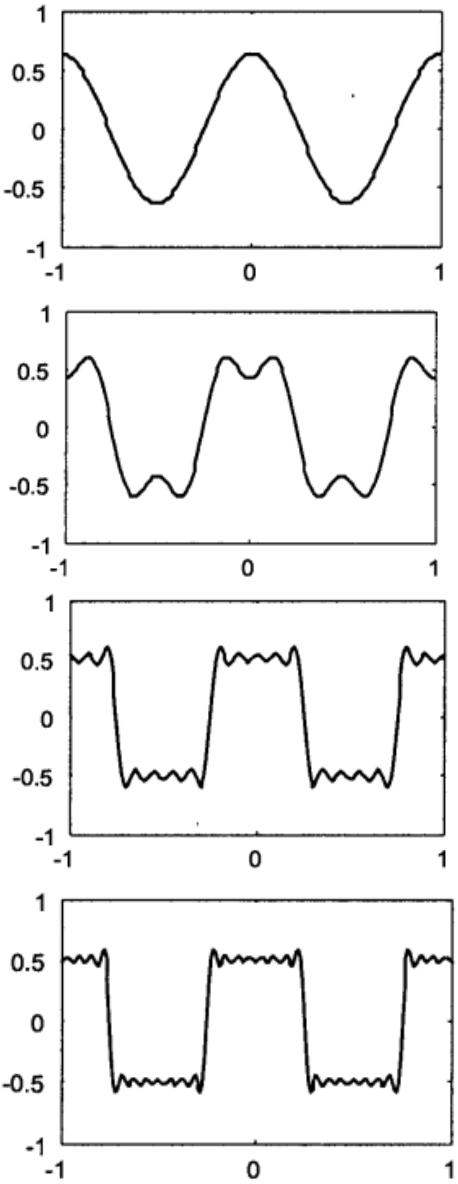


Рис. 3.2. Стадии суммирования ряда Фурье для меандра

3.3.3. Фурье-преобразование пилообразного сигнала

Данный сигнал является нечетной функцией, поэтому его ряд Фурье будет содержать только синусные составляющие

$$b_k = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} \frac{2A}{T} t sin\left(\frac{2\pi k}{T}t\right) dt = -\frac{2A}{\pi k}(-1)^k.$$

Ряд Фурье для такого сигнала имеет следующий вид:

$$s(t) = \frac{2A}{\pi}\left(sin\left(\frac{2\pi}{T}t\right) - \frac{1}{2}sin\left(2\frac{2\pi}{T}t\right) + \frac{1}{3}sin\left(3\frac{2\pi}{T}t\right) - \cdots\right).$$

У рассмотренных ваше спектров сигналов (прямоугольного и пилообразного) есть общая черта – их амплитуды гармоник убывают пропорционально значению параметра *k*.

3.3.4. Фурье-преобразование последовательности треугольных импульсов

Данный сигнал является четной функцией, поэтому его ряд Фурье будет в обязательном порядке содержать только косинусные слагаемые:

$$a_k = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} A\left(1 - 4\frac{|t|}{T}\right) cos\left(\frac{2\pi k}{T}t\right) dt =$$

$$= \frac{4A}{(\pi k)^2}\left(1 - (-1)^k\right) == \begin{cases} 0, & k = 2m, \\ \dfrac{8A}{(\pi k)^2}, & k = 2m+1. \end{cases}$$

Как и в случае меандра, здесь присутствуют только нечетные гармоники. Ряд Фурье, в этом случае, имеет следующий вид:

$$s(t) = \frac{8A}{\pi^2}\left(cos\left(\frac{2\pi}{T}t\right) + \frac{1}{3^2}cos\left(3\frac{2\pi}{T}t\right) + \frac{1}{5^2}cos\left(5\frac{2\pi}{T}t\right) + \cdots\right).$$

В отличие от последовательности прямоугольных импульсов и последовательности пилообразных импульсов, для данного периодического сигнала характерно убывание амплитуды пропорционально второй степени номеров гармоник k. Это свойство следует из того обстоятельства, что скорость убывания спектра зависит от степени гладкости сигнала.

Так как и пилообразный и прямоугольный сигналы имеют множество разрывов первого рода (скачков) – в их спектрах присутствует множитель 1/k. Треугольный же сигнал является непрерывной функцией, хотя ее первая производная содержит разрывы. Амплитуды гармоник его ряда Фурье содержат множитель $1/k^2$.

На основании этого, в литературе [5] получается следующее правило: если N – это номер последней непрерывной производной сигнала, то его спектр будет убывать со скоростью $1/k^{N-2}$. Предельным случаем будет являться гармонический сигнал, дифференцировать который без потери непрерывности можно бесконечное число раз.

3.4. Фурье-преобразование цифровых сигналов

Преобразование Фурье является одним из основных инструментов спектрального анализа непериодических сигналов. К периодическим сигналам оно также применимо, но с некоторыми ограничениями.

Это разложение может быть обобщено и на случай непериодической функции. Действительно, непериодическую функцию можно рассматривать как периодическую, как при неотрицательно возрастающем периоде, т.е. $T \to \infty$. Вернёмся к формуле для $f(t)$ и подставив в неё формулу для C_k, получим

$$X(t) = \frac{1}{T}\sum_{k=\infty}^{\infty} e^{-i2\cdot\pi\cdot\frac{kt}{T}} \int_{-T/2}^{T/2} X(t)\ e^{-i2\pi\frac{kt}{T}} dt\,.$$

Устремим T $\to \infty$. Вместо $\frac{1}{T}$ введем круговую частоту $\omega = 2\pi \cdot \frac{1}{T}$, которая является дискретом наращивания частоты. Тогда при T $\to \infty$ получим $\omega = 2\pi \cdot \frac{1}{T} \to 0$, $\omega \to d\omega$.

Сумма переходит в интеграл и

$$X(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega t} d\omega \int_{-\infty}^{\infty} X(t) e^{-i\omega t} dt.$$

Обозначим через $S(\omega)$ второй интеграл в выражении (2.43), тогда

$$S(\omega) = \int_{-\infty}^{\infty} X(t) e^{-i\omega t} dt,$$

и

$$X(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega t} S(\omega) d\omega \cdot$$

Преобразование задается следующим образом:

$$X(\omega) = \int_{-\infty}^{\infty} x(t) \cdot e^{-2\pi i \omega t} dt,$$

где x(t) – реальный сигнал, наблюдаемый во временном интервале, а X(ω) – его отображение в частотном интервале.

Соответственно обратное преобразование задаётся формулой

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega)\ e^{i2\pi\omega t} d\omega .$$

При цифровой обработке входной сигнал представляет собой дискретную последовательность отсчетов. Нельзя говорить о преобразовании Фурье от последовательности отсчетов. Тем не менее, можно связать дискретную последовательность чисел *X(nT)* с временной функцией

$$X^*(t) = \sum_{-\infty}^{\infty} x(nT)\ \delta(xt - nT).$$

где δ – единичная функция.

В этом случае можно воспользоваться преобразованием Фурье функции $X^*(t)$.

Дискретным аналогом метода Фурье являются следующие формулы:

$$X(j) = \sum_{k=0}^{N-1} x(k) e^{-i2\pi\frac{jk}{N}},$$

$$x(k) = \frac{1}{N} \sum_{j=0}^{N-1} X(j) e^{i2\pi\frac{jk}{N}},$$

при *j = 0,1,2, … , N – 1, k = 0,1,2, … , N – 1.*

На практике обычно пользуются формулами, в которых экспоненциальный член записывается в виде W_N, т.е. $W_N = e^{-i2\frac{\pi}{N}}$, тогда

$$X(j) = \sum_{k+0}^{N-1} x(k) W_N^{jk} .$$

Общее число операций равно *N2* комплексных сложений и умножений. Даже при умеренно большом числе точек (*N = 1024*) общее число операций оказывается неприемлемым. Задача быстрого преобразования Фурье (БПФ) заключается в сокращении числа выполняемых операций. Одним из приемов такого сокращения является разделение исходной последовательности на несколько подмножеств.

Пример 3.1. Вычисление дискретного преобразования Фурье.

Пусть задана дискретная последовательность из четырёх точек x(nt) = ={1,1,0,0}.

Прямое 4-хточечное дискретное преобразование Фурье:

$$X(i) = \sum_{j=0}^{3} x(j) e^{-i2\pi\frac{jt}{4}}.$$

Обратное 4-х точечное дискретное преобразование Фурье.

$$x(k) = \frac{1}{4}\sum_{j=0}^{3} x(j) e^{-i2\pi\frac{jk}{4}}.$$

Определяем выходную последовательность:

X(0) = x(0) + x(1) + x(2) + x(3) =2;

$$X(1) = x(0) + x(1)\, e^{-i2\pi\frac{1}{4}} + x(2)\, e^{-i2\pi\frac{2}{4}} + x(3)\, e^{-i2\pi\frac{3}{4}} = 1 - i;$$

$$X(2) = x(0) + x(1)\, e^{-i2\pi\frac{2}{4}} + x(2)\, e^{-i2\pi\frac{4}{4}} + x(3)\, e^{-i2\pi\frac{6}{4}} = 0;$$

$$X(3) = x(0) + x(1)\, e^{-i2\pi\frac{3}{4}} + x(2)\, e^{-i2\pi\frac{6}{4}} + x(3)\, e^{-i2\pi\frac{9}{4}} = 1 - i.$$

Проведем обратное преобразование и найдем исходную последовательность:

$x(0) = ¼\,(X(0) + X(1) + X(2) + X(3)) =$

$= ¼\,(2 + 1 - i + 0 + 1 + i) = 1;$

$$x(1) = ¼\left(X(0) + X(1)\,e^{-i2\pi 1/4} + X(2)\,e^{-i2\pi 2/4} + X(3)\,e^{-i2\pi 3/4}\right) =$$

$$= ¼\cdot\left(2 + (1 - i)i + 0 + (i + 1)\cdot(-i)\right) = 1;$$

$$x(2) = ¼\left(X(0) + X(1)\,e^{-i2\pi 2/4} + X(2)\,e^{-i2\pi 4/4} + X(3)\,e^{-i2\pi 6/4}\right) =$$

$= ¼\,(2 + (1 - i)(-1) + 0 + (i + 1)\cdot(-1)) = 0.$

$$x(3) = ¼\left(X(0) + X(1)\,e^{-i2\pi 3/4} + X(2)\,e^{-i2\pi 6/4} + X(3)\,e^{-i2\pi 9/4}\right) =$$

$= ¼\,(2 + (1 - i)\cdot(-i) + 0 + (i + 1)\,i) = 0.$

3.5. Дискретное преобразование Фурье с прореживанием по времени

Запишем выражение ДПФ в виде:

$$A(j) = \sum_{k-0}^{N-1} a(k) W_N^{jk} .$$

Разобьем последовательность *a(0), a(1), a(2), ..., a(N – 1)* на две. При записи для упрощения будем писать a_0 вместо *a(0)*, a_1 вместо *a(1)* и т.д.

$$x_1(k) = a_0; a_2; a_4 ...; a_{N/2};$$
$$x_2(k) = a_1; a_3; a_5 ...; a_{N/2-1}.$$

В этом случае *N*-точечное ДПФ можно записать как:

$$A(j) = \underbrace{\sum_{k=0}^{N-1} x_1(k) W_N^{jk}}_{k-четное} + \underbrace{\sum_{k=0}^{N-1} x_2(k) W_N^{nk}}_{k-нечетное} = \sum_{k=0}^{\frac{N}{2}-1} a(2k) W_N^{2kj} + \sum_{k=0}^{\frac{N}{2}} a(2k+1) W_N^{(2k+1)j}.$$

С учетом того, что

$$W_N^{\,2} = \left[e^{-i(\frac{2}{\pi})} \right]^{-2} = e^{-i\left(\frac{2\pi}{N/2}\right)} = W_{N/2},$$

перепишем выражение в виде

$$A(j)=\sum_{k=0}^{\frac{N}{2}-1}x_1(k)W_{N/2}^{jk}+W_{N/2}{}^{j}\sum_{k=0}^{\frac{N}{2}-1}x_2(k)W_{N/2}^{jk};$$

$$A(j)=A_1(j)+W_N{}^{i}A_2(j),$$

где *A1(j)* и *A2(j) – N/2*-точечные ДПФ.

Поскольку *A(j)* определено на *N* точках, а $A_1(j)$ и $A2(j)$ определены на *N/2* точках, то необходимо доопределить *A(j)* до *N* точек. Это доопределение достаточно очевидно:

при $0\le j\le\frac{N}{2}-1;\ A(j)=A_1(j)+W_N{}^{j}A_2(j),$

при $\frac{N}{2}\le j\le N-1;$

$$A(j)=A_1\left(j-\frac{N}{2}\right)-W_N{}^{j}A_2\left(j-\frac{N}{2}\right).$$

Алгоритм выполнения БПФ сформулируем в форме направленного графа для *N=8*. Первое разбиение ДПФ показано на рис. 3.3.

На рис. 3.3 показано, что полное ДПФ разбито на два ДПФ меньшего размера. В общем случае это два *N/2*-точечных ДПФ, в нашем случае это два 4-точечных ДПФ.

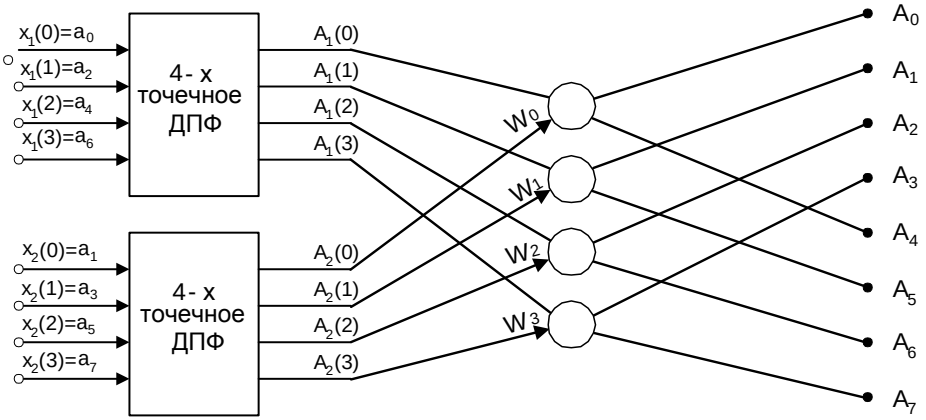
Вершина графа обозначена кружком.

В вершине выполняется две операции.

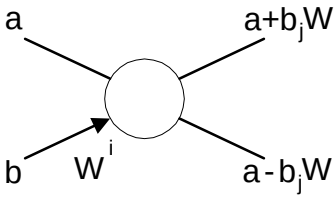
Результат первой операции формирует значение на верхнем выходе вершины графа по следующему правилу:

$$A_0=A_1(0)+A_2(0)W_N^0.$$

Или, в соответствии с принятыми обозначениями, N внутри одного и того же графа опускается, что и сделано на рис. 3.3.



а



б

Рис. 3.3. Дискретное преобразование Фурье

Соответственно для нижнего выхода имеем

$$A_4 = A_1(0) + (A_2(0)W_N^0)W_N^{W/2}.$$

Учитывая, что $W_N^{N/2} = e^{\frac{-i2\pi k\frac{N}{2}}{N}} = -1$, запишем

$$A_4 = A_1(0) - A_2(0)W_N^0,$$

т.е. значение выходного данного формируется как сумма значения, поступающего на верхний вход, и значения, поступающего на нижний

вход, умноженного на значение коэффициента *Wi*, указанного над соответствующим входом.

На операцию умножения указывает стрелка на линии графа. Полученное значение поступает на первый выход вершины графа. На второй выход поступает та же сумма, но второй член суммы умножается на $W_N^{N/2} = -1$.

Рассмотренная схема может быть использована для расчета N/2-точечных ДПФ. В этом случае число умножений и сложений равно ((N/2)2 +N). При (N/2)2 >> N выигрыш составляет около 50 %.

Процесс разбиения может быть продолжен и N/2-точечную ДПФ можно вычислить как две N/4-точечные ДПФ:

$$A_1(j) = B_1(j) + W_{\frac{N}{2}}{}^{j} B_2(j),$$

или

$$A_1(j) = B_1(j) + W_N{}^{2j} B_2(j).$$

Здесь

$$A_1(j) = \sum_{k=0}^{N/2-1} x_1(k) W_{N/2}^{jk},$$

$$A_2(j) = \sum_{k=0}^{N/2-1} x_2(k) W_{N/2}^{jk},$$

где $x_1(k) = a(2k), \;\; x_2(k) = a(2k+1)$.

Тогда имеем

$$A_1(j) = \sum_{k=0}^{N/4-1} x_1(2k) W_{N/2}^{2jk} + \sum_{k=0}^{N/4-1} x_1(2k-1) W_{N/2}^{(2k+1)j}.$$

Окончательно получим

$$A_1(j) = \underbrace{\sum_{k=0}^{N/4-1} x_{11}(k) W_{N/4}^{kj}}_{B_1} + W_N^{2j} \underbrace{\sum_{k=0}^{N/4-1} x_{12}(k) W_{N/4}^{kj}}_{B_2}.$$

Аналогично для $A_2(j)$

$$A_2(j) = \underbrace{\sum_{k=0}^{N/4-1} x_{21}(k) W_{N/4}^{kj}}_{C_1} + W_N^{2j} \underbrace{\sum_{k=0}^{N/4-1} x_{22}(k) W_{N/4}^{kj}}_{C_2}.$$

Процесс уменьшения размерности ДПФ продолжается до тех пор, пока не останутся лишь 2-точечные ДПФ.

Двухточечные ДПФ не содержат умножения. Действительно,

$$F(0) = f(0) + W_N^0 f(1),$$
$$F(1) = f(0) + W_N^{N/2} f(1).$$

Так как $W_N^0 = 1$, а $W_N^{N/2} = e^{-i2\pi\frac{N}{2N}} = -1$, то

$$\left.\begin{aligned} F(0) &= f(0) + f(1), \\ F(1) &= f(0) - f(1). \end{aligned}\right\},$$

Учитывая, что 2-точечные БПФ вычисляются, используя только операции сложения и вычитания входных аргументов, их реализация достаточно проста. На рис. 3.4 показаны различные формы представления их на графах.

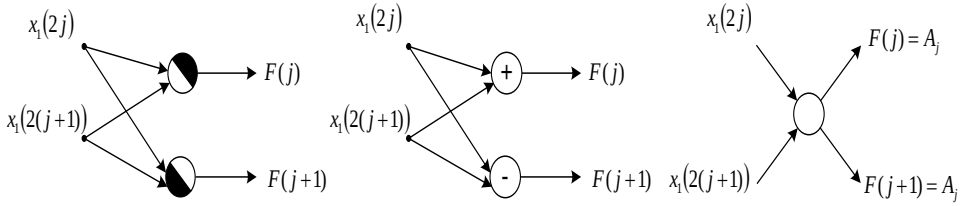


Рис. 3.4. Дискретное преобразование Фурье

С учетом формы реализации 2-точечных БПФ общая граф-схема 8-точеч-ного БПФ будет иметь вид, показанный на рис. 3.5. Здесь все вершины графа имеют одну и ту же структуру.

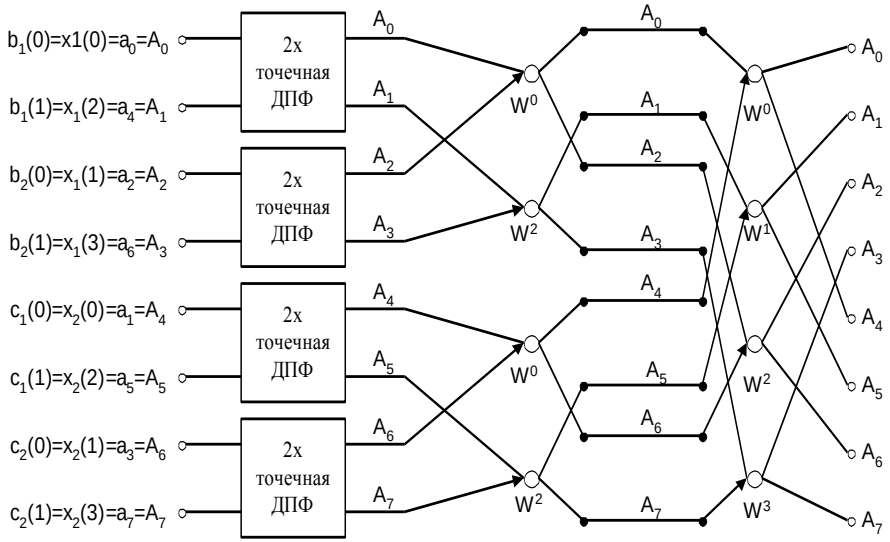


Рис. 3.5. ДПФ на основе 2-точечных преобразований Фурье

Освобождаясь от 2-точечных ДПФ, получим граф приведенный на рис. 3.6.

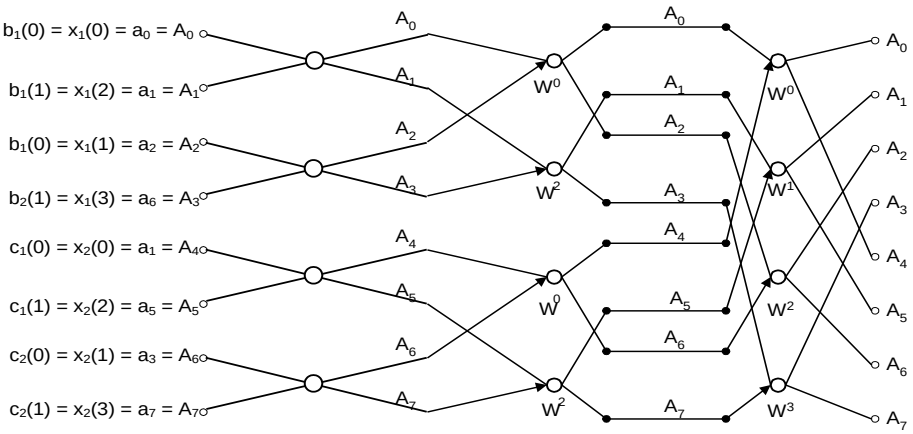


Рис. 3.6. ДПФ на основе 1-точечных преобразований Фурье

Рассмотренный метод построения БПФ носит название «прореживание во времени». Сигнальный граф может быть представлен и в других формах. Основной операцией на сигнальном графе является «бабочка». Это вершина графа с двумя входящими и двумя выходящими переменными. Каждую вершину можно разделить на две. Тогда основной операцией будет «полубабочка». Соответственно получим следующий направленный сигнальный граф (рис. 3.7).

Для реализации сигнального графа, изображенного на рис. 3.7, необходимо предварительно осуществить перестановку входной последовательности данных.

Можно построить несколько другой граф, в котором не изменяется порядок следования входных данных. Однако выходная последовательность формируется в двоично-инверсной форме. Возможно построение графа с сохранением естественного порядка следования входной и выходной последовательности данных.

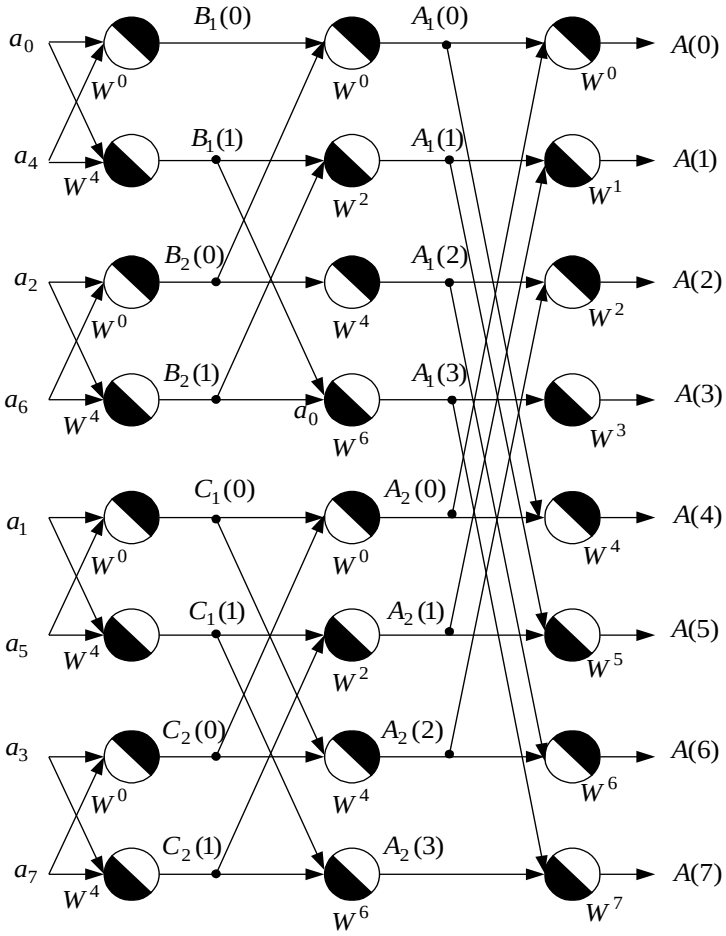


Рис. 3.7. Сигнальный граф ДПФ

3.6. Дискретное преобразование Фурье с прореживанием по частоте

Существует другая форма алгоритма быстрого преобразования Фурье. Его идея основана на прямом разделении массива входных данных. Пусть задана N-точечная последовательность (N=2m) $a\ell$. Разделим ее на две части по N/2 точек. Формально можно записать

$$\left.\begin{aligned} f_\ell &= a_\ell, \\ h_\ell &= a_{\ell+N/2} \end{aligned}\right\} \quad для \;\; \ell = 0,\ 1,\ \ldots\ ,\ N/2 - 1.$$

В этом случае N-точечное БПФ может быть записано через переменные f_ℓ, h_ℓ:

$$A(k) = \sum_{\ell=0}^{N/2-1} (f_\ell W^{\ell k} + h_\ell W^{(\ell+N/2)k}).$$

Учитывая, что $W^{N/2\,k} = e^{-i2\pi\frac{Nk}{2N}} = e^{-i\pi k}$, получим

$$A(k) = \sum_{\ell=0}^{N/2-1} (f_\ell + e^{-i\pi\ k} h_\ell) W^{\ell k}.$$

Разобьем суммирование так, чтобы одно определяло частоты с четным k, а второе – с нечетными k, т.е. проведем прореживание по частоте. Тогда

$$A(2k) = \sum_{\ell=0}^{N/2-1} f_\ell W^{\ell 2k} + e^{-2\pi\ k} h_\ell W^{\ell 2k}.$$

Проведя несложные преобразования, получим

$$A_{2k} = \sum_{\ell=0}^{N/2-1} (f_\ell + h_\ell) W_{N/2}^{\ell k},$$

Далее запишем

$$A(2k+1) = \sum_{\ell=0}^{N/2-1} (f_\ell + e^{-i\pi(2k+1)} h_\ell) W^{\ell(2k+1)}$$

и

$$A(2k+1) = \sum_{\ell=0}^{N/2-1} [(f_\ell - h_\ell)W^\ell]\, W_{N/2}^{\ell k},$$

Приведенные выражения и представляют собой N/2-точечные ДПФ от функций $(f_\ell + h_\ell)$ и $(f_\ell W^\ell - h_\ell W^\ell)$.

Сигнальный граф, показывающий путь вычисления N-точечного ДПФ через два N/2-точечных ДПФ, показан на рис. 3.8. при N = 8.

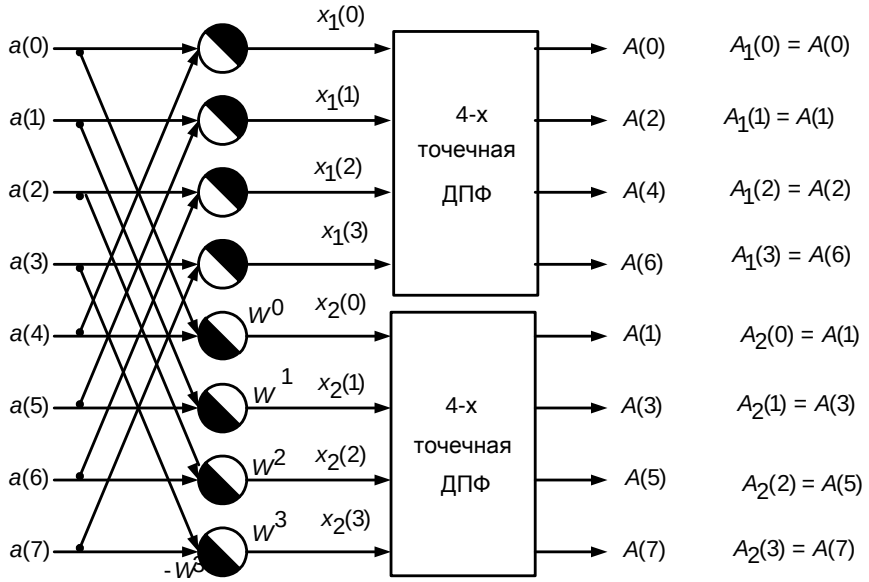


Рис. 3.8. Сигнальный граф на 2-точечных ДПФ

Используя обозначения Рис. 3.8, можно записать

$$\left.\begin{aligned} A_1(\ell) &= \sum_{k=0}^{N/2-1} x_1(k) W_{N/2}^{\ell k}, \\ A_2(\ell) &= \sum_{k=0}^{N/2-1} x_2(k) W_{N/2}^{\ell k}. \end{aligned}\right\} \qquad (\ell = 0,\ 1,\ \ldots\ ,\ (N/2)-1)$$

Теперь применим к каждой N/2-точечной ДПФ редукцию, предложенную выше.

Результат этой редукции показан на рис. 3.9.

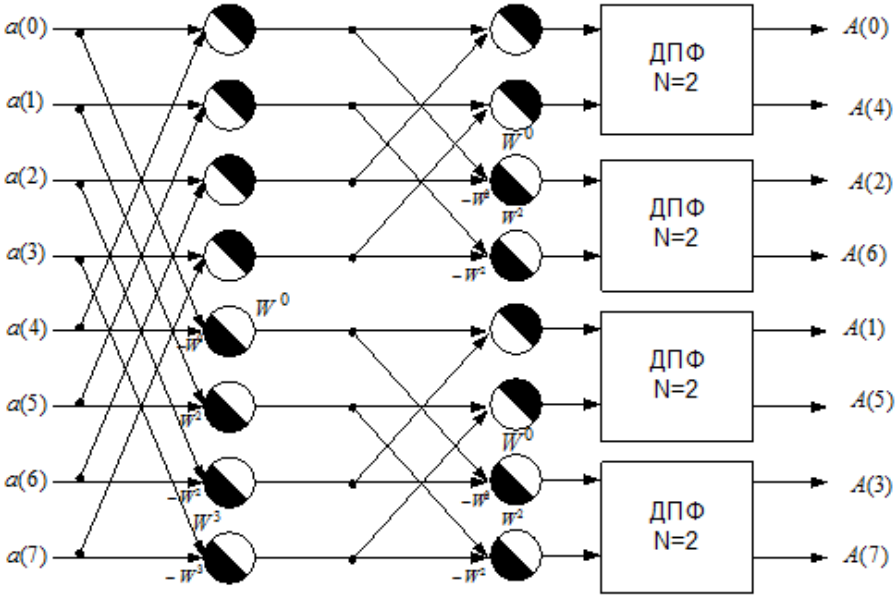


Рис. 3.9. Редуцированный сигнальный граф

Здесь во второй строке значение $W'_{N/2}$ заменено на $W_N^2 = W^2$.

2-точечное ДПФ, сведенное к комплексному сложению/вычитанию, имеет тот же вид, что и полученное ранее.

На рис. 3.10 показана диаграмма вычислений, сведенная к комплексным сложениям и умножениям, пропорциональным $N \log_2 N$, т.е. эквивалентная БПФ.

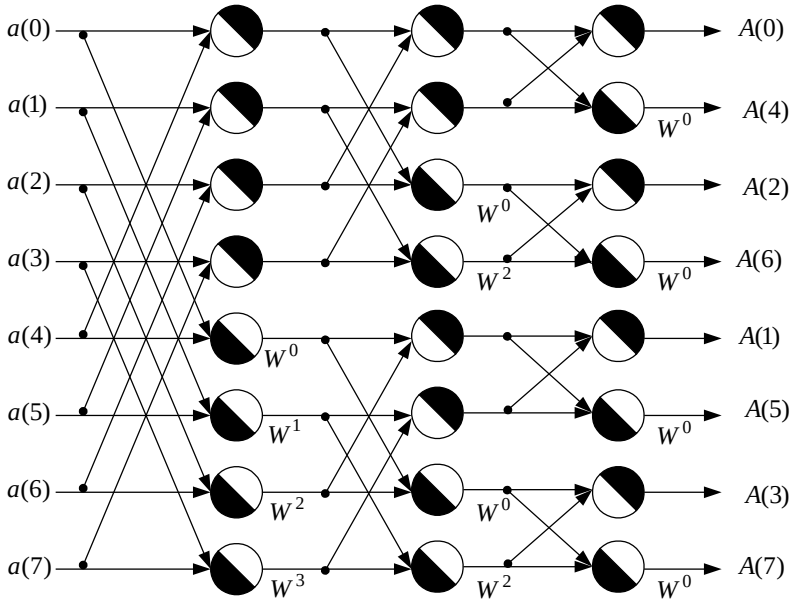


Рис. 3.10. Развернутый сигнальный граф

На графе приняты те же обозначения, что и ранее. В вершинах, затемненных сверху, осуществляется суммирование входных данных, затемненных снизу, из верхнего значения вычитается нижнее. Стрелка указывает на операцию умножения переменной, передаваемой по ребру графа, на переменную, указанную рядом. Точка указывает на размножение переменной.

Недостатком вычислений по графу на рис. 3.10 является необходимость считывания сразу массива чисел и, как следствие, необходимость некоторого числа дополнительных регистров. Если обсчет начинается с верхней вершины, то минимально необходимо восемь регистров. В связи с чем перестроим граф так, чтобы при последовательном обсчете вершин были готовы входные данные при минимуме регистров. Преобразованный граф показан на рис. 3.11.

Сигнальные направленные графы по методу «прореживания во времени» и методу «прореживания по частоте» по структуре одинаковы и отличаются лишь местом расположения коэффициентов W в операции умножения. В структуре БПФ «прореживание по

частоте» умножение производится после выполнения операции сложения/вычитания.

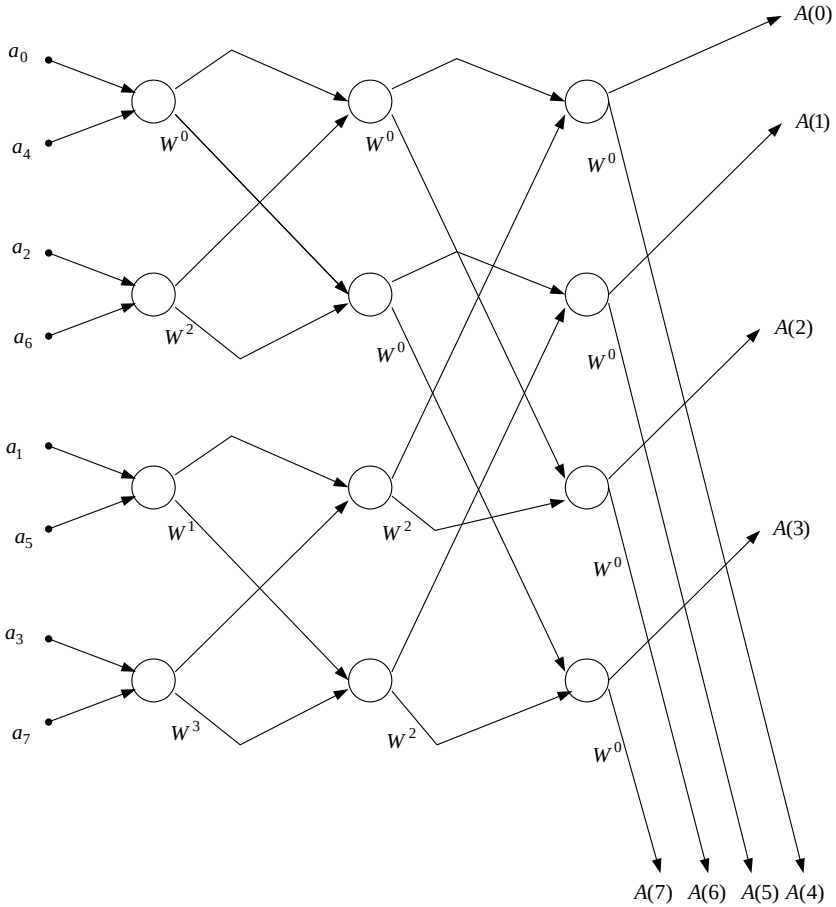


Рис. 3.11. Завершающий преобразованный граф БПФ

3.7. Обратное дискретное преобразование Фурье цифровых сигналов

Обратное дискретное преобразование Фурье задаётся следующим образом:

$$x(\mathrm{n}) = \frac{1}{N}\sum_{j=0}^{N-1} X(j) e^{i2\pi\frac{jn}{N}}.$$

Так как исходные переменные получаются на основе быстрого преобразования Фурье (БПФ), то исходными переменными являются *A(j)*. Тогда

$$x(\mathrm{n}) = \frac{1}{N}\sum_{j=0}^{N-1} A(j) e^{i2\pi\frac{jn}{N}},$$

Запишем прямое преобразование $\mathrm{X}(j) = \sum x(k) \cdot e^{\frac{-i2\pi \cdot jn}{N}}$, так как размерность последовательности *x(n)* и *X(j)* совпадают и равно $A(j)$, то можно воспользоваться алгоритмом получения быстрого преобразования Фурье и для получения быстрого обратного преобразования Фурье (ОБПФ). Единственное отличие заключается в том, что аргумент экспоненциального члена (*W*) в ОБПФ берется с противоположным знаком, т. е. со знаком плюс.

Соответственно при формировании быстрых алгоритмов обратного преобразования Фурье используются те же приёмы, которые рассмотрены выше для прямого преобразования.

Пример 3.2. Вычисление Фурье преобразования.

Для заданной дискретной последовательности

$$\mathrm{x(n)} = \{0, 1, 0, -1\}.\ T = \frac{\pi}{2} \cdot n,\ \mathrm{T} = \{0,\ \frac{\pi}{2},\ \pi,\ \frac{3}{2}\pi\ \},$$

необходимо найти Фурье образ заданной дискретной последовательности.

$$X(j) = \sum_{k=0}^{3} x(n) e^{-i2\pi\frac{kn}{4}};$$

X(0) = 0 + 1 + 0 – 1 = 0.

$X(1) = 0 + 1 \cdot e^{-i2\pi\frac{1}{4}} + 0 - e^{-i2\pi 3/4} = -i - e^{-i2\pi 1/4} \cdot e^{-i2\pi 2/4} =$

$= -i + e^{-i\pi/2} = -2i.$

$X(2) = 0 + 1 \cdot e^{-i2\pi 2/4} + 0 - e^{-i2\pi 6/4} = e^{-i\pi} - e^{-i2\pi 4/4} =$

$= -i + i = 0.$

$X(3) = 0 + 1 \cdot e^{-i2\pi 3/4} + 0 - e^{-i2\pi 9/4} = e^{-i2\pi 3/4} \cdot e^{-i2\pi 5/4} =$

$= -\left(e^{-i\pi 1/2} + e^{-i\pi 1/2}\right) = 2i.$

На рис. 3.12 показан граф 4-точечного БПФ для заданной задачи.

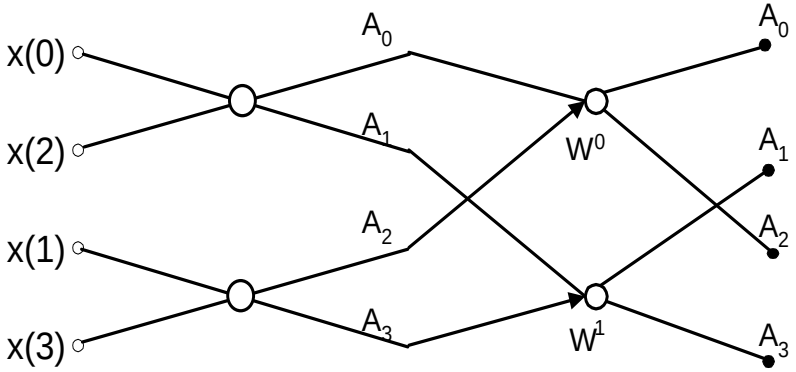


Рис. 3.12. Сигнальный граф для Примера 3.2

Выполняя все действия, предписываемые графом БПФ, получим

$A'(0) = x0 + x2\, W_4^0 = (x0 + x2),$

A(0) = (x0 + x2) + (x1 + x3) w_4^0 ,

A'(1) = x0 – x2 w_4^0 = (x0 – x2),

A(1) = (x0 + x2) – (x1 – x3) w'_4 ,

A'(2) = x1 + x3 w_4^4 = (x1 + x3),

A(2) = (x0 – x2) – (x1 + x3) w_4^0 ,

A'(3) = x1 – x3 w^4 = (x1 – x3),

A(3) = (x0 – x2) + (x1 – x3) w'_4 ,

A(0) = 0,

A(1) = $2 \cdot e^{-i 2\pi \frac{1}{4}}$ = – 2i.

A(2) =0,

A(3) = $-2 \cdot e^{-i 2\pi \frac{1}{4}}$ = 2i.

Пример 3.3. Вычисление обратного Фурье преобразования.

Найти обратное преобразование для заданной в примере 3.3 после-довательности;

$$A(j) = \{0; -2i; 0; 2i\}.$$

Для x(0) имеем

$$x(0) = ¼\left(A(0) + A(1) + A(2) + A(3)\right) = ¼\,(0 - 2i + 0 + 2i) = 0.$$

Для x(1):

$$x(1) = ¼\left(A(0) + A(1)\cdot e^{i2\pi 1/4} + A(2)\cdot e^{i2\pi 3/4} + A(3)\cdot e^{i2\pi 2/4}\right) =$$

$$¼\left(0 + (-2i)\cdot i + 0 + (-1)\, e^{i2\pi 1/4}\cdot e^{i2\pi 2/4}\right) = 1.$$

Для x(2):

$$x(2) = ¼\left(A(0) + A(1)\cdot e^{i2\pi 2/4} + A(2)\cdot e^{i2\pi (2/4)\cdot 2} + A(3)\cdot e^{i2\pi (3/4)\cdot 2}\right) =$$

$$¼\left(0 + (-2i)\cdot e^{i\pi} + 0 + (2i)\cdot e^{i\pi}\right) = 0.$$

Для x(3):

$$x(3) = ¼\left(A(0) + A(1)\cdot e^{i2\pi 3/4} + A(2)\cdot e^{i2\pi (3/4)\cdot 2} + A(3)\cdot e^{i2\pi (3/4)\cdot 3}\right) =$$

$$= ¼\left(0 + (-2i)\cdot e^{i\pi}\cdot e^{i\pi/2} + 0 + (2i)\cdot e^{i4\pi}\cdot e^{i\pi/2}\right) = -1.$$

3.8. Аналитический расчет и моделирование цифрового преобразования Фурье

Для выполнения преобразования Фурье цифровых сигналов в аналитической форме можно воспользоваться одной из существующих сред математического моделирования и выполнения математических

расчетов. Примером такого программного комплекса может служить MathCad.

Выполним дискретное преобразование модельного сигнала при помощи MathCad.

Сгенерируем тестовый сигнал как сумму из трех синусоид:

$$N := 128,$$
$$xMAX := 100,$$
$$\Delta := \frac{xMAX}{N},$$
$$i := 0..N-1,$$
$$x_i := i * \Delta,$$
$$y_i := \sin(2\pi\ 0{,}05x_i) + 0{,}5\sin(2\pi\ 0{,}1x_i) + 0{,}25\sin(2\pi\ 0{,}4x_i).$$

График построенного сигнала представлен на рис. 3.13.

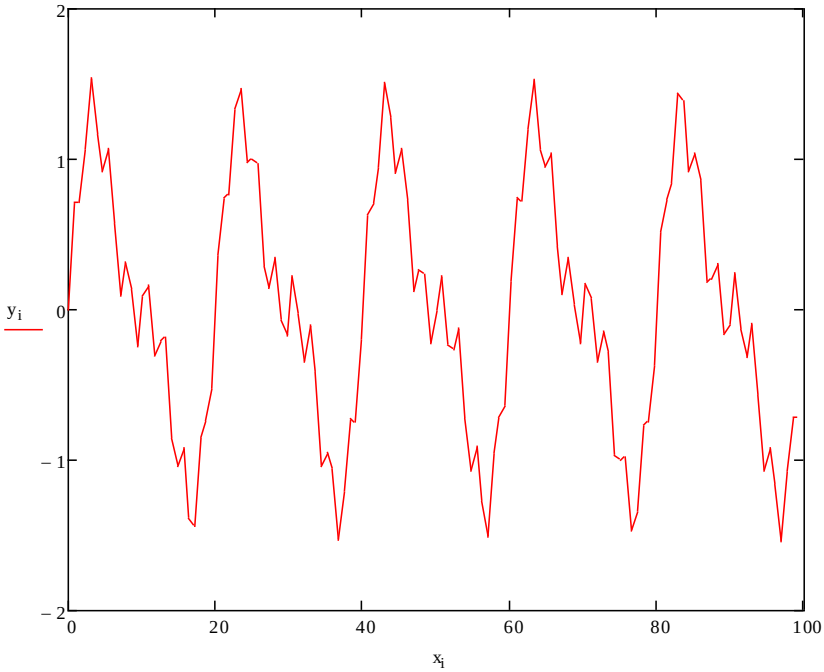


Рис. 3.13. Модельный сигнал

Далее выполним преобразование Фурье модельного сигнала встроенными средствами MathCad:

$$F := fft(y),\ \Omega_i := \frac{(i+1)}{xMAX}.$$

Далее можно получить график модуля полученных коэффициентов Фурье для оценки степени вклада каждой гармоники в формирование модельного сигнала.

Модуль коэффициентов Фурье представлен на рис. 3.14.

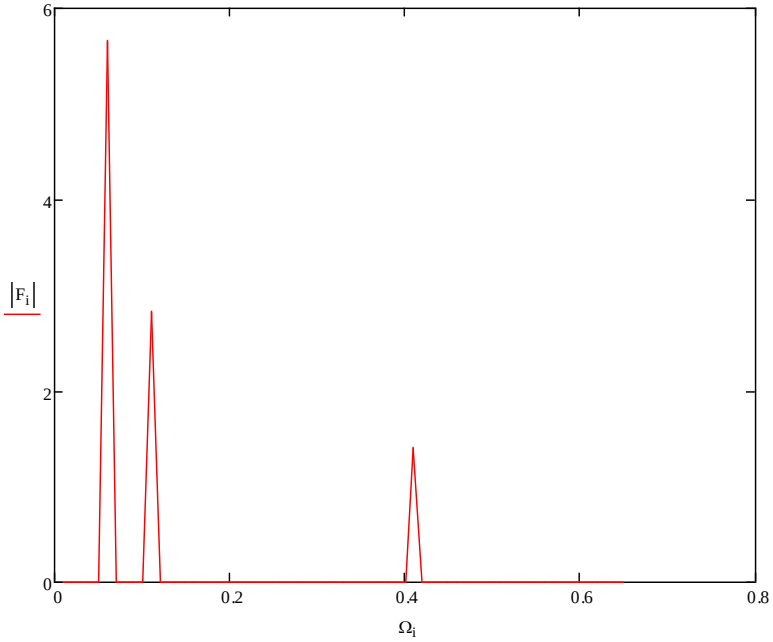


Рис. 3.14. Преобразование Фурье для модельного сигнала

По данному графику можно сделать вывод о том, что основной гармоникой в сигнале является первая, имеющая наибольшую амплитуду и наименьшую частоту.

Далее рассмотрим программную реализацию цифрового преобразования Фурье.

3.9. Программная реализация преобразования Фурье

Рассмотрим конкретную реализацию быстрого преобразования Фурье. Программный код преобразования Фурье, реализованный на языке Си и приведенный в [16], может быть легко портирован практически на любую архитектуру и запущен под любой операционной системой. Единственным ограничением тут является необходимость запуска для цифрового сигнала с количеством точек, равным степень двух.

Прежде всего необходимо разбить последовательность отсчетов входного сигнала на две последовательности: четные и нечетные. Далее точно также необходимо поступить и с каждым полученным множеством значений. Этот итерационный процесс повторяется до тех пор, пока не будет достигнута минимальная длина последовательности, равная двум.

На рис. 3.15 показан пример перестановки для последовательности в 16 элементов.

Итого выполняется $(log_2 N) - 1$ операций.

Рассмотрим двоичное представление номеров элементов и занимаемых ими мест. Элемент с номером 0 (двоичное 0000) после всех перестановок занимает позицию 0 (0000), элемент 8 (1000) – позицию 1 (0001), элемент 4 (0100) – позицию 2 (0010), элемент 12 (1100) – позицию 3 (0011). И так далее. Нетрудно заметить связь между двоичным представлением позиции до перестановок и после всех перестановок: они зеркально симметричны. Двоичное представление конечной позиции получается из двоичного представления начальной позиции перестановкой битов в обратном порядке. И наоборот.

$$x_0\,x_1 \quad x_2\,x_3 \quad x_4\,x_5 \quad x_6\,x_7 \quad x_8\,x_9 \quad x_{10}\,x_{11} \quad x_{12}\,x_{13} \quad x_{14}\,x_{15}$$

$$x_0\,x_2 \quad x_4\,x_6 \quad x_8\,x_{10} \quad x_{12}\,x_{14} \mid x_1\,x_3 \quad x_5\,x_7 \quad x_9\,x_{11} \quad x_{13}\,x_{15}$$

$$x_0\,x_4 \quad x_8\,x_{12} \mid x_2\,x_6 \quad x_{10}\,x_{14} \mid x_1\,x_5 \quad x_9\,x_{13} \mid x_3\,x_7 \quad x_{11}\,x_{15}$$

$$x_0\,x_8 \mid x_4\,x_{12} \mid x_2\,x_{10} \mid x_6\,x_{14} \mid x_1\,x_9 \mid x_5\,x_{13} \mid x_3\,x_{11} \mid x_7\,x_{15}$$

Рис. 3.15. Перестановка на 16 элементов

Этот факт является закономерностью. На первом шаге четные элементы с номером n переместились в позицию n/2, а нечетные из позиции в позицию N/2+(n-1)/2. Где n=0, 1,…, N-1. Таким образом, новая позиция вычисляется из старой позиции с помощью функции:

```
ror(n,N) = [n/2] + N{n/2}
```

Здесь как обычно [x] означает целую часть числа, а {x} – дробную.

В ассемблере эта операция называется *циклическим сдвигом вправо (ror)*, если *N* - это степень двойки.

Некоторую проблему представляет собой операция обратной перестановки бит номера позиции `reverse`(), которая не реализована ни в популярной архитектуре Intel, ни в наиболее распространенных языках программирования. Некоторую проблему представляет собой операция обратной перестановки бит номера позиции `reverse`(), которая не реализована ни в популярной архитектуре Intel, ни в наиболее распространенных языках программирования. Приходится реализовывать ее через другие битовые операции. Необходимо реализовывать ее через другие битовые операции.

Вместо того чтобы переставлять биты позиций местами, можно применить и другой метод. Для этого надо вести отсчет 0,1,2,…,N/2-1 уже с обратным следованием битов. Опять-таки ни в Ассемблере Intel, ни в распространенных языках программирования не реализованы операции над обратным битовым представлением, но так как алгоритм приращения на единицу известен, то он реализован программно в приведенном примере.

В этом алгоритме используется тот общеизвестный факт, что при увеличении числа от 0 до бесконечности (с приращением на единицу) каждый бит меняется с 0 на 1 и обратно с определенной периодичностью: младший бит – каждый раз, следующий – каждый второй раз, следующий – каждый четвертый и так далее.

Эта периодичность реализована в виде *T* вложенных циклов, в каждом из которых один из битов позиции *J* переключается туда и обратно с помощью операции *XOR* (В C/C++ она записывается как ^=). Позиция *I* использует обычный инкремент *I*++, уже встроенный в язык программирования.

Данный алгоритм имеет тот недостаток, что требует разного числа вложенных циклов в зависимости от *T*. На практике это не очень плохо, поскольку *T* обычно ограничено некоторым разумным пределом (16 .. 20), так что можно написать столько вариантов алгоритма, сколько нужно. Тем не менее, это делает программу громоздкой, поэтому в примере предлагается вариант, который эмулирует вложенные циклы.

Перестановку бит в одном байте уже можно делать по таблице. Для нее нужно заранее приготовить массив `reverse256` из 256 элементов. Этот массив будет содержать 8-битовые числа. Записываем туда числа от 0 до 255 и переставляем в каждом порядок битов.

Оценим сложность описанных алгоритмов. Понятно, что все они пропорциональны *N* с каким-то коэффициентом. Точное значение коэффициента зависит от конкретной ЭВМ. Во всех случаях мы имеем *N* перестановок со сравнением *I* и *J*, которое предотвращает повторную перестановку некоторых элементов. Рядом присутствует некоторый обрамляющий код, применяющий достаточно быстрые операции над целыми числами: присваивания, сравнения, индексации, битовые операции и условные переходы. Среди них в архитектуре Intel наиболее накладны переходы. Поэтому я бы рекомендовал последний алгоритм. Он содержит всего *N* переходов, а не *2N* как в алгоритме со вложенными циклами или их эмуляцией и не *NT* как в самом первом алгоритме.

С другой стороны, предварительная перестановка занимает мало времени по сравнению с последующими операциями, использующими *(N/2)log2N* умножений комплексных чисел.

Внешний цикл − это основные итерации. На каждой из них *2Nmax/N* ДПФ (длиной по *N/2* элементов каждое) преобразуются в *Nmax/N* ДПФ (длиной по *N* элементов каждое). Следующий цикл по *k* представляет собой цикл синхронного вычисления элементов с индексами *k* и *k + N/2* во всех результирующих ДПФ. Самый внутренний цикл перебирает Nmax/N штук ДПФ одно за другим.

В приведенном примере присутствует реализация простейших операций над комплексными числами (классы Complex и ShortComplex), оптимизированная под данную задачу. Обычно та или иная реализация уже есть для многих компиляторов, так что можно использовать и ее.

Кроме того, массив *W2n* содержит заранее вычисленные коэффициенты *W2, W4, W8,...,WNmax.*

Для вычислений используется наиболее точное представление чисел с плавающей точкой в C++: long double размером в 10 байт на платформе Intel. Для хранения результатов в массивах используется тип double длиной 8 байт. Причина в том, что 64-битные и 32-битные процессоры лучше работают с выровненными по границе 8 байт данными, чем с выровненными по границе 10 байт.

Исходный код для вычисления преобразования Фурье приведен в прил. Г данного учебного пособия.

3.10. Аппаратная реализация преобразования Фурье

Программная реализация преобразования Фурье может быть выполнена на ПЭВМ или на базе процессоров цифровой обработки сигналов. Его практическое применение также возможно и в специализированных техни-ческих средствах, хотя в этом случае более целесообразна реализация на аппаратном уровне.

На базе ПЛИС есть несколько вариантов решения задачи преобразования Фурье, среди которых можно выделить:

1. Использование встроенных средств САПР Quartus для реализации цифрового фильтра при помощи MegaWizard Plugin Manager – FIR Compiler [11, 12].
2. Генерация программного кода на языках описания аппаратуры или создание схемотехнического решения так называемого «с нуля».

Рассмотрим более подробно все перечисленные варианты решения.

3.10.1. Генерация блоков Фурье-преобразования с аппаратным решением на базе Quartus Altera

Генерация блоков прямого цифрового преобразования Фурье для внутрисхемного применения в базисе ПЛИС может проводиться непосред-ственно средствами САПР Quartus II. Данная подпрограмма называется FFT MegaCore Function. Внешний вид конфигурационных окон показан на рис. 3.16.

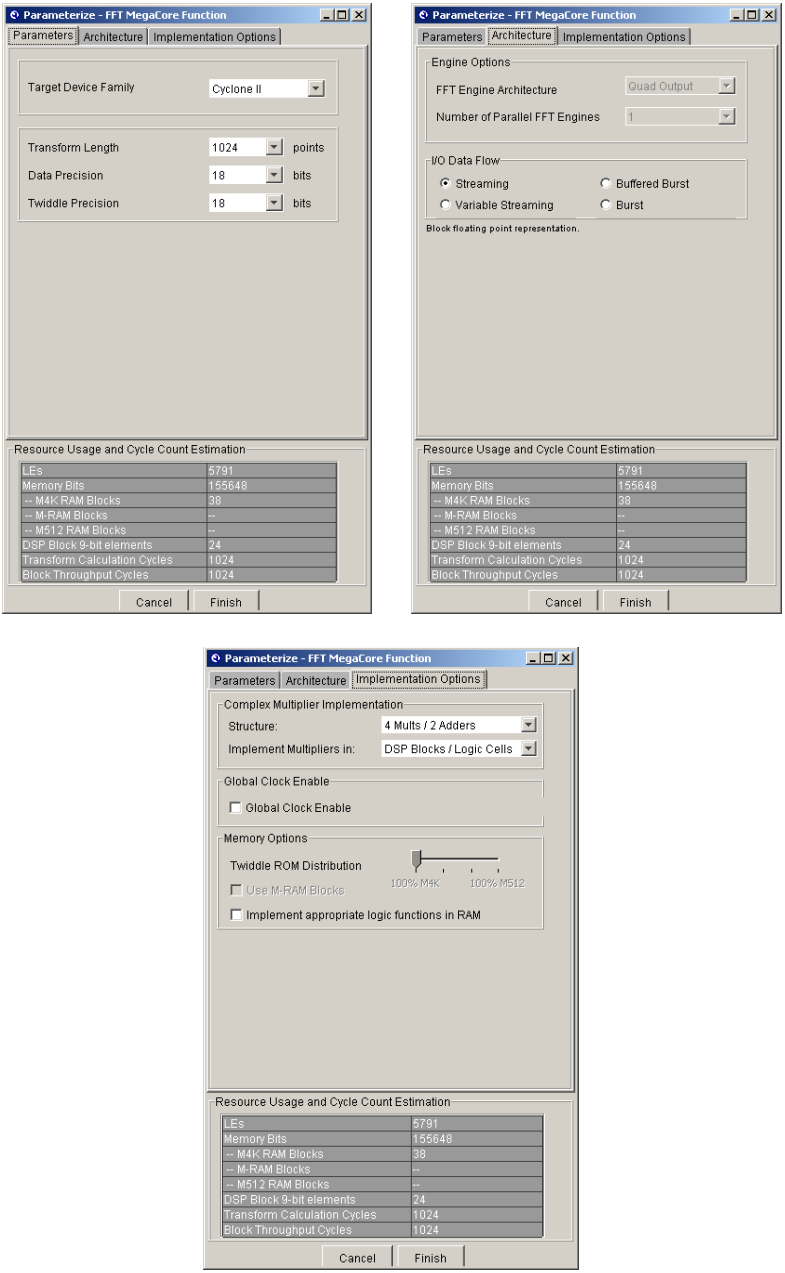


Рис. 3.16. Окна FFT MegaCore Function

В результате работы FFT MegaCore Function получается мегафункция с заданными параметрами, которую можно использовать в проекте.

Условное графическое обозначение сгенерированного решения показано на рис. 3.17.

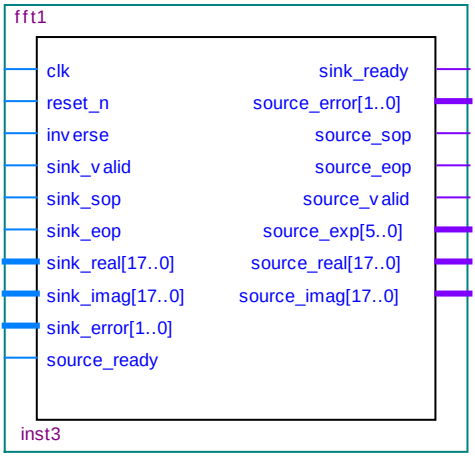


Рис. 3.17. УГО блока дискретного преобразования Фурье

В дополнение к графическому элементу, в проекте генерируется набор заголовочных файлов для подключения к остальным программным компонентам.

Основной заголовочный файл приведен в примере 3.6.

Пример 3.6. Заголовочный файл для подключения синтезированного блока преобразования Фурье.

```
library IEEE;
use IEEE.std_logic_1164.all;
library fft_lib;
use fft_lib.fft_pack_fft_91.all;

ENTITY fff IS
 PORT (
 clk              : IN STD_LOGIC;
 reset_n          : IN STD_LOGIC;
```

```
  inverse          : IN STD_LOGIC;
  sink_valid       : IN STD_LOGIC;
  sink_sop         : IN STD_LOGIC;
  sink_eop         : IN STD_LOGIC;
  sink_real        : IN STD_LOGIC_VECTOR
                     (17 DOWNTO 0);
  sink_imag        : IN STD_LOGIC_VECTOR
                     (17 DOWNTO 0);
  sink_error       : IN STD_LOGIC_VECTOR
                     (1 DOWNTO 0);
  source_ready     : IN STD_LOGIC;
  sink_ready       : OUT STD_LOGIC;
  source_error     : OUT STD_LOGIC_VECTOR
                     (1 DOWNTO 0);
  source_sop       : OUT STD_LOGIC;
  source_eop       : OUT STD_LOGIC;
  source_valid     : OUT STD_LOGIC;
  source_exp       : OUT STD_LOGIC_VECTOR
                     (5 DOWNTO 0);
  source_real      : OUT STD_LOGIC_VECTOR
                     (17 DOWNTO 0);
  source_imag      : OUT STD_LOGIC_VECTOR
                     (17 DOWNTO 0)
  );
END fff;

ARCHITECTURE SYN OF fff IS

COMPONENT asj_fft_sglstream_fft_91
GENERIC (
       nps   : NATURAL;
       bfp   : NATURAL;
       nume  : NATURAL;
       mpr   : NATURAL;
       twr   : NATURAL;
       bpr   : NATURAL;
       bpb   : NATURAL;
       fpr   : NATURAL;
       mram  : NATURAL;
```

```
        m512  : NATURAL;
        mult_type  : NATURAL;
        mult_imp   : NATURAL;
        srr   : STRING;
        rfs1  : STRING;
        rfs2  : STRING;
        rfs3  : STRING;
        rfc1  : STRING;
        rfc2  : STRING;
        rfc3  : STRING
    );
    PORT (
    clk              : IN STD_LOGIC;
    reset_n          : IN STD_LOGIC;
    inverse          : IN STD_LOGIC;
    sink_valid       : IN STD_LOGIC;
    sink_sop         : IN STD_LOGIC;
    sink_eop         : IN STD_LOGIC;
    sink_real        : IN STD_LOGIC_VECTOR
                     (17 DOWNTO 0);
    sink_imag        : IN STD_LOGIC_VECTOR
                     (17 DOWNTO 0);
    sink_error       : IN STD_LOGIC_VECTOR
                     (1 DOWNTO 0);
    source_ready     : IN STD_LOGIC;
    sink_ready       : OUT STD_LOGIC;
    source_error     : OUT STD_LOGIC_VECTOR
                     (1 DOWNTO 0);
    source_sop       : OUT STD_LOGIC;
    source_eop       : OUT STD_LOGIC;
    source_valid     : OUT STD_LOGIC;
    source_exp       : OUT STD_LOGIC_VECTOR
                     (5 DOWNTO 0);
    source_real      : OUT STD_LOGIC_VECTOR
                     (17 DOWNTO 0);
    source_imag      : OUT STD_LOGIC_VECTOR
                     (17 DOWNTO 0)
);
END COMPONENT;
```

```
    BEGIN

    asj_fft_sglstream_fft_91_inst                    :
    asj_fft_sglstream_fft_91
    GENERIC MAP (
           nps => 1024,
           bfp => 1,
           nume => 1,
           mpr => 18,
           twr => 18,
           bpr => 16,
           bpb => 4,
           fpr => 4,
           mram => 0,
           m512 => 0,
           mult_type => 1,
           mult_imp => 0,
           srr                                       =>
"AUTO_SHIFT_REGISTER_RECOGNITION=OFF",
           rfs1 => "fff_1n1024sin.hex",
           rfs2 => "fff_2n1024sin.hex",
           rfs3 => "fff_3n1024sin.hex",
           rfc1 => "fff_1n1024cos.hex",
           rfc2 => "fff_2n1024cos.hex",
           rfc3 => "fff_3n1024cos.hex"
     )
     PORT MAP (
           clk  =>  clk,
           reset_n  =>  reset_n,
           inverse  =>  inverse,
           sink_valid  =>  sink_valid,
           sink_sop  =>  sink_sop,
           sink_eop  =>  sink_eop,
           sink_real  =>  sink_real,
           sink_imag  =>  sink_imag,
           sink_ready  =>  sink_ready,
           sink_error  =>  sink_error,
           source_error  =>  source_error,
           source_ready  =>  source_ready,
```

```
        source_sop  =>  source_sop,
        source_eop  =>  source_eop,
        source_valid  =>  source_valid,
        source_exp  =>  source_exp,
        source_real  =>  source_real,
        source_imag  =>  source_imag
);
END SYN;
```

Далее рассмотрим основы генерации аппаратных решений для цифровых фильтров средствами САПР Quartus II.

3.10.2. Генерация аппаратного решения средствами Quartus II

Альтернативным способом решения задачи цифрового преобразования Фурье для ПЛИС является его прямая реализация на схемотехническом уровне или на языке описания аппаратуры высокого уровня [15]. Полный текст модулей приведен в прил. Д данного учебного пособия.

Комплект состоит из следующих файлов:

1. FFTBeh.vhd – основной файл для выполнения вычислений дискретного преобразования Фурье.
2. TST_FFT.vhd – тестбенч для проверки работоспособности модуля.
3. SQ_GEN.vhd – генератор модельного сигнала для проверки.
4. TO_POLAR.vhd – преобразователь в полярную систему координат.

Для основного блока FFTBeh можно сгенерировать следующее условное графическое обозначение, представленное на рис. 3.18.

Parameter	Value	Type
N	6	Signed Integer
IFFT	0	Signed Integer
PT	0	Signed Integer
overs	0	Signed Integer

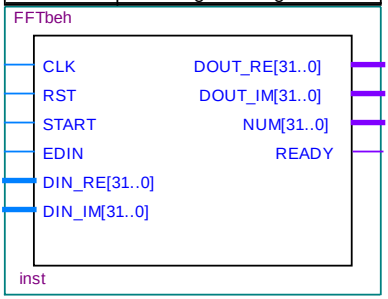


Рис. 3.18. УГО блока FFTbeh

Условное графическое обозначение для блока генерации тестовых сигналов представлено на рис. 3.19.

Parameter	Value	Type
tclk	P"10000000 fs"	Auto
G_period	12	Signed Integer
H_len	4	Signed Integer
E_period	2	Signed Integer
start_period	64	Signed Integer
magn	10	Signed Integer

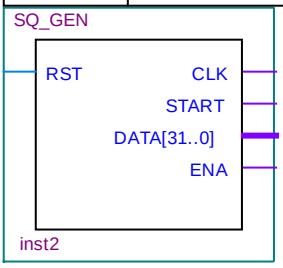


Рис. 3.19. УГО блока генерации тестовых сигналов

УГО блока преобразования в полярную систему координат представлено на рис. 3.20.

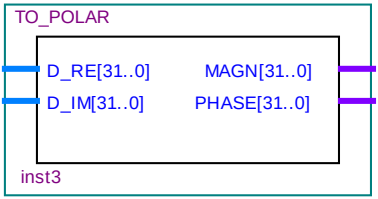


Рис. 3.20. УГО преобразователя в полярную систему координат

Целесообразно вычислять БПФ как цепочку процедур. Первая из них, называемая Gather_C, накапливает входной массив комплексных исходных данных. Вторая процедура – FFT – выполняет алгоритм БПФ, а заключительная процедура Scatter_C выдает результирующий массив в виде последовательности данных.

Указанные процедуры выполняются объектом FFTbeh. Этот объект можно вставлять в нужное место разрабатываемого проекта, как компонент, с целью измерить спектр интересующих сигналов или отобразить сигнал из временной области в частотную.

В библиотеках IEEE.MATH_REAL и IEEE.MATH_COMPLEX хранятся объявления типа комплексного данного, такие функции, как синус, косинус, квадратный корень, функции от комплексных аргументов и другие, которые применяются для вычисления БПФ.

Настроечная переменная N задает длину преобразования. При IDCT = 0 выполняется прямое преобразование, а при 1 – обратное. Переменная PT задает масштаб результата. Таким образом, преобразование равно

$$Y_k = \frac{1}{2^{PT}} \sum_{i=0}^{2^N-1} X_i e^{\left(-j\,2\pi ik\,/2^N\right)},$$

где X_i – отсчет входных комплексных данных: DIN_RE +j*DIN_IM, Yk – отсчет комплексного спектра: DOUT_RE +j*DOUT_IM.

По сигналу, приходящему в порт START, начинается накопление массива входных данных. Окончание вычислений БПФ указывается сигналом с выходного порта READY. Выдаваемые отсчеты спектра сопровождаются номером отсчета NUM, по которому удобно находить значение частоты той или иной спектральной линии.

Различные блоки обработки сигналов целесообразно выполнить с интер-фейсом, аналогичным интерфейсу объекта FFTbeh. Тогда их можно соединять между собой последовательно в соответствии с основным алгоритмом. При этом выходной порт READY одного блока соединяется с входным портом START следующего блока.

Сигнал на порте EDIN стробирует прием входных данных. В общем случае, этот сигнал разрешает работу блока в отдельных тактах. Это необходимо для моделирования и стыковки блоков, работающих с взаимно кратными периодами дискретизации. Например, с помощью блока FFTbeh можно определять узкополосный спектр после предварительной фильтрации полосовым фильтром и снижения частоты дискретизации в M раз. При этом сигнал EDIN должен иметь скважность 1:M.

Поведение объекта FFTbeh описано в архитектуре FFT_int. Для пред-ставления массивов данных в ней объявлены следующие типы массивов комплексных и действительных данных.

В процедуре GATHER_C описана последовательность действий, по которой накапливается массив входных комплексных данных.

Такая процедура существенно отличается от процедур, к которым привыкло большинство программистов. В интерфейсе процедуры явно указываются не только тип данных, но и то, что они – сигналы, а также их направление: входные или выходные данные. При вызове процедуры ей не просто передается управление, а ее копия переписывается в модуль, который ее вызывает и там она связывается. Выполнение этой процедуры запускается по фронту синхросигнала CLK в операторе wait и останавливается после выполнения ее последнего оператора.

В данной процедуре при каждом ее запуске при EDIN = '1' в массив RAM по счетчику datact записывается очередная пара данных.

Процедура SCATTER_C работает так же, как и предыдущая. Но наоборот, накопленные в массиве данные последовательно выдаются на ее выход (gather – собирать, scatter – разбрасывать).

Наконец, процедура, вычисляющая БПФ массива данных, имеет название FFT и приведена также в соответствующей части прил. Д.

Здесь выполняется известный алгоритм БПФ по основанию 2 с прореживанием по времени с замещением данных. Процедура имеет два участка: первый из них выполняется однократно в начале моделирования, а второй – собственно БПФ − выполняется

периодически. При выполнении первого участка формируется таблица вращающих коэффициентов, размер которой равен длине преобразования и которая затем используется алгоритмом БПФ.

Второй участок представляет собой бесконечный цикл. Цикл запускается, как только готов массив исходных данных, подготавливаемый процедурой GATHER_C. После запуска данные переписываются из входного массива в рабочий массив. При этом выполняется двоичная инверсия адресов записи. Следует отметить простую и оригинальную возможность двоичной инверсии адреса, которую предоставляет язык VHDL. Адрес представляется битовым вектором и биты в нем переставляются в инверсном порядке непосредственно. В обычных языках для этого необходимо выполнить сложный и не всегда очевидный алгоритм.

Как и все алгоритмы БПФ, данный алгоритм представляет собой гнездо циклов. Во внутреннем цикле выполняются базовые операции БПФ, а внешний цикл образован N итерациями алгоритма БПФ.

Для понимания исполнения программы следует напомнить, что параллельный вызов процедуры, в которой применяется оператор wait, эквивалентен оператору процесса, тело которого состоит из тела процедуры. Таким образом, все три процедуры, как и эквивалентные им процессы, исполняются параллельно в некотором конвейерном режиме. При этом данные и управление передаются от процедуры к процедуре как между ступенями конвейера. В результате, объект FFTbeh обрабатывает непрерывный поток данных, сегментируя его на блоки длиной 2**N.

Процесс DEL выполняет задержку сигнала готовности READY и потока номеров NUM отсчетов результата на один такт, чтобы они соответствовали выходным отсчетам сигналов DOUT_RE и DOUT_IM.

Рассмотрим простой пример применения объекта FFTbeh для измерения спектра сигнала, выдаваемого генератором прямоугольных импульсов.

Графическая программа, подготовленная в САПР, показана на рис. 3.21.

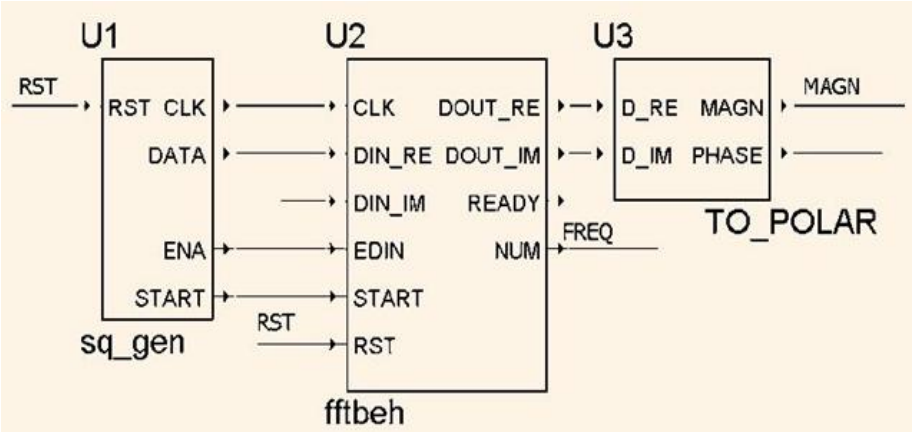


Рис. 3.21. Пример использования блока БПФ в ПЛИС

Здесь компонент U1 представляет собой программируемый генератор прямоугольных импульсов DATA с выдачей синхросерии CLK, сигналов START и ENA. Компонент U2 − это описанный выше блок БПФ, а U3 – преобразователь комплексных чисел из прямоугольной системы координат в полярную.

При необходимости, предложенный в [16] блок БПФ можно моди-фицировать для работы с данными с плавающей запятой и использовать в своих проектах,

Заключение

Таким образом, в данной главе были приведены все необходимые теоретические и практические материалы для выполнения быстрых преобразований Фурье в аналитической форме и для внедрения на практике, что и предлагается сделать в соответствующей лабораторной работе № 2, исходные данные к которой приведены в прил. Б данного учебного пособия.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Гузик В.Ф., Золотовский В.Е. Проблемно-ориентированные высоко-производительные вычислительные системы. – Таганрог, Изд-во ТРТУ, 1999 – 232 с.
2. Гузик В.Ф., Золотовский В.Е., Беспалов Д.А., Ляпунцова Е.В. Лабораторный практикум по курсу «Проблемно-ориентированные вычислительные системы». − Таганрог: Изд-во ТТИ ЮФУ, 2009. − 113 с.
3. Гузик В.Ф. Проектирование проблемно-ориентированных вычислительных систем. Часть 1: Монография. – Таганрог: Изд-во ТТИ ЮФУ, 2009. – 463с.
4. Золотовский В.Е. Проектирование высокопроизводительных проблемно-ориентированных вычислительных систем. Ч. 2. – Таганрог: Изд-во ТТИ ЮФУ, 2011. – 356с.
5. Сергиенко А.Б. Цифровая обработка сигналов. СПб. Изд-во Питер, 2002. – 608 с.
6. Каппелини В., Константинидис Дж., Эмилиани П. Цифровые фильтры и их применение. – М.: Энергоатомиздат, 1983.
7. Хемминг Р.В. Цифровые фильтры. – М.: Недра, 1987.
8. Рабинер Л.Р., Гоулд В. Теория и применение цифровой обработки сигналов. – М.: Мир, 1978.
9. Яншин В.В., Калинин Г.А. Обработка изображений на языке С для IBM PC: Алгоритмы и программы. – М.: Мир, 1994.
10. Бочканов С., Быстрицкий В. Быстрое преобразование Фурье и его приложения. http://alglib.sources.ru/fft/. 2009.
11. Дьяконов В., Абраменкова И. MATLAB. Обработка сигналов и изображений. СПб. Изд-во Питер, 2002.
12. Куприянов М. С., Матюшин Б. Д. Цифровая обработка сигналов. – СПб.: Политехника, 1998. – 592 с.
13. Introduction to the Quartus II Software. Version 10.0. ALTERA Corporation. 2010.
14. VHDL code for a 4 tap FIR filter. http://vhdlguru.blogspot.com /2011/06/vhdl-code-for-4-tap-fir-filter.html.
15. Сергиенко А.М. Поведенческая модель процессора БПФ. http://kanyevsky.kpi.ua/useful_core/behavioral_fft.html.
16. Войнаровский М. Быстрое преобразование Фурье. http://psi-logic.narod.ru/fft/fft.htm.

ПРИЛОЖЕНИЕ А.

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ № 1: РАСЧЕТ И ИССЛЕДОВАНИЕ ЦИФРОВЫХ ФИЛЬТРОВ

А.1. Содержание отчета к лабораторной работе № 1

1. Титульный лист с названием работы, фамилией студента и преподавателя.
2. Цель лабораторной работы.
3. Вариант задания.
4. Структура фильтра.
5. Расчётные формулы.
6. Процедура генерации входного сигнала.
7. График входного сигнала.
8. Графики АЧХ и ФЧХ фильтра.
9. Процедуру фильтрации сигнала.
10. График обработанного сигнала.
11. Совмещенный график исходного и обработанного в MathCad сигнала.
12. Листинг программной реализации процедуры фильтрации сигнала.
13. График сигнала, полученного в результате программной обработки.
14. Совмещенный график исходного и обработанного программой сигнала.
15. Совмещенный график обработанного программой сигнала и сигнала, обработанного в MathCad.
16. Процедура генерации аппаратного решения задачи цифровой фильтрации в базисе ПЛИС.
17. Исходные коды и схемы модулей, составляющих аппаратное решение.
18. Готовая схема аппаратной конфигурации ПЛИС.
19. Временные диаграммы результатов моделирования.
20. Выводы по лабораторной работе.
21. Список использованных источников.

А.2. Исходные данные для генерации входного сигнала к лабораторной работе № 1.

Таблица А.1

№		1	2	3	4	5	6	7
1	F, кГц	0,5	1	2	5	7	9	12
	A	1	5	3	7	3	2	1
2	F, кГц	3	5	6	9	12	15	18
	A	7	9	8	12	7	4	6
3	F, кГц	0,9	1	2	9	12	15	18
	A	2	5	3	12	7	4	6
4	F, кГц	3	5	6	8	9	10	12
	A	7	9	8	7	3	2	5
5	F, кГц	0,5	1	2	9	12	14	17
	A	1	5	3	12	15	4	6
6	F, кГц	2	4	5	7	9	12	15
	A	11	15	12	13	21	10	9
7	F, кГц	9	11	15	17	21	29	35
	A	17	19	36	34	28	25	40
8	F, кГц	0,3	0,56	1,2	3,5	5	7	9
	A	21	18	23	19	13	20	11
9	F, кГц	10	13	14	17	21	22	25
	A	9	14	15	19	21	26	30
10	F, кГц	1	5	7	13	18	19	29
	A	9	11	12	17	21	29	35
11	F, кГц	10	11	12	13	15	20	25
	A	9	11	15	17	21	29	35
12	F, кГц	21	25	32	33	41	45	50
	A	9	11	15	17	21	29	35
13	F, кГц	20	21	22	23	27	30	40
	A	9	11	15	17	21	29	35
14	F, кГц	11	15	12	13	21	23	25
	A	19	12	15	27	23	29	25
15	F, кГц	5	11	12	13	21	23	30
	A	9	11	15	17	21	29	35
16	F, кГц	11	13	15	17	21	23	25
	A	22	23	25	17	21	29	35

Формула генерации входного сигнала

$$X = A1 \sin 2\pi f1\, t + A2 \sin 2\pi f2\, t + A3 \sin 2\pi f3\, t + A4 \sin 2\pi f4\, t + \\ + A5 \sin 2\pi f5\, t + A6 \sin 2\pi f6 t + A7 \sin 2\pi f7\, t.$$

А.3. Наборы коэффициентов фильтров к лабораторной работе № 1

Таблица А.2

Вариант 1 h(1)= 8.58398885E-0003=h(16) h(2)= 1.01415502E-0002=h(15) h(3)=-1.18238033E-0002=h(14) h(4)=-4.97749198E-0002=h(13) h(5)=-5.10048196E-0002=h(12) h(6)= 3.87851131E-0002=h(11) h(7)= 2.01367094E-0001=h(10) h(8)= 3.33522835E-0001=h(9)	**Вариант 2** h(1)=-1.49502378E-0002=h(16) h(2)=-4.91854122E-0002=h(15) h(3)=-1.93852179E-0002=h(14) h(4)= 5.71528560E-0002=h(13) h(5)=-2.44721295E-0003=h(12) h(6)=-1.15195509E-0001=h(11) h(7)= 9.20622552E-0002=h(10) h(8)= 4.94733996E-0001=h(9)
Вариант 3 h(1)=-3.74076788E-0002=h(16) h(2)=-5.42711395E-0003=h(15) h(3)= 2.96063461E-0002=h(14) h(4)=-5.59466344E-0002=h(13) h(5)= 6.30845354E-0002=h(12) h(6)=-2.47230490E-0002=h(11) h(7)=-9.96993267E-0002=h(10) h(8)= 5.95044626E-0001=h(9)	**Вариант 4** h(1)= 3.42591553E-0003=-h(16) h(2)= 1.98488129E-0002=-h(15) h(3)=-5.43602554E-0002=-h(14) h(4)= 2.56355684E-0002=-h(13) h(5)= 6.93667396E-0002=-h(12) h(6)=-6.39440064E-0002=-h(11) h(7)=-1.63419803E-0001=-h(10) h(8)= 4.29283935E-0001=-h(9)
Вариант 5 h(1)=-1.25937288E-0002=-h(16) h(2)=-2.68906576E-0002=-h(15) h(3)= 7.24777621E-0002=-h(14) h(4)=-1.01029481E-0001=-h(13) h(5)= 6.35336599E-0002=-h(12) h(6)= 5.55492140E-0002=-h(11) h(7)=-2.12490481E-0001=-h(10) h(8)= 3.24046289E-0001=-h(9)	**Вариант 6** h(1)= 3.42591553E-0003=-h(16) h(2)= 1.98488129E-0002=-h(15) h(3)=-5.43602554E-0002=-h(14) h(4)= 2.56355684E-0002=-h(13) h(5)= 6.93667396E-0002=-h(12) h(6)=-6.39440064E-0002=-h(11) h(7)=-1.63419803E-0001=-h(10) h(8)= 4.29283935E-0001=-h(9)

Продолжение табл. А.2

Вариант 7
h(1)=-2.91731066E-0002=h(16)
h(2)=-2.72994673E-0002=h(15)
h(3)=-1.67785454E-0002=h(14)
h(4)= 1.88149767E-0002=h(13)
h(5)= 7.89431816E-0002=h(12)
h(6)= 1.52199144E-0001=h(11)
h(7)= 2.19067035E-0001=h(10)
h(8)= 2.59023716E-0001=h(9)

Вариант 8
h(1)=-2.96369758E-0002=h(16)
h(2)= 3.96338407E-0002=h(15)
h(3)=-6.04446063E-0002=h(14)
h(4)= 8.28993022E-0002=h(13)
h(5)=-1.04456130E-0001=h(12)
h(6)= 1.22393812E-0001=h(11)
h(7)=-1.34283081E-0001=h(10)
h(8)= 1.38447825E-0001=h(9)

Вариант 9
h(1)=-5.48848226E-0003=h(16)
h(2)=-1.12987967E-0002=h(15)
h(3)= 1.77272999E-0002=h(14)
h(4)= 2.91855736E-0002=h(13)
h(5)=-6.71082094E-0002=h(12)
h(6)=-4.98384054E-0002=h(11)
h(7)= 3.02652469E-0001=h(10)
h(8)= 5.59163628E-0001=h(9)

Вариант 10
h(1)= 1.33048788E-0002=h(26)
h(2)= 6.45961826E-0003=h(25)
h(3)=-2.12425816E-0002=h(24)
h(4)= 7.56317498E-0003=h(23)
h(5)=-3.77655834E-0003=h(22)
h(6)= 2.30796766E-0002=h(21)
h(7)= 2.14396319E-0002=h(20)
h(8)=-1.12723132E-0001=h(19)
h(9)= 5.00865353E-0002=h(18)
h(10)= 1.70572861E-0001=h(17)
h(11)=-2.03180859E-0001=h(16)
h(12)=-8.55489605E-0002=h(15)
h(13)= 2.86737905E-0001=h(14)

Вариант 11
h(1)= 1.33048788E-0002=h(26)
h(2)= 6.45961826E-0003=h(25)
h(3)=-2.12425816E-0002=h(24)
h(4)= 7.56317498E-0003=h(23)
h(5)=-3.77655834E-0003=h(22)
h(6)= 2.30796766E-0002=h(21)
h(7)= 2.14396319E-0002=h(20)
h(8)=-1.12723132E-0001=h(19)
h(9)= 5.00865353E-0002=h(18)
h(10)= 1.70572861E-0001=h(17)
h(11)=-2.03180859E-0001=h(16)
h(12)=-8.55489605E-0002=h(15)
h(13)= 2.86737905E-0001=h(14)

Вариант 12
h(1)= 8.81847553E-0004=h(26)
h(2)=-1.87613842E-0006=h(25)
h(3)= 1.27260558E-0002=h(24)
h(4)= 1.17333529E-0006=h(23)
h(5)=-3.85919636E-0002=h(22)
h(6)=-7.94669754E-0007=h(21)
h(7)= 2.17106507E-0002=h(20)
h(8)=-4.60638630E-0007=h(19)
h(9)= 9.94375191E-0002=h(18)
h(10)=-1.26607506E-0007=h(17)
h(11)=-2.79463243E-0001=h(16)
h(12)=-2.18474142E-0007=h(15)
h(13)= 3.68401231E-0001=h(14)

Продолжение таблицы А.2

Вариант 13 h(1)=-6.67003549E-0003=h(26) h(2)=-1.13440416E-0002=h(25) h(3)= 1.26537690E-0003=h(24) h(4)= 1.81025848E-0002=h(23) h(5)= 1.03126475E-0002=h(22) h(6)= 1.13130144E-0003=h(21) h(7)= 3.54303667E-0002=h(20) h(8)= 5.60615704E-0002=h(19) h(9)=-4.35435667E-0002=h(18) h(10)=-1.86302739E-0001=h(17) h(11)=-1.37032000E-0001=h(16) h(12)= 1.22843757E-0001=h(15) h(13)= 2.79974827E-0001=h(14)	**Вариант 14** h(1)=-7.74655555E-0003=h(26) h(2)= 2.66044188E-0006=h(25) h(3)= 1.10328299E-0002=h(24) h(4)= 6.53431514E-0006=h(23) h(5)= 3.39948164E-0002=h(22) h(6)= 9.19342888E-0006=h(21) h(7)=-5.60192530E-0002=h(20) h(8)= 9.49357724E-0006=h(19) h(9)=-6.20920915E-0002=h(18) h(10)= 9.79372561E-0006=h(17) h(11)= 2.97792590E-0001=h(16) h(12)= 9.79239746E-0006=h(15) h(13)= 5.76269151E-0001=h(14)
Вариант 15 h(1)= 8.85904164E-0004=h(26) h(2)=-5.22651573E-0006=h(25) h(3)=-1.27182193E-0002=h(24) h(4)=-6.91671243E-0006=h(23) h(5)=-3.85883801E-0002=h(22) h(6)= 2.25802543E-0006=h(21) h(7)=-2.17178060E-0002=h(20) h(8)= 9.67454560E-0006=h(19) h(9)= 9.94318261E-0002=h(18) h(10)= 2.25802543E-0006=h(17) h(11)= 2.79471093E-0001=h(16) h(12)=-1.21432281E-0005=h(15) h(13)= 3.68419060E-0001=h(14)	**Вариант 16** h(1)=-4.17748598E-0002=h(26) h(2)= 4.62757226E-0002=h(25) h(3)= 1.59902868E-0002=h(24) h(4)=-1.19796053E-0002=h(23) h(5)=-1.49204402E-0002=h(22) h(6)= 6.34415745E-0002=h(21) h(7)=-9.61881971E-0003=h(20) h(8)=-9.91513855E-0002=h(19) h(9)= 1.70396963E-0001=h(18) h(10)=-1.11097547E-0001=h(17) h(11)=-2.41879918E-0002=h(16) h(12)= 1.48144291E-0001=h(15) h(13)= 8.08229724E-0001=h(14)

Окончание таблицы А.2

Вариант 17

H(1)= 8.50294410E-0003=-h(30)
H(2)= 2.53034852E-0004=-h(29)
H(3)=-1.01544916E-0004=-h(28)
H(4)=-5.11994595E-0003=-h(27)
H(5)= 2.53222342E-0002=-h(26)
H(6)= 4.38108535E-0002=-h(25)
H(7)=-2.86237806E-0002=-h(24)
H(8)=-5.51989237E-0004=-h(23)
H(9)=-2.34531488E-0002=-h(22)
H(10)=-2.02102712E-0001=-h(21)
H(11)=-1.88400375E-0002=-h(20)
H(12)= 1.68516943E-0001=-h(19)
H(13)= 8.52473506E-0004=-h(18)
H(14)= 1.70612542E-0001=-h(17)
H(15)= 2.18274175E-0001=-h(16)

Вариант 19

$$H(z) = C\prod_{i=1}^{4}\frac{b_{0i} + b_{1i}z^{-1} + b_2 z^{-2}}{a_{0i}+a_{1i}z^{-1} + a_{2i}z^{-2}},$$

C= 598.415484
b0i=1; a0i=1; b11=1; b21=0;
a11 = - 0.5562958; a21=0;
b12 =-0.5954765; b22=1;
a12=-1.1651881; a22=0.6287697;
b13 =-0.3192287; b23=1;
a13 = -1.0107832; a23 =0.8114283
b14 = -0.6279658; b24 =1;
a14=-0.9464295; a24=0.9463153

Вариант 18

$$H(z) = C\prod_{i=1}^{4}\frac{b_{0i} + b_{1i}z^{-1} + b_2 z^{-2}}{a_{0i}+a_{1i}z^{-1} + a_{2i}z^{-2}},$$

C= 449.074575
b0i=1; a0i=1;
b11=1; b21=0;
a11 = - 0.7240457; a21=0;
b12 =- 1.4033106; b22=1;
a12=-1.4677223;
a22=0.6287697;
b13 =-1.2358506;
b23=1;
a13 = -1.5071789;
a23 =0.8114283;
b14 = -0.4545053;
b24 =1;
a14=-1.5552730;
a24=0.9463153

ПРИЛОЖЕНИЕ Б.

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ № 2: РЕАЛИЗАЦИЯ БЫСТРОГО ПРЕОБРАЗОВАНИЯ ФУРЬЕ

Б.1. Содержание отчета к лабораторной работе № 2

1. Титульный лист с названием работы, фамилией студента и преподавателя.
2. Цель и вариант лабораторной работы.
3. Структура вычисления дискретного и непрерывного преобразования Фурье.
4. Расчётные формулы.
5. Процедура вычисления быстрого преобразования Фурье.
6. График входного сигнала (по исходным данным к лабораторной работе № 1).
7. Процедуру обработки сигнала.
8. График коэффициентов Фурье для исходного сигнала (до фильтрации).
9. График коэффициентов Фурье для обработанного сигнала (после фильтрации)
10. Совмещенный график исходного и обработанного в MathCad сигнала.
11. Набор характерных гармоник сигнала (не изменившейся, ослабленной/усиленной, удаленной в процессе фильтрации).
12. Листинг программной реализации процедуры Фурье-преобразования.
13. График коэффициентов Фурье, полученных в результате программной обработки.
14. Совмещенный график программной реализации преобразования Фурье и реализации в MathCad.
15. Процедура генерации аппаратного решения задачи преобразования Фурье в базисе ПЛИС.
16. Исходные коды и схемы модулей, составляющих аппаратное решение.
17. Готовая схема аппаратной конфигурации ПЛИС.
18. Временные диаграммы результатов моделирования.
19. Выводы по лабораторной работе.
20. Список использованных источников.

ПРИЛОЖЕНИЕ В.

ИСХОДНЫЙ КОД ДЛЯ РЕАЛИЗАЦИИ ЦИФРОВОГО ФИЛЬТРА В БАЗИСЕ ПЛИС НА ЯЗЫКЕ ОПИСАНИЯ АППАРАТУРЫ VHDL

В.1. Основной модуль.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity fir_4tap is
port (
       Clk   : in std_logic;
       Xin   : in signed(7 downto 0);
       Yout : out signed(15 downto 0)
);
end fir_4tap;

architecture Behavioral of fir_4tap is

component DFF is
port(
     Q      : out signed(15 downto 0);
     Clk    : in std_logic;
     D      : in  signed(15 downto 0)
);
end component;

signal H0,H1,H2,H3
            : signed(7 downto 0) := (others =>
            '0');
signal MCM0,MCM1,MCM2,MCM3,
            add_out1,add_out2,add_out3
            : signed(15 downto 0) := (others
            => '0');
```

```
    signal Q1,Q2,Q3
                : signed(15 downto 0) := (others
                => '0');

    begin

    -- filter coefficient initializations.
    -- H = [-2 -1 3 4].
    H0 <= to_signed(-2,8);
    H1 <= to_signed(-1,8);
    H2 <= to_signed(3,8);
    H3 <= to_signed(4,8);

    -- multiple constant multiplications.
    MCM3 <= H3*Xin;
    MCM2 <= H2*Xin;
    MCM1 <= H1*Xin;
    MCM0 <= H0*Xin;

    -- adders
    add_out1 <= Q1 + MCM2;
    add_out2 <= Q2 + MCM1;
    add_out3 <= Q3 + MCM0;

    -- flipflops(for introducing a delay).
    dff1 : DFF port map(Q1, Clk, MCM3);
    dff2 : DFF port map(Q2, Clk, add_out1);
    dff3 : DFF port map(Q3, Clk, add_out2);

    -- an output produced at every positive edge
of clock cycle.
    Process (Clk)
    begin
     if (rising_edge(Clk)) then
           Yout <= add_out3;
    end if;
    end process;

    end Behavioral;
```

B.2. Модуль задержки

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DFF is
 port (
       Q     : out signed(15 downto 0);
       Clk   : in std_logic;
       D     : in  signed(15 downto 0)
   );
end DFF;

architecture Behavioral of DFF is
signal qt    : signed(15 downto 0) :=
             (others => '0');
begin

Q <= qt;

process(Clk)
begin
if ( rising_edge(Clk) ) then
 qt <= D;
end if
end process;

end Behavioral;
```

ПРИЛОЖЕНИЕ Г.

ИСХОДНЫЙ КОД ДЛЯ ПРОГРАММНОЙ РЕАЛИЗАЦИИ АЛГОРИТМА БЫСТРОГО ПРЕОБРАЗОВАНИЯ ФУРЬЕ.

Г.1. Модуль FFT.H.

```
#ifndef FFT_H_
#define FFT_H_

struct Complex;
struct ShortComplex;

/*  Fast  Fourier  Transformation:  direct
(complement= false) and complement (complement
= true). 'x' is data source. 'x' contains 2^T
items. */

extern void fft(ShortComplex *x, int T,
                  bool complement);

struct ShortComplex {
 double re, im;
 inline void operator=(const Complex &y);
};

struct Complex {
 long double re, im;
 inline void operator= (const Complex &y);
 inline void operator=
                  (const ShortComplex &y);
};

inline    void    ShortComplex::operator=(const
Complex &y)
               {   re   =   (double)y.re;   im   =
          (double)y.im;  }
```

```
    inline void Complex::operator= (const Complex
&y)
                { re = y.re; im = y.im; }
    inline    void    Complex::operator=    (const
ShortComplex &y)
                { re = y.re; im = y.im; }
    #endif
```

Г.2. Модуль FFT.C.

```
    #include "fft.h"

    // This array contains values from 0 to 255
    with reverse bit
    // order
    static unsigned char reverse256[]= {
    0x00,  0x80,  0x40,  0xC0,  0x20,  0xA0,  0x60,
    0xE0,  0x10,  0x90,  0x50,  0xD0,  0x30,  0xB0,
    0x70,  0xF0,  0x08,  0x88,  0x48,  0xC8,  0x28,
    0xA8,  0x68,  0xE8,  0x18,  0x98,  0x58,  0xD8,
    0x38,  0xB8,  0x78,  0xF8,  0x04,  0x84,  0x44,
    0xC4,  0x24,  0xA4,  0x64,  0xE4,  0x14,  0x94,
    0x54,  0xD4,  0x34,  0xB4,  0x74,  0xF4,  0x0C,
    0x8C,  0x4C,  0xCC,  0x2C,  0xAC,  0x6C,  0xEC,
    0x1C,  0x9C,  0x5C,  0xDC,  0x3C,  0xBC,  0x7C,
    0xFC,  0x02,  0x82,  0x42,  0xC2,  0x22,  0xA2,
    0x62,  0xE2,  0x12,  0x92,  0x52,  0xD2,  0x32,
    0xB2,  0x72,  0xF2,  0x0A,  0x8A,  0x4A,  0xCA,
    0x2A,  0xAA,  0x6A,  0xEA,  0x1A,  0x9A,  0x5A,
    0xDA,  0x3A,  0xBA,  0x7A,  0xFA,  0x06,  0x86,
    0x46,  0xC6,  0x26,  0xA6,  0x66,  0xE6,  0x16,
    0x96,  0x56,  0xD6,  0x36,  0xB6,  0x76,  0xF6,
    0x0E,  0x8E,  0x4E,  0xCE,  0x2E,  0xAE,  0x6E,
    0xEE,  0x1E,  0x9E,  0x5E,  0xDE,  0x3E,  0xBE,
    0x7E,  0xFE,  0x01,  0x81,  0x41,  0xC1,  0x21,
    0xA1,  0x61,  0xE1,  0x11,  0x91,  0x51,  0xD1,
    0x31,  0xB1,  0x71,  0xF1,  0x09,  0x89,  0x49,
    0xC9,  0x29,  0xA9,  0x69,  0xE9,  0x19,  0x99,
    0x59,  0xD9,  0x39,  0xB9,  0x79,  0xF9,  0x05,
```

```
    0x85,  0x45,  0xC5,  0x25,  0xA5,  0x65,  0xE5,
    0x15,  0x95,  0x55,  0xD5,  0x35,  0xB5,  0x75,
    0xF5,  0x0D,  0x8D,  0x4D,  0xCD,  0x2D,  0xAD,
    0x6D,  0xED,  0x1D,  0x9D,  0x5D,  0xDD,  0x3D,
    0xBD,  0x7D,  0xFD,  0x03,  0x83,  0x43,  0xC3,
    0x23,  0xA3,  0x63,  0xE3,  0x13,  0x93,  0x53,
    0xD3,  0x33,  0xB3,  0x73,  0xF3,  0x0B,  0x8B,
    0x4B,  0xCB,  0x2B,  0xAB,  0x6B,  0xEB,  0x1B,
    0x9B,  0x5B,  0xDB,  0x3B,  0xBB,  0x7B,  0xFB,
    0x07,  0x87,  0x47,  0xC7,  0x27,  0xA7,  0x67,
    0xE7,  0x17,  0x97,  0x57,  0xD7,  0x37,  0xB7,
    0x77,  0xF7,  0x0F,  0x8F,  0x4F,  0xCF,  0x2F,
    0xAF,  0x6F,  0xEF,  0x1F,  0x9F,  0x5F,  0xDF,
    0x3F, 0xBF, 0x7F, 0xFF, };

    // This is minimized version of type
'complex'.
    // All operations is inline
    static long double temp;
    inline void operator+=(ShortComplex &x, const
Complex &y)
          { x.re += (double)y.re; x.im +=
     (double)y.im; }
    inline void operator-=(ShortComplex &x, const
Complex &y)
          { x.re -= (double)y.re; x.im -=
     (double)y.im; }
    inline void operator*=(Complex &x, const
Complex &y)
          { temp = x.re; x.re = temp * y.re - x.im
     * y.im;
            x.im = temp * y.im + x.im * y.re; }
    inline void operator*=(Complex &x, const
ShortComplex &y)
          { temp = x.re; x.re = temp * y.re - x.im
          * y.im; x.im = temp * y.im + x.im *
          y.re; }
```

```
    inline void operator/=(ShortComplex &x, double
          div)                  { x.re /= div; x.im
          /= div; }

    //This  is  array  exp(-2*pi*j/2^n)  for  n=
1,...,32
    //exp(-2*pi*j/2^n) = Complex( cos(2*pi/2^n), -
sin(2*pi/2^n) )
    static Complex W2n[32]={
      {-1.000000000000000000000000000000,
      0.000000000000000000000000000000},
      //                W2                calculator
      {0.000000000000000000000000000000,
      -1.000000000000000000000000000000},
      // W4: p/2=o, p/2=s
      {0.707106781186547524400844362104 85,
      -0.707106781186547524400844362104 85},
      // W8: p/4=o, p/4=s
      {0.923879532511286756128183189396 79,
      -0.382683432365089771728459984030 40},
      // p/8=o, p/8=s
      {0.980785280403230449126182236134 24,
      -0.195090322016128267848284868477 02},
      // p/16=
      {0.995184726672196886244836953109 48,
      -9.801714032956060199419556388 86e-2},
      // p/32=
      {0.998795456205172392714771604759 10,
      -4.906767432741801425495497694 26e-2},
      // p/64=
      {0.999698818696204220115765649666 17,
      -2.454122852291228803173452945 92e-2},
      // p/128=
      {0.999924701839144540921646491196 38,
      -1.227153828571992607940826195 10e-2},
      // p/256=
      {0.999981175282601142656990437728 57,
      -6.135884649154475359640234590 37e-3},
      // p/(2y9)=
```

```
{0.99999529380957617151158012570012,
-3.06795676296597627014536549091e-3},
// p/(2y10)=
{0.99999882345170190992902571017153,
-1.53398018628476561230369715026e-3},
// p/(2y11)=
{0.99999970586288221916022821773877,
-7.66990318742704526938568357948e-4},
// p/(2y12)=
{0.99999992646571785114473148070739,
-3.83495187571395589072461681181e-4},
// p/(2y13)=
{0.99999998161642929380834691540291,
-1.91747597310703307439909561989e-4},
// p/(2y14)=
{0.99999999540410731289097193313961,
-9.58737990959773458705172109764e-5},
// p/(2y15)=
{0.99999999885102682756267330779455,
-4.79368996030668845490039904946e-5},
// p/(2y16)=
{0.99999999971275670684941397221864,
-2.39684498084182187291865771650e-5},
// p/(2y17)=
{0.99999999992818917670977509588385,
-1.19842249050697064215215615969e-5},
// p/(2y18)=
{0.99999999998204729417728262414778,
-5.99211245264242784287971180889e-6},
// p/(2y19)=
{0.99999999999551182354431058417300,
-2.99605622633466075045481280835e-6},
// p/(2y20)=
{0.99999999999887795588607701655175,
-1.49802811316901122885427884615e-6},
// p/(2y21)=
{0.99999999999971948897151921479472,
-7.49014056584715721130498566730e-7},
// p/(2y22)=
```

```
      {0.99999999999992987224287980123973,
      -3.74507028292384123903169179084e-7},
      // p/(2y23)=
      {0.99999999999998246806071995015625,
      -1.87253514146195344868824576593e-7},
      // p/(2y24)=
      {0.99999999999999561701517998752946,
      -9.36267570730980827990672866808e-8},
      // p/(2y25)=
      {0.99999999999999890425379499688176,
      -4.68133785365490926951155181385e-8},
      // p/(2y26)=
      {0.99999999999999972606344874922040,
      -2.34066892682745527595054934190e-8},
      // p/(2y27)=
      {0.99999999999999993151586218730510,
      -1.17033446341372771812462135032e-8},
      // p/(2y28)=
      {0.99999999999999998287896554682627,
      -5.85167231706863869080979010083e-9},
      // p/(2y29)=
      {0.99999999999999999571974138670657,
      -2.92583615853431935792823046906e-9},
      // p/(2y30)=
      {0.99999999999999999892993534667664,
      -1.46291807926715968052953216186e-9},
      // p/(2y31)=
    };
    /* x: x - array of items, T: 1 << T = 2 power
    T - number of items in array complement: false
    - normal (direct) transformation, true -
    reverse transformation */
    void fft(ShortComplex *x, int T, bool
complement) {
        unsigned int I, J, Nmax, N, Nd2, k, m,
mpNd2, Skew;
        unsigned char *Ic = (unsigned char*) &I;
        unsigned char *Jc = (unsigned char*) &J;
        ShortComplex S;
```

```
        ShortComplex *Wstore, *Warray;
        Complex WN, W, Temp, *pWN;
        Nmax = 1 << T;
        //first interchanging
        For (I = 1; I < Nmax - 1; I++) {
            Jc[0]  =  reverse256[Ic[3]];  Jc[1]  =
reverse256[Ic[2]];
            Jc[2]  =  reverse256[Ic[1]];  Jc[3]  =
reverse256[Ic[0]];
            J >>= (32 - T);
            if (I < J) {
                S = x[I];
                x[I] = x[J];
                x[J] = S;
            }
        }
        //rotation multiplier array allocation
        Wstore = new ShortComplex[Nmax / 2];
        Wstore[0].re = 1.0;
        Wstore[0].im = 0.0;
        //main loop
        For (   N = 2, Nd2 = 1, pWN = W2n, Skew =
Nmax >> 1;
                N <= Nmax; Nd2 = N, N += N, pWN++,
                Skew >>= 1 ) {
           //WN = W(1, N) = exp(-2*pi*j/N)
           WN= *pWN;
           if (complement)
                WN.im = -WN.im;
           For (Warray = Wstore, k = 0; k < Nd2;
                k++, Warray += Skew) {
               if (k & 1) {
                     W *= WN; *Warray = W;
               }
               else W = *Warray;
               for(m = k; m < Nmax; m += N) {
                     mpNd2 = m + Nd2;Temp = W;
                     Temp       *=       x[mpNd2];
     x[mpNd2] = x[m];
```

```
                        x[mpNd2] -= Temp;      x[m]  +=
Temp;
                }
            }
        }
        delete [] Wstore;
        if (complement) {
            for( I = 0; I < Nmax; I++ )
                x[I] /= Nmax;
        }
    }
```

ПРИЛОЖЕНИЕ Д.

КОМПЛЕКТ ИСХОДНЫХ КОДОВ ДЛЯ РЕАЛИЗАЦИИ ПРЕОБРАЗОВАНИЯ ФУРЬЕ В БАЗИСЕ ПЛИС

Д.1. Модуль FFTbeh.VHD

```
    ------------------------------------------------
    -- DESIGN         :    Library for DSP
    -- FILENAME       :    FFTBeh.vhd
    -- DESCRIPTION    :    Fast  Fourier  Transform
behavioral model
    -- AUTHOR   :    Anatolij Sergyienko.
    -- DESCRIPTION    :    Fast  Fourier  Transform
                        for NN=16,
    --                  32,...,i.e. N=4,5,...20
    --    input    dates    are    in    the    order:
                        (Re(0),Im(0)),...,(Re(NN-
    --                  1),Im(NN-1)
    -- START precedes Re(0) to 1 cycle, and inits
FFT
    --    output    dates       are    in    the    same
order,accompanied by NUM
    -- output data is scaled by division to 2**PT
    ------------------------------------------------
    library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.std_logic_ARITH.all;
    use IEEE.math_real.all;
    use IEEE.MATH_COMPLEX.all;
    entity FFTbeh is
    generic(
     N:positive:=6;        --2**N              -длина
преобразования
     IFFT:natural:=0;      --0   -   прямое,   1   -
обратное преобразование
```

```
      PT:natural;            --масштаб     результата
=2**PT
      overs:integer:=0       --перекрытие 0,1 - без;
2 - 1/2 4 - 1/4...
      -- -1,-2 - 1/2 массива=0, те. для корреляции
и свертки,
      -- если IFFT и -2 то на выходе - склеивание
      соседних массивов    -- как при свертке
      );
     port (
      CLK: in STD_LOGIC;
      RST: in STD_LOGIC;
      START: in STD_LOGIC;          --           начало
накопления входного массива
      EDIN: in STD_LOGIC;           -- строб данных
      DIN_RE: in integer;           --  реальная  часть
данных
      DIN_IM: in integer;           --   мнимая   часть
данных
      DOUT_RE: out integer:=0;      --  реальная  часть
спектра
      DOUT_IM: out integer:=0;      -- мнимая часть
      NUM: out integer:=0;  --     номер    выходного
отсчета
      READY: out STD_LOGIC          --  начало  вывода
результатов
      );
     end FFTbeh;

     architecture FFT_int of FFTbeh is
      constant NN: integer:=2**N;
      type MEMOC is array (0 to NN)of complex;
      type  MEMOC_2  is  array  (0  to  NN/2-1  )of
complex;
      type MEMOR is array (0 to NN-1)of real;
      signal
RAM1,RAM2,RAMD,RAMD2:MEMOC:=(others=>(0.0,0.0));
      signal rdy,idatardy:STD_ULOGIC:='0';
      signal fl:bit;
```

```
     signal datact_I,datact_O:natural;

     procedure  GATHER_C(signal  CLK,START,EDIN:in
std_logic;
          --реальная и мнимая часть вх.данных

          signal DR,DI    : in integer;
          --сигнал готовности
          signal rdy : out std_logic;
          --номер данного
          signal datact   : inout natural;
          -- рабочий массив
          signal RAM :out MEMOC)
          is
          variable TR,TI  :real;
     begin
          wait until  CLK= '1' ;

          if EDIN = '1' then
               rdy <= '0';
               TR := real (DR);
               TI := real (DI);
               RAM(datact)<=(TR,TI);
               if datact < RAM'right then
                    datact<=datact+1;
               end if;

               if datact =RAM'right-1 then --##
                    rdy<='1';
               end if;
               if START='1' then
                    datact<=RAM'left;
               end if;
          end if;
     end procedure;

     --накопление с замещением 1/2 массива
```

```
     procedure                    GATHER_C1to2(signal
CLK,START,EDIN:in std_logic;
          signal DR,DI     : in integer;
          --реальная и мнимая часть вх.данных
          signal rdy : out std_logic;      --
сигнал готовности
          --номер данного
          signal datact    : inout natural;
          --к-т перекрытия массива
          overs      : in integer;
          signal RAM :out MEMOC)           --
рабочий массив
          is
          variable TR,TI:real;
     begin
          wait until  CLK= '1' ;

          if EDIN = '1' then
               --rdy<='0';
               TR:= real (DR);
               TI:= real (DI);
               RAM(datact)<=(TR,TI);
               if datact < RAM'right then
                    datact<=datact+1;
               end if;

               if          ((datact+1)          mod
(RAM'right/overs)) = 0 then
               --255+1 mod 64 =0
                    rdy<='1';
               end if;
               if START='1' then
                    datact<=RAM'left;
               end if;
          end if;
     end procedure;

     --накопление для корреляции 1 FFT - 1/2
массива =0
```

```
     procedure                GATHER_C_corr0(signal
CLK,START,EDIN:in std_logic;
          --реальная и мнимая часть вх.данных
          signal DR,DI: in integer;
          --сигнал готовности
          signal rdy: out std_logic;
          --номер данного
          signal datact: inout natural;
          --к-т перекрытия массива
          overs: in integer;
          -- рабочий массив
          signal RAM:out MEMOC)
          is
          variable TR,TI:real;
          variable startd,rdyd,rd: std_logic:='0';
     begin
          if clk='1' and clk'event then
               rdyd:=rd;
               startd:=START;
               rdy<='0';

               if EDIN = '1'  then
                    TR:= real (DR);
                    TI:= real (DI);

                    RAM(datact)<=(TR,TI);
                    datact<=(datact+1)        mod
(RAM'right/2);
                    if datact = RAM'right/2-2 and
rdyd='0' then
                         rdy<='1';
                         rd:='1';

                    end if;
               end if;
               if START='1'  then
                    --спад START  and startd='1'
                    datact<=RAM'left;
                    for i in 0  to RAM'right loop
```

```
                              -- RAM'right/2
                              RAM(i)<=(0.0,0.0);
                        end loop;
                  end if;
            end if;

      end procedure;

      --накопление с перекрытем 1/2 с умножением на
окно
      procedure                  GATHER_C_corr1(signal
CLK,START,EDIN:in std_logic;
            --реальная и мнимая часть вх.данных
            signal DR,DI: in integer;
            --сигнал готовности
            signal rdy: out std_logic;
            --номер данного
            signal datact: inout natural;
            --к-т перекрытия массива
            overs: in integer;
            --предыстория
            signal RAMD:inout MEMOC;
            -- рабочий массив
            signal RAM:out MEMOC)
            is
            variable TR,TI:real;
            variable startd,rdyd,rd: std_logic:='0';
            variable rami:MEMOC;
      begin
            if clk='1' and clk'event then
                  rdyd:=rd;
                  startd:=START;
                  rdy<='0';

                  if EDIN = '1'  then
                        TR:= real (DR);
                        TI:= real (DI);
                        --*(0.5+0.5*cos(MATH_2_PI*
--(real(datact)/real(RAM'right))));
```

```
                    RAM(datact+NN/2)<=(TR,TI);
                    RAMD(datact)<=(TR,TI);
                    datact<=(datact+1) mod
                    (RAM'right/2);
                    if datact = RAM'right/2-2 and
                    rdyd='0' then
                          rdy<='1';
                          rd:='1';
                    end if;
                    if datact =
                    RAM'right/2-1              then
                    RAMD(RAM'right/2             to
                    RAM'right-1)
                          <=RAMD(0 to RAM'right/2-
                    1);

                    for i in 0 to RAM'right/2-1
                    loop
                               -- RAM'right/2
                               RAMi(i):=RAMD(i);
                               -- *(0.5+0.5*
                               --
                          cos(MATH_2_PI*(real(i+1)
                          /--    real(RAM'right))-
                          MATH_PI));
                          end loop;
                          RAM(0 to RAM'right/2-1)
                          <=RAMi(0 to RAM'right/2-
                          1);
                    end if;
              end if;
              if START='1'  then
                    --спад START  and startd='1'
                    datact<=RAM'left;
              end if;
        end if;
        wait on clk;
end procedure;
```

```
    --накопление для корреляции 2 IFFT -1отсчет за
1 такт
     procedure                GATHER_C_corr2(signal
CLK,START,EDIN:in std_logic;
          --реальная и мнимая часть вх.данных

          signal DR,DI     : in integer;
          --сигнал готовности
          signal rdy: out std_logic;
          --номер данного
          signal datact: inout natural;
          --к-т перекрытия массива
          overs: in integer;
          -- рабочий массив
          signal RAM:inout MEMOC)

          is
          variable TR,TI:real;
          variable startd,rdyd,rd: std_logic:='0';
     begin
          if clk='1' and clk'event then
               rdyd:=rd;
               startd:=START;
               rdy<='0';

               --    if EDIN = '1'  then
               TR:= real (DR);
               TI:= real (DI);

               RAM(datact)<=(TR,TI);

               datact<=(datact+1)              mod
(RAM'right);
               if   datact   =   RAM'right-2   and
rdyd='0' then
                    rdy<='1';
                    rd:='1';
               end if;
               --end if;
```

```
                if START='1'  then
                      --спад START  and startd='1'
                      datact<=RAM'left;
                --    for i in RAM'right/2 to
                      RAM'right loop
                --          RAM(i)<=(0.0,0.0);
                --          end loop;
                end if;
           end if;
      end procedure;

      procedure  SCATTER_C(signal  CLK,START,EDIN:in
std_logic;
           --масштаб результата
           pt:natural;
           --входной массив
           signal RAM : in MEMOC;
           --счетчик данных
           signal datact    : inout natural;
           -- конец массива
           signal rdy : out std_logic;
           -- выходные данные
           signal DR,DI     : out integer    )
           is
           variable s :real:=real(2**pt);
           variable startd:std_logic:='0';
      begin
           --    wait until CLK='1';
           if CLK='1' and CLK'event then
                --    elsif EDIN='1'  then

                RDY<='0';
                DR<= integer(RAM(datact).RE/s);
                DI<= integer(RAM(datact).IM/s);
                if  datact< RAM'right then
                     datact<=datact+1;
                end if;
                if datact=RAM'right-1 then
                     RDY<='1';
```

```
                end if;
                if START='1' and startd='0' then
                      datact<=RAM'left;
                end if;
                startd:=start;
          end if;
     end procedure;

     --для свертки - выход
     procedure                SCATTER_C_conv(signal
CLK,START,EDIN:in std_logic;
          --масштаб результата
          pt:natural;
          --входной массив
          signal RAM :in MEMOC;
          --входной массив
          signal RAMD      :inout MEMOC;
          --счетчик данных
          signal datact    : inout natural;
          -- конец массива
          signal rdy : out std_logic;
          -- выходные данные
          signal DR,DI     : out integer    )
          is
          variable s:real:=real(2**pt);
          variable startd:std_logic:='0';
     begin

          if CLK='1' and CLK'event then

                if EDIN='1'  then
                      RDY<='0';

                      --
                +RAMD(datact+RAM'right/2).RE/s);

                      DR<=integer(RAMD(datact).RE/s
                );
```

```
                    --
                +RAMD(datact+RAM'right/2).IM/s);

     DI<=integer(RAMD(datact).IM/s);
                    --if     datact<  RAM'right/2
then-- /2
                    datact<=(datact+1)        mod
(RAM'right);--/2
                    --end if;
                    if     datact=RAM'right*3/4-1
then
                         RDY<='1';
                    end if;
                    if  START='1'  and  startd='0'
then
                         datact<= RAM'right/4;  -
-
                         RAMD<=RAM;
                         -  RAMD2<=RAMD;
                    end if;
               end if;
               startd:=start;
          end if;
     end procedure;

     procedure FFT(N:positive;  --          размер
преобразования =2**N
          signal rdy:in std_logic;   --     запуск
преобразования
          signal RAM_I: in MEMOC;    --    входной
массив данных
          signal RAM_O:out MEMOC)    --   выходной
массив спектров
          is
          constant NN:positive:=2**N;
          -- массив вращающих коэффициентов
          variable TWIDDLE:MEMOC_2;
          variable RAM:MEMOC;        --    рабочий
массив
```

```
          variable a,b,c,d,w: complex;
          variable al:real;
          variable
base,itera,datact,twiddlect,twiddleinc: natural;
          variable delta:integer:=1;
          variable   addrf:   std_LOGIC_VECTOR(N-1
downto 0);
          variable   addri:   std_LOGIC_VECTOR(N-1
downto 0);
     begin
          --    формирование    таблицы    вращающих
коэффициентов
          for i in 0 to NN/2-1 loop
               al:=
(MATH_PI*real(2*i))/real(NN);
               if IFFT=0 then
                    -- для прямого БПФ
                    TWIDDLE(i):=(COS(al),-
SIN(al));
               else
                    -- для обратного БПФ
                    TWIDDLE(i):=(COS(al),
SIN(al));
               end if;
          end loop;

          loop
               -- начало БПФ
               wait until rdy='1';
               --  двоично-инверсная  перестановка
входных данных
               --   вектор   адреса   для   прямого
          порядка
               addrf:=(others=>'0');
               datact:=0;
               for i in 1 to NN loop
                    --bit revers
                    for ind in 0 to N-1 loop
```

```
                    -- вектор инверсного
              адреса
                    addri(ind):=addrf(N-1-
ind);
              end loop;

              RAM(CONV_INTEGER(unsigned(add
        ri))):=
                    RAM_I(CONV_INTEGER(unsig
              ned(addrf)));
              addrf:=unsigned(addrf)+1;
        end loop;

        -- собственно БПФ
        itera:=0;
        delta:=1;
        twiddleinc:=0;
        for itera in 1 to N loop --начало
итерации
              base:=0;
              twiddlect:=0;
              for butterfly in 0 to NN/2 -
1 loop
                    -- начало базовой
              операции
                    a:=RAM(base);
                    base:=base+delta;
                    b:=RAM(base);
                    w:=TWIDDLE(twiddlect);
                    -- собственно бабочка
                    c:=a + b * w;
                    d:=a - b * w;
                    RAM(base-delta):=c;
                    RAM(base):=d;

                    -- конец базовой
              операции
                    base:= base + delta;
```

```
                              --            модификация
параметров базовой операции
                              base:= base + delta;
                              if base >= NN then
                                    base:=base-NN+1;

      twiddlect:=twiddlect+twiddleinc;
                              end if;
                        --конец итерации

                        end loop;
                        --    модификация    параметров
итерации
                        delta:=delta*2;
                        if itera=1 then
                              twiddleinc:=NN/4;
                        else

      twiddleinc:=twiddleinc/2;
                        end if;
                  end loop;
                  -- результаты БПФ
                  RAM_O<=RAM;
            end loop;
      end procedure;

    ---------- БПФ  -------------------
    begin
      INPUT:      if overs=0  generate
            GATHER_C(CLK,START,EDIN,
            DR=>DIN_Re,DI=>DIN_IM,

      rdy=>idatardy,datact=>datact_I,RAM=>RAM1);
      end generate;

      Inp_1to2:  if overs>1 generate
            INPUT:      GATHER_C1to2(CLK,START,EDIN,
            DR=>DIN_Re,DI=>DIN_IM,
```

```
      rdy=>idatardy,datact=>datact_I,overs=>overs,R
AM=>RAM1);
      end generate;

      Inp_corr1: if overs=-1 generate
            INPUT:
      GATHER_C_corr1(CLK,START,EDIN,
            DR=>DIN_Re,DI=>DIN_IM,
            rdy=>idatardy,datact=>datact_I,overs=>
            overs,RAMD=>RAMD2,RAM=>RAM1);
      end generate;

      Inp_corr2: if overs=-2 generate
            INPUT:
      GATHER_C_corr2(CLK,START,EDIN,
            DR=>DIN_Re,DI=>DIN_IM,

      rdy=>idatardy,datact=>datact_I,overs=>overs,R
AM=>RAM1);
      end generate;

      SPECTRUM:FFT(N,idatardy,RAM_I=>RAM1,RAM_O=>RA
M2);

      Outp: if overs>=-1  generate
            OUTPUT:
SCATTER_C(CLK,start=>idatardy,edin=>EDIN,pt=>PT,
            RAM=>RAM2,DATACT=>datact_o,
            rdy=>rdy,DR=>DOUT_RE,DI=>DOUT_IM);
      end generate;

      Outp_conv: if overs=-2  generate

      OUTPUT:SCATTER_C_conv(CLK,start=>idatardy,edi
n=>
                  EDIN,pt=>PT,
            RAM=>RAM2,RAMD=>RAMD,DATACT=>datact_o,
            rdy=>rdy,DR=>DOUT_RE,DI=>DOUT_IM);
```

```
 end generate;

 DEL: process(CLK) begin
      if CLK='1' and CLK'event  then
           if  EDIN='1' or overs=-1 then
                READY<=idatardy;
                num<= datact_o;

           end if;
      end if;
 end process;
end FFT_int;
```

Д.2. Модуль FFTtst.VHD

```
-----------------------------------------------
-- File     : FFTtst.vhd
-- Design   :  Library for DSP
-----------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_ARITH.all;
use IEEE.math_real.all;
use IEEE.MATH_COMPLEX.all;

entity FFTtst is
end FFTtst;

architecture FFTtst of FFTtst is

 component fftbehgeneric(
      --2**N -длина преобразования
      N          :positive:=6;
      --0    -    прямое,    1    -    обратное
 преобразование
      IFFT       :natural:=0;
      --масштаб результата =2**PT
      PT         :natural;
```

```
            --перекрытие 0,1 - без; 2 - 1/2 4 -
      1/4...
            Overs :integer:=0
            --  -1  -  1/2  массива=0,  те.  для
корреляции и свертки
            );
      port (
            CLK         : in STD_LOGIC;
            RST         : in STD_LOGIC;
            -- начало накопления входного массива
            START : in STD_LOGIC;
            -- строб данных
            EDIN        : in STD_LOGIC;
            -- реальная часть данных
            DIN_RE      : in integer;
            -- мнимая часть данных
            DIN_IM      : in integer;
            -- реальная часть спектра
            DOUT_RE     : out integer:=0;
            -- мнимая часть
            DOUT_IM     : out integer:=0;
            -- номер выходного отсчета
            NUM         : out integer:=0;
            -- начало вывода результатов
            READY : out STD_LOGIC
            );
     end    component;

     constant n:positive:=6;
     signal       RST,CLK,START,  EDIN:
STD_LOGIC:='0';
     signal       DIN,DOUT_re,DOUT_IM,Magn,num
:integer;
     signal       READY :STD_LOGIC;

     begin

      CLK <=not CLK after 10 ns;
      RST <= '0' ,'1' after 5 ns, '0' after 30 ns;
```

```
      START <= '0' after 0 ns, '1' after 50 ns, '0'
after 75 ns;
           DIN<= 0, 4000 after 115 ns,0000 after
255 ns;

      EDIN<='1';

      U1: FFTbeh

      generic map(
      N=>N,
      pt=>1,
            Ifft=>0
      )

      port map(
            CLK=>CLK,
            RST=>RST,
            START=>START,
            EDIN=>EDIN,
            DIN_re=>DIN,
            DIN_IM=>0,
            DOUT_re=>DOUT_re,
            DOUT_im=>DOUT_im,
            num=>num,
            READY=>READY
            );
    end FFTtst;
```

Д.3. Модуль SQ_GEN.VHD

```
-----------------------------------------------
-- Title       : SQ_GEN
-- Design      : Library for DSP
-----------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity SQ_GEN is
 generic(
       --период выходной синхросерии
       tclk       : time:=10 ns;
       -- период импульсов
       G_period   : positive:=12;
       -- ширина импульса по уровню 1
       H_len : positive:=4;
       -- период имульсов разрешения
       E_period   : positive:=2;
       -- период импульсов START
       start_period: positive:=64;
       -- амплитуда импульсов
       magn       : integer:=10);

 port (
       RST        : in STD_LOGIC;
       CLK        : out STD_LOGIC;
       -- импульс начала
       START : out STD_LOGIC;
       -- собственно импульсы
       DATA  : out INTEGER;
       -- сигнал разрешения (замедление)
       ENA        : out STD_LOGIC
       );
 begin assert H_len
            <  G_period  report  "Некорректная
       скважность"
 severity error;
end SQ_GEN;

architecture BEH of SQ_GEN is
 signal starti,c       : std_logic:='0';
 signal cte,ctg,ctstart     : natural;
begin
 c <=not c after tclk/2.0 ;
 process(c,RST,ctg)
 begin
       if RST='1' then
```

```
                cte<=0;
                ctg<=0;
                ctstart<=0;
                STARTi<='0';
          elsif c='1' and c'event then
                if cte = E_period-1 then
                      cte<=0;
                      if ctg = G_period-1 then
                            ctg<=0;
                      else
                            ctg<=ctg+1;
                      end if;
                      if  ctstart  =  start_period-1
then
                            ctstart<=0;
                            STARTi<='1';
                      else
                            ctstart<=ctstart+1;
                            STARTi<='0';
                      end if;
                else
                      cte<=cte+1;
                end if;
          end if;
          if ctg<=H_len and ctg/=0 then
                DATA<=magn after 1 ns;
          else
                DATA<=0;
          end if;
     end process;

     ENA<='1' when cte=0 else '0';
     START<=starti when cte=0 else '0';
     CLK<=c when RST ='0' else '0';
   end BEH;
```

Д.4. Модуль TO_POLAR.VHD

```
-------------------------------------------
--
-- Title          : TO_POLAR
-- Design         : Library for DSP
-------------------------------------------
library IEEE;
use IEEE.MATH_REAL.all;
entity TO_POLAR is
 port(
       D_RE  : in INTEGER;
       D_IM  : in INTEGER;
       MAGN  : out INTEGER;
       PHASE      : out INTEGER
       );
end TO_POLAR;

architecture TO_POLAR of TO_POLAR is
begin
 process(D_RE,D_IM)
 variable r,i:real;
 variable m:integer;
 begin
       r:=real( D_RE);
       i:=real( D_IM);
       if  r=0.0  and  i=0.0  then  --  to  prevent
division 0/0
            r:=0.5;
       end if;
       m:=(255+1) mod 64;
       MAGN<=integer(SQRT(r*r+i*i));

 PHASE<=integer(32768.0*ARCTAN(i,r)/MATH_PI);
 end process;
end TO_POLAR;
```

Гузик Вячеслав Филиппович,
Золотовский Виктор Евдокимович,
Беспалов Дмитрий Анатольевич,
Ляпунцова Елена Вячеславовна

Проектирование проблемно-ориентированных вычислительных систем

Учебное пособие

Ответственный за выпуск	Беспалов Д.А.
Редактор	Селезнева Н.И.
Корректор	Селезнева Н.И.

ЛР № 020565 от 23.06.1997 г. Подписано к печати 27.12.2012 г.

Формат $60\times84\,{}^{1}\!/_{16}$. Бумага офсетная.

Печать офсетная. Усл. п.л. – 7,9.

Уч.–изд. л. - 7,7

Заказ № . Тираж 100 экз.

“С”

Издательство Технологического института
Южного федерального университета
ГСП 17А, Таганрог, 28, Некрасовский, 44

Типография Технологического института
Южного федерального университета
ГСП 17А, Таганрог, 28, Энгельса, 1

174