МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)

БГТУ.СМК-Ф-4.2-К5-01

Факультет И (шифр) Информационные и управляющие системы (наименование)

Кафедра И4 (шифр) Радиоэлектронные системы управления (наименование)

Дисциплина Проектирование цифровых систем на ПЛИС

КУРСОВАЯ РАБОТА

на тему

Реализация цифрового фильтра для алгоритма комплексной обработки информации в инерциальной навигационной системе

Выполнил студент группы И4М31
Шевцова Ю.О.
Фамилия И.О.

РУКОВОДИТЕЛЬ

Ширшов А.Д. _____
Фамилия И.О. Подпись

Оценка _____

«_____» _____ 2018 г.

САНКТ-ПЕТЕРБУРГ

2018 г.

Содержание

Введение 3

1 Цифровая обработка сигналов 4

1.1 Цифровая фильтрация 4

1.2 Цифровые фильтры 6

1.3 Постановка задачи 7

1.4 Фильтр с конечной импульсной характеристикой 8

1.5 Свойства КИХ-фильтров 8

2 Реализация фильтра на ПЛИС 10

2.1 Требования к ПЛИС для реализации фильтра 10

2.3 Программная реализация фильтра 12

3 Моделирование работы фильтра 21

Заключение 29

Список использованных источников 30

Введение

Подавляющее большинство сигналов, обрабатываемых современными техническими системами, так, или иначе, имеет цифровое представление. Применение цифровой обработки сигнала улучшает помехозащищенность канала связи, предоставляет бесконечные возможности кодирования информации.

Системы, выполняющие обработку дискретных сигналов, называются цифровыми фильтрами. Цифровой фильтр обладает рядом существенных преимуществ. Сюда относятся, например, высокая стабильность параметров, возможность получать самые разнообразные формы АЧХ и ФЧХ. Цифровые фильтры не требуют настройки и легко реализуются на ЭВМ программными методами.

Выделяют следующие виды цифровых фильтров: фильтры частотной селекции (к ним относятся фильтры сигналов с конечной и бесконечной импульсной характеристикой), оптимальные (квазиоптимальные) фильтры, адаптивные и эвристические фильтры. Фильтры сигналов с конечной импульсной характеристикой выделяются устойчивостью по определению и простотой поведения и математического описания.

Цель данной работы: реализовать цифровой фильтр с использованием ПЛИС, который применяется в алгоритме комплексной обработки информации инерциальной радионавигационной системы.

1 Цифровая обработка сигналов

1.1 Цифровая фильтрация

Обработка сигналов длительное время используется для управления и преобразования аналоговых или цифровых сигналов. Цифровая обработка сигналов применяется во многих областях от передачи данных, речи, аудио или биомедицинской обработки сигналов до приборостроения и робототехники. Системы ЦОС заменили традиционные системы аналоговой обработки сигналов в связи с тем, что, например, первые не восприимчивы к изменениям температуры, допуску компонентов, не подвержены старению. ЦОС тесно связана с представлением сигналов в виде последовательностей чисел, т. е. для начала необходимо преобразовать заданный аналоговый сигнал в исходную последовательность чисел, затем эта последовательность преобразуется в новую по некоторому алгоритму. Результирующий аналоговый сигнал формируется из вычисленной последовательности [1]. Данный процесс представлен на рисунке 1.

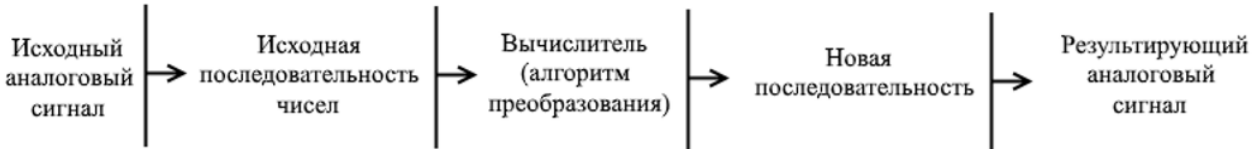

Рисунок 1 – Этапы цифровой обработки сигналов

Для получения цифрового сигнала из аналогового необходимо произвести дискретизацию по времени, т. е. взять значения сигнала через равные промежутки времени, называющиеся периодом дискретизации. Период дискретизации T определяет частоту дискретизации $f_Д$, ($T=1/f_Д$), т. е., чем меньше период, тем выше частота дискретизации и тем точнее цифровое представление аналогового сигнала.

Процесс проектирования цифровых фильтров состоит из тех же этапов, что и процесс проектирования аналоговых фильтров. Сначала формулируются требования к желаемым характеристикам фильтра, по которым затем

рассчитываются параметры фильтра. Амплитудная и фазовая характеристики формируются аналогично аналоговым фильтрам. Ключевое различие между аналоговым и цифровым фильтрами заключается в том, что, вместо вычисления величин сопротивлений, емкостей индуктивностей для аналогового фильтра, рассчитываются значения коэффициентов для цифрового фильтра. Иными словами, в цифровом фильтре числа заменяют физические сопротивления и емкости аналогового фильтра. Эти числа являются коэффициентами фильтра, они постоянно находятся в памяти и используются для обработки (фильтрации) дискретных данных, поступающих от АЦП.

Цифровой фильтр, работающий в реальном масштабе времени, оперирует с дискретными по времени данными в противоположность непрерывному сигналу, обрабатываемому аналоговым фильтром. При этом очередной отсчет, соответствующий отклику фильтра, формируется по окончании каждого периода дискретизации. Вследствие дискретной природы обрабатываемого сигнала, на отсчеты данных зачастую ссылаются по их номерам, например, отсчет 1, отсчет 2, отсчет 3 и т.д.

1.2 Цифровые фильтры

Цифровой фильтр – это линейная дискретная система, которая по алгоритму, описываемому разностным уравнением, преобразует входную последовательность чисел. ЦФ отображается заданной структурой, реализованной аппаратно, программно или программно-аппаратно. Линейность системы означает, что она отвечает условиям аддитивности и однородности. Несмотря на то, что ЦФ может быть нелинейным и нестационарным, наиболее популярные – линейные стационарные ЦФ в силу простоты их поведения и математического описания. Линейные стационарные фильтры взаимодействуют со своими входными сигналами посредством, так называемой линейной свёртки, обозначаемой $y = h * x$, где h – импульсная характеристика фильтра, x – входной сигнал, y – свёрнутый выход (реакция). Процесс линейной свёртки формально определён как:

$$y(n) = x(n) * h(n) = \sum x(k)h(n-k) = \sum h(k)x(n-k)\ kk \qquad (1)$$

Различают два типа линейных цифровых фильтров: фильтр сигналов с бесконечной импульсной характеристикой (БИХ), который имеет обратную связь, т. е. является рекурсивным, и фильтр сигналов с конечной импульсной характеристикой (КИХ), который является преимущественно нерекурсивным.

Разностные уравнения цифровых фильтров определяются следующим выражением:

$$y(k) = \frac{1}{a_0} \cdot \left(\sum_{m=0}^{k} b_m \cdot x(k-m) - \sum_{m=1}^{k} a_m \cdot y(k-m)\right) \qquad (2)$$

где y(k) - отсчеты на выходе фильтра, x(k) - входные отсчеты, b_m и a_m - коэффициенты числителя и знаменателя передаточной характеристики фильтра соответственно. Если все коэффициенты a_m кроме a_0 равны нулю, то такой фильтр называется КИХ-фильтром, а если хотя бы один коэффициент a_m помимо a_0 отличен от нуля, то такой фильтр называется БИХ-фильтром.

Согласно выражению (2), сигнал на выходе фильтра зависит от задержанного входного сигнала, а также от предыдущих отсчетов на выходе,

поэтому для реализации фильтра потребуются линии задержки. Согласно z-преобразованию, задержка на один отсчет соответствует умножению образа на z^{-1}. Также потребуются умножители на постоянные коэффициенты b_m и a_m и сумматоры. На рисунке 2 показаны обозначения основных блоков для построения цифрового фильтра.

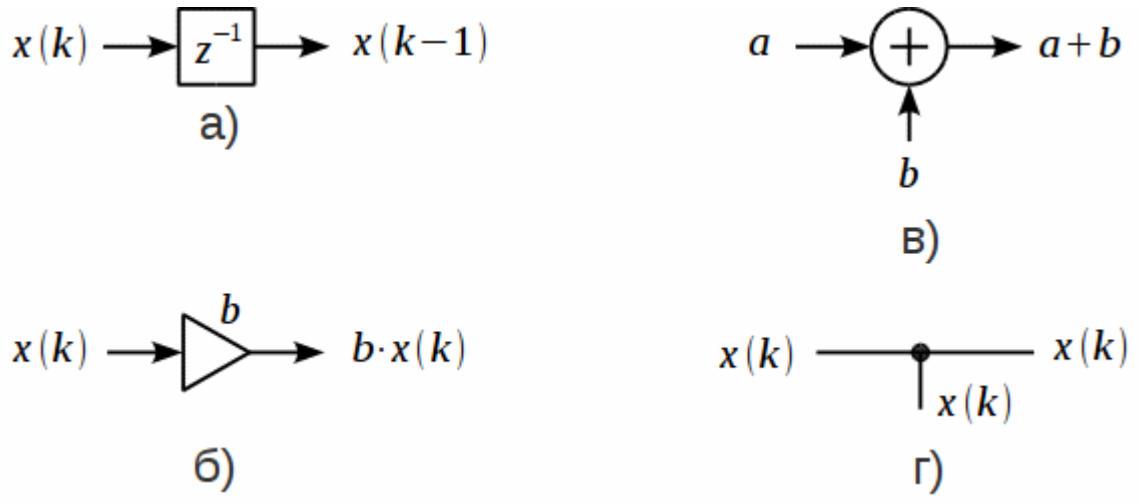

Рисунок 2 - Обозначения блоков цифрового фильтра

На рисунке 2 обозначены: а) - линия задержки, б) - умножитель на константу, в) - сумматор и г) - разветвление.

1.3 Постановка задачи

В данной работе необходимо реализовать цифровой фильтр на ПЛИС. Фильтр является частью алгоритма комплексной обработки информации предназначенного для улучшения характеристик определения координат и составляющих скорости летательного аппарата. Одним из основных требований к фильтру является обеспечение линейной ФЧХ и быстродействие. Такой возможностью обладает фильтр сигналов с конечной импульсной характеристикой.

1.4 Фильтр с конечной импульсной характеристикой

Разностное уравнение КИХ фильтра не содержит рекурсивной части:

$$y(k) = \frac{1}{a_0} \cdot \left(\sum_{m=0}^{k} b_m \cdot x(k-m) \right) \quad (3)$$

Выражение (3) получается из выражения (2) при a_0=1 и a_m=0, m>0.

Структурная схема нерекурсивного КИХ-фильтра показана на рисунке 3.

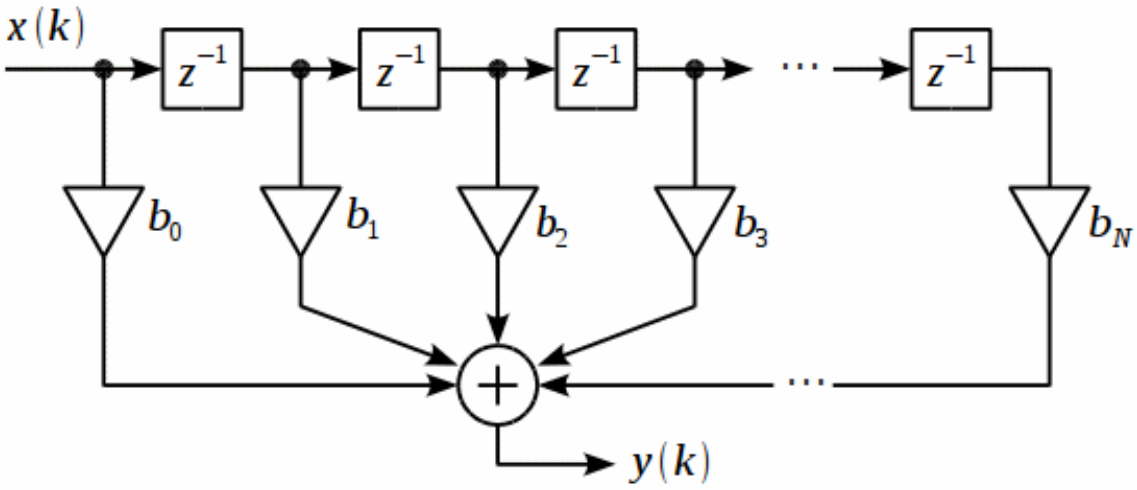

Рисунок 3 - Структурная схема нерекурсивного КИХ-фильтра

КИХ фильтр порядка N содержит N линий задержки и N+1 коэффициент. Если коэффициент b_0=1, то получим КИХ фильтр порядка N у которого умножение на b_0=1 будет тривиальным. Импульсная характеристика КИХ-фильтра всегда конечна и полностью совпадает с коэффициентами фильтра.

1.5 Свойства КИХ-фильтров

КИХ-фильтры отличаются возможностью обеспечить строго линейную ФЧХ (ЛФЧХ) и устойчивостью по определению. Линейность обеспечивается, если $h(n)$ симметрична [$h(n) = h(N–1–n)$] или антисимметрична [$h(n) = –h(N–1–n)$], где ось симметрии/антисимметрии ИХ $h(n)$ проходит через точку $n = R/2$. При этом линейность ФЧХ рассматривается с точностью до скачков на π (скачки происходят в точках, где АЧХ равна нулю). Четыре вида КИХ-фильтров с ЛФЧХ описаны на рисунке 4. Свойства симметрии/антисимметрии,

присущие КИХ-фильтрам с ЛФЧХ, также могут быть использованы для уменьшения необходимого числа умножителей.

Тип КИХ-фильтра	ЛФЧХ (с точностью до скачков на π)	ЦФ
Тип 1: порядок R – чётный; ИХ h(n) – симметричная	$-\frac{R}{2}\hat{w}$	ФНЧ, ФВЧ, ПФ, РФ
Тип 2: порядок R – нечётный; ИХ h(n) – симметричная	$-\left(\frac{R-1}{2}+\frac{1}{2}\right)\hat{w}$	ФНЧ, ПФ
Тип 3: порядок R – чётный; ИХ h(n) – антисимметричная	$\frac{\pi}{2}-\frac{R}{2}\hat{w}$	ПФ, ЦПГ, ЦД
Тип 4: порядок R – нечётный; ИХ h(n) – антисимметричная	$\frac{\pi}{2}-\left(\frac{R-1}{2}+\frac{1}{2}\right)\hat{w}$	ФВЧ, ПФ, ЦПГ, ЦД

Рисунок 4 - Виды КИХ-фильтров с ЛФЧХ

2 Реализация фильтра на ПЛИС

2.1 Требования к ПЛИС для реализации фильтра

Программируемая логическая интегральная схема (ПЛИС, англ. programmable logic device, PLD) — электронный компонент (интегральная микросхема), используемый для создания конфигурируемых цифровых электронных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задаётся посредством программирования (проектирования). Для программирования используются программатор и IDE (отладочная среда), позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках описания аппаратуры: Verilog, VHDL, AHDL и др.[3]

ПЛИС представляют собой элементную базу, совмещающую в себе гибкость заказных БИС и доступность «жёсткой» логики, а также возможность настройки самим пользователем на выполнение заданных функций. Для ПЛИС характерны высокое быстродействие, низкая стоимость, низкая потребляемая мощность, многократность перепрограммирования, значительные функциональные возможности, высокая скорость разработки даже сложных проектов. Для того, чтобы реализация алгоритмов ЦОС была возможна, ПЛИС должна содержать набор умножителей и достаточное количество памяти для хранения коэффициентов. Наиболее популярными производителями являются компании Altera и Xilinx. Альтернативой применения ПЛИС являются DSP общего назначения, которые оказываются на порядки дешевле, чем мощные ПЛИС, хотя существуют и бюджетные семейства ПЛИС, такие как Altera Cyclone II и Xilinx Spartan [6].

Преимуществом ПЛИС является возможность многоканальной обработки данных или многоступенчатой фильтрации при работе с целочисленными данными, они позволяют эффективно реализовать сложные параллельные алгоритмы даже на микросхемах относительно недорогих семейств [6].

В среде проектирования для ПЛИС Altera Quartus II есть мегафункции для CIC и FIR фильтров:

- CIC фильтр — это интегрально гребенчатый фильтр (cascaded integral-comb filter).
- FIR фильтр – фильтр с конечной импульсной характеристикой (finite impulse response filter).

Каждая из мегафункций имеет очень много разных параметров.

Первым шагом необходимо запустить Quartus II Megafunction Wizard для создания модуля CIC фильтра. Нужно задать требуемые параметры CIC фильтра и Wizard сам создаст несколько дизайн файлов, модели для симуляции, и прочее. Среди созданных файлов будет файл с расширением *.m – это программа для MATLAB, которая должна помочь рассчитать коэффициенты компенсационного FIR фильтра. Вторым шагом необходимо запустить Megafunction Wizard для FIR фильтра и внести туда полученные ранее коэффициенты.

Но, к сожалению, имеющиеся мегафункции Quartus доступны только для ознакомления. Они работают ограниченное время. Требуется приобретение полной версии САПР, чтобы применять FIR без ограничений, поэтому необходимо использовать сторонние приложения и программы для симуляции Verilog проектов. Такой возможностью обладает пакет программ Icarus Verilog.

Icarus Verilog — компилятор языка описания аппаратуры Verilog. Он поддерживает версии 1995, 2001 и 2005, частично SystemVerilog и некоторые расширения. Используется для симуляции и верификации проектов. Кроме того, в версиях с 0.2 по 0.8 мог использоваться для синтеза (в формат XNF), для ПЛИС Xilinx[4].

Существует также среда симуляции ModelSim компании Mentor Graphics (или ModelSim-Altera Edition), которая значительно превосходит по

возможностям Icarus Verilog, но для достижения поставленной цели достаточно будет возможностей Icarus Verilog.

2.3 Программная реализация фильтра

Согласно рисунку 3, структура реализуемого фильтра выглядит следующим образом:

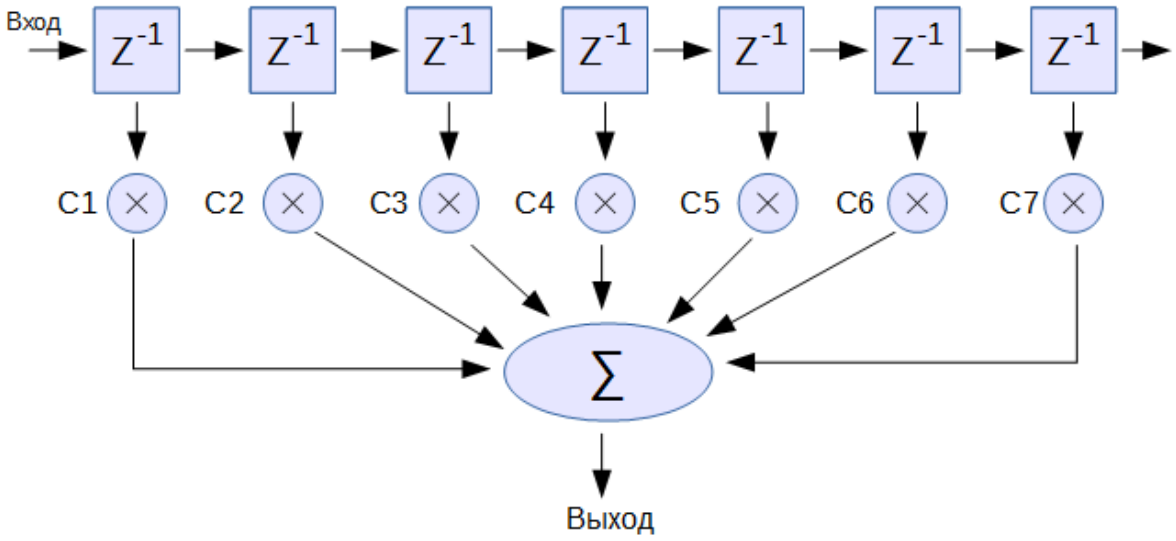

Рисунок 5 – Структура КИХ-фильтра

Здесь блоки Z^{-1} - это линии задержки. На практике цепочка блоков Z^{-1} - это просто последовательность регистров для хранения данных. По каждому фронту тактовой частоты, которая и является частотой дискретизации, входные данные движутся от первого регистра к последующим. Фильтр может содержать цепочку регистров произвольной длины. Дополнительно, от каждого блока Z^{-1} идет передача данных на умножитель. Каждый из умножителей умножает свои собственные данные на некоторую константу. Потом, все результаты умножения складываются и получается отфильтрованный сигнал. Необходимо рассчитать длину фильтра и коэффициенты для умножителей. От коэффициентов зависит характеристика фильтра.

КИХ-фильтр имеет несколько важных особенностей:

- его поведение устойчиво и предсказуемо;

Если входные данные 16-ти битные и коэффициенты для умножителей 16-ти битные, то после умножения результат становится 32-х битный. Далее идет суммирование. Сложение двух 32-х битных чисел дает 33-х битное число. Зная длину фильтра можно точно сказать, какой разрядности будет результат. Например фильтр длиной 4 будет на выходе иметь 34-х битное число. Математические операции внутри фильтра не накапливают ошибку, не приводят к переполнению результата.

- можно реализовать очень быстродействующий фильтр, так как в его алгоритме данные движутся только вперед.

Поскольку данные движутся только вперед, то довольно просто использовать преимущества конвейерной обработки (pipeline). Например, пока умножители умножают очередные данные из задерживающих регистров, сумматор может вычислять сумму от предыдущих результатов умножителей. И сами умножители могут использовать внутренний pipeline и сумматор может использовать внутренний pipeline.

Для конкретизации задачи возьмем для реализации два фильтра:

1. Полосовой фильтр. Частота дискретизации 20МГц. Пусть фильтром нужно подавлять частоты ниже 1МГц и выше 4 МГц.
2. Фильтр нижних частот. Частота дискретизации 20МГц. Пусть фильтром на частотах выше 2 МГц нужно значительно подавить все частоты, не менее -40Дб.

Тогда коэффициенты для умножителей полосового фильтра[1]: -801, -1026, -210, 1914, 4029, 3905, 330, -5174, -8760, -7040, -152, 7700, 11130, 7700, -152, -7040, -8760, -5174, 330, 3905, 4029, 1914, -210, -1026, -801. Всего коэффициентов 25 – это длина фильтра. АЧХ полосового фильтра представлена на рисунке:

[1] Рассчитываются с помощью программы TFilter

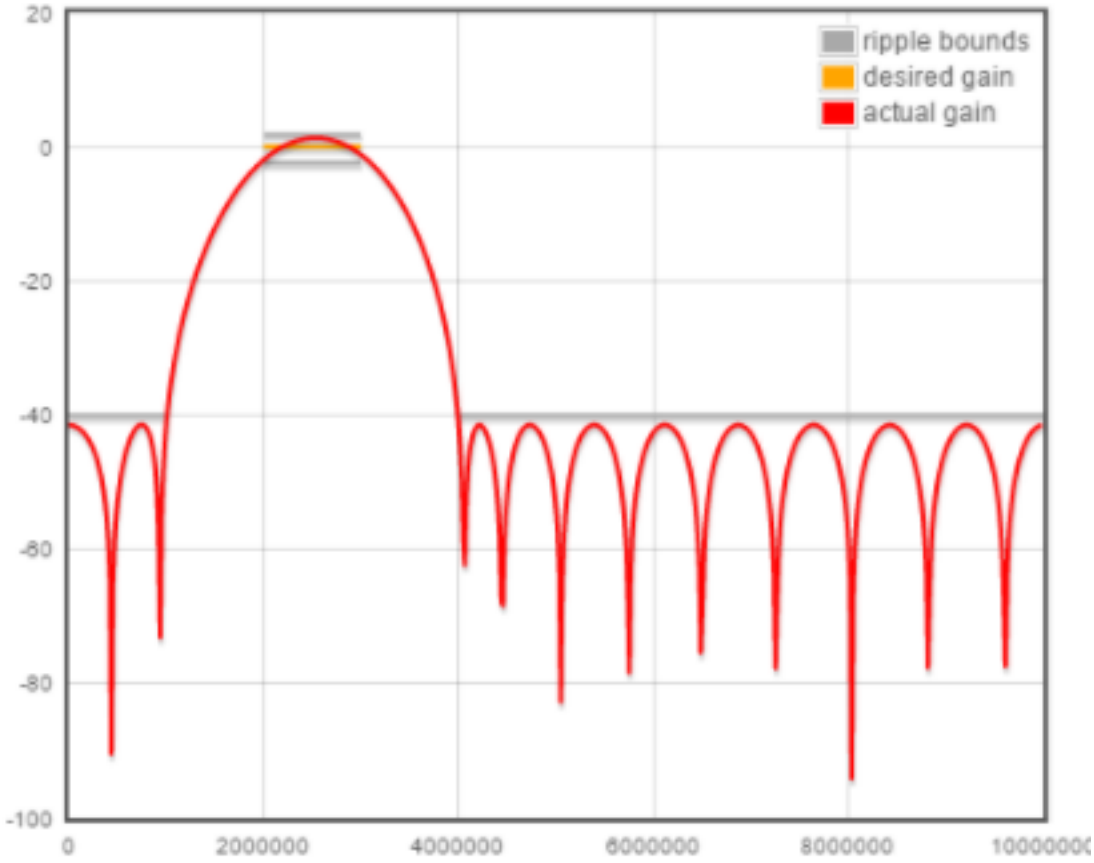

Рисунок 6 – АЧХ полосового фильтра

Коэффициенты фильтра нижних частот: -510, -520, -625, -575, -287, 306, 1232, 2467, 3927, 5477, 6948, 8162, 8962, 9241, 8962, 8162, 6948, 5477, 3927, 2467, 1232, 306, -287, -575, -625, -520, -510. Длина фильтра – 27. АЧХ фильтра нижних частот:

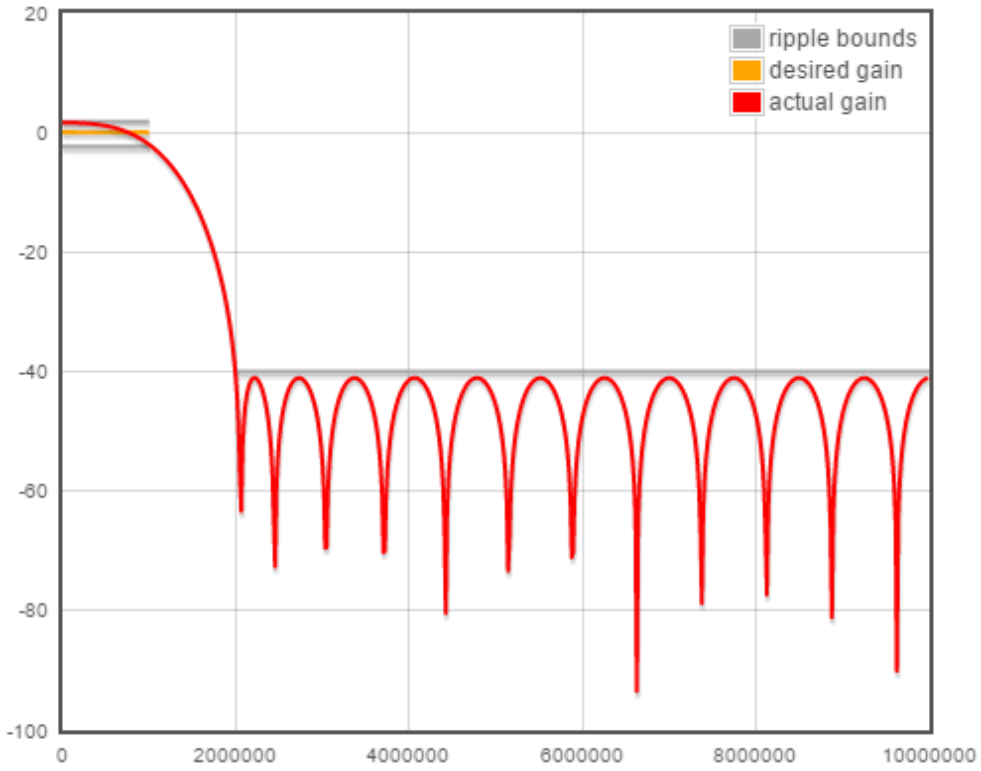

Рисунок 7 – АЧХ фильтра нижних частот

Так как коэффициенты были рассчитаны по заданным, для примера, двум фильтрам, необходимо организовать параметрический модуль, чтобы его можно было легко настраивать для других данных фильтров.

Параметры модуля фильтра - это его длина, то есть количество линий задержки, разрядность входных данных и разрядность коэффициентов для умножителей.

Если разрядность входных данных - это IWIDTH, а число регистров в цепочке линии задержки - это TAPS, то код выглядит следующим образом:

```
module fir ( clk, coefs, in, out );
parameter IWIDTH = 16; //input data (signal) width
parameter TAPS = 2; //number of filter taps
input wire clk;
input wire [IWIDTH-1:0]in;
genvar i;
generate
  for( i=0; i<TAPS; i=i+1 )
  begin:tap
    //make tap register chain
    reg [IWIDTH-1:0]r=0;
    if(i==0)
    begin
```

```
      //1st tap takes signal from input
      always @(posedge clk)
        r <= in;
    end
    else
    begin
      //tap reg takes signal from prev tap reg
      always @(posedge clk)
        tap[i].r <= tap[i-1].r;
    end
end
endgenerate
endmodule
```

Конструкция generate-endgenerate языка Verilog позволяет динамически создать нужное число регистров и присоединять их как надо в цепочку:

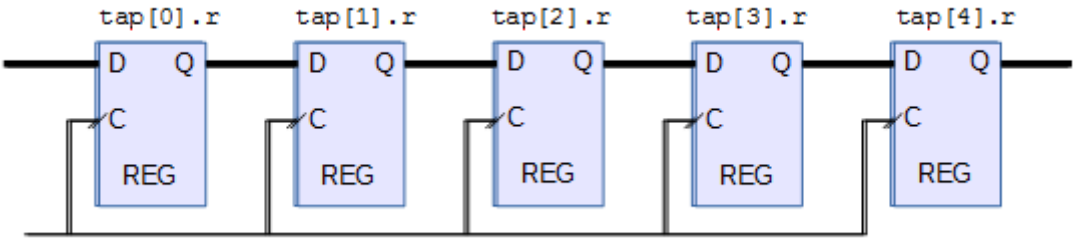

Рисунок 8 – Схема соединения регистров

Таким образом, длина фильтра может быть задана через параметр модуля Verilog. Аналогичным образом, внутри цикла for() внутри generate-endgenerate создаются умножители:

```
genvar i;
generate
  for( i=0; i<TAPS; i=i+1 )
  begin:tap
    ........
    //get tap multiplication constant coef
    wire [CWIDTH-1:0]c;
    assign c = coefs[((TAPS-1-i)*32+CWIDTH-1):(TAPS-1-i)*32];
    //calculate multiplication and fix result in register
    reg [MWIDTH-1:0]m;
    always @(posedge clk)
      m <= $signed(r) * $signed( c );
```

Таким образом, дополнительно к регистрам линии задержки будут сформированы и подключены умножители со своими коэффициентами:

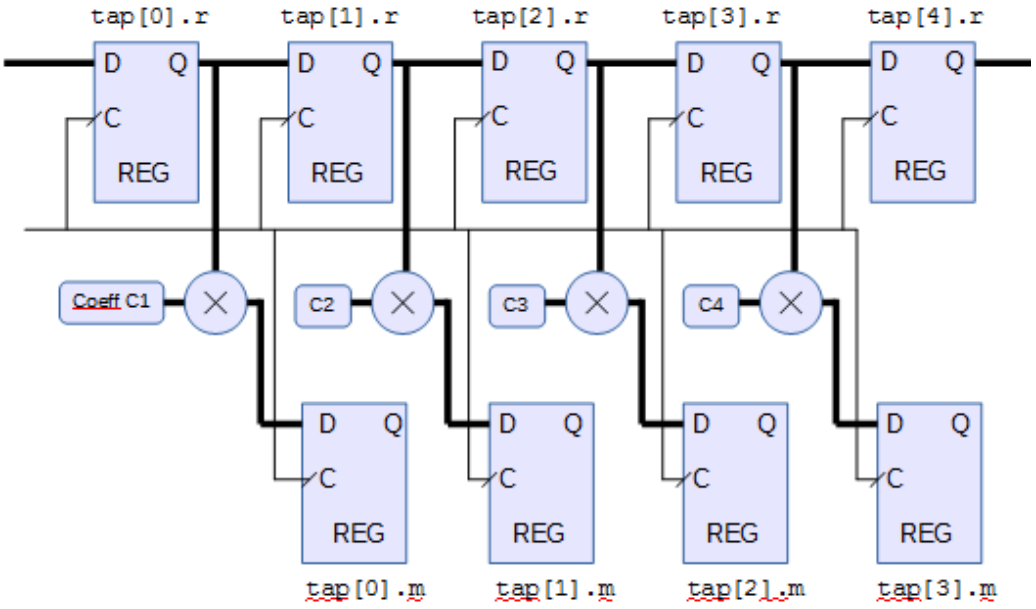

Рисунок 9 – Схема подключения умножителей с коэффициентами

Коэффициенты для умножителей можно передавать в модуль объединенные в шину {...} как входной сигнал:

```
fir #( .TAPS(27) ) fir_lp_inst(
.clk(tb_clk),
.coefs( {
  -32'd510,
  -32'd520,
  .......
   32'd575,
   32'd625,
  -32'd520,
  -32'd510
  } ),
.in(),
.out()
);
```

Здесь каждый коэффициент фильтра занимает в шине фиксированное число бит 32. Но из этих 32х бит модуль выберет нужное объявленное число бит для коэффициентов. Например, parameter CWIDTH = 16; от количества бит в коэффициенте для умножения зависит в конечном итоге ширина результата.

Для простоты реализации суммирование организовано следующим образом:

```
module fir ( clk, coefs, in, out );


parameter IWIDTH = 16;      //input data (signal) width

parameter CWIDTH = 16;      //tap coef data width (should be less then 32
bit)

parameter TAPS   = 2;       //number of filter taps

localparam MWIDTH = (IWIDTH+CWIDTH); //multiplied width

localparam RWIDTH = (MWIDTH+TAPS-1); //filter result width

input  wire clk;

input  wire [IWIDTH-1:0]in;

input  wire [TAPS*32-1:0]coefs; //all input coefficient concatineted

output wire [RWIDTH-1:0]out; //output takes only top bits part of result

genvar i;

generate

       for( i=0; i<TAPS; i=i+1 )

       begin:tap

              //make tap register chain

              reg [IWIDTH-1:0]r=0;

              if(i==0)

              begin

                     //1st tap takes signal from input

                     always @(posedge clk)

                            r <= in;

              end

              else

              begin
```

```
                //tap reg takes signal from prev tap reg
                always @(posedge clk)
                        tap[i].r <= tap[i-1].r;
        end

        //get tap multiplication constant coef
        wire [CWIDTH-1:0]c;
        assign c = coefs[((TAPS-1-i)*32+CWIDTH-1):(TAPS-1-i)*32];

        //calculate multiplication and fix result in register
        reg [MWIDTH-1:0]m;
        always @(posedge clk)
                m <= $signed(r) * $signed( c );

        //make combinatorial adders
        reg [MWIDTH-1+i:0]a;
        if(i==0)
        begin
                always @*
                        tap[i].a = $signed(tap[i].m);
        end
        else
        begin
                always @*
                        tap[i].a = $signed(tap[i].m)+$signed(tap[i-
1].a);
        end
    end
endgenerate
```

```
//fix calculated taps summa in register
reg [RWIDTH-1:0]result;
always @(posedge clk)
       result <= tap[TAPS-1].a;
//deliver output
assign out = result;
endmodule
```

3 Моделирование работы фильтра

Как уже было указанно ранее, для того чтобы убедиться в работоспособности реализованного фильтра необходимо использовать симулятор Verilog.

Необходимо написать testbench (метод форсирования сигналов), который будет иммитировать входной сигнал на разных частотах, чтобы посмотреть отклик на выходе фильтра. Если testbench будет синтезировать входной сигнал как синусоиду и если плавно менять ее частоту в некотором диапазоне, от низкой частоты к более высокой, то наблюдая отклик на выходе фильтра можно увидеть его амплитудно-частотную характеристику.

Структура программы testbench.v:

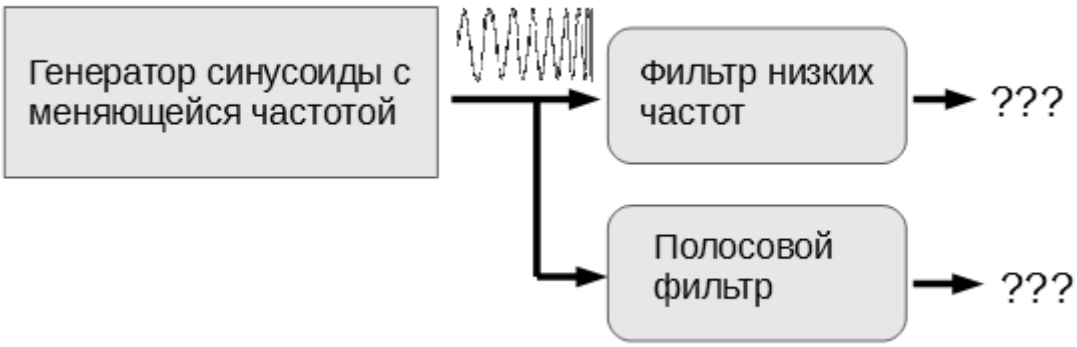

Рисунок 10 - Структура программы testbench.v

Код программы:

```
`timescale 1ns / 1ns


module testbench();


reg tb_clk;
initial tb_clk=0;
always
       #25 tb_clk = ~tb_clk;
```

```
real PI=3.14159265358979323846;
real last_time=0; //Sec
real current_time=0; //Sec
real angle=0; //Rad
real frequency=100; //Hz
integer freq_x100kHz=0; //*100kHz
reg signed [15:0]sin16;

//function which calculates Sinus(x)
function real sin;
input x;
real x;
real x1,y,y2,y3,y5,y7,sum,sign;
 begin
  sign = 1.0;
  x1 = x;
  if (x1<0)
  begin
   x1 = -x1;
   sign = -1.0;
  end
  while (x1 > PI/2.0)
  begin
   x1 = x1 - PI;
   sign = -1.0*sign;
  end
  y = x1*2/PI;
```

```
  y2 = y*y;
  y3 = y*y2;
  y5 = y3*y2;
  y7 = y5*y2;
  sum = 1.570794*y - 0.645962*y3 +
      0.079692*y5 - 0.004681712*y7;
  sin = sign*sum;
 end
endfunction

task set_freq;
input f;
real f;
begin
       frequency = f;
       freq_x100kHz = f/100000.0;
end
endtask

always @(posedge tb_clk)
begin
       current_time = $realtime;
       angle = angle+(current_time-
last_time)*2*PI*frequency/1000000000.0;
       //$display("%f %f",current_time,angle);
       while ( angle > PI*2.0 )
       begin
              angle = angle-PI*2.0;
       end
```

```
        sin16 = 32000*sin(angle);
        last_time = current_time;
end

//low-pass filter
wire [57:0]out_lowpass;
fir #( .TAPS(27) ) fir_lp_inst(
        .clk(tb_clk),
        .coefs( {
                -32'd510,
                -32'd520,
                -32'd625,
                -32'd575,
                -32'd287,
                 32'd306,
                 32'd1232,
                 32'd2467,
                 32'd3927,
                 32'd5477,
                 32'd6948,
                 32'd8162,
                 32'd8962,
                 32'd9241,
                 32'd8962,
                 32'd8162,
                 32'd6948,
                 32'd5477,
                 32'd3927,
```

```
                32'd2467,
                32'd1232,
                32'd306,
               -32'd287,
               -32'd575,
               -32'd625,
               -32'd520,
               -32'd510
               } ),
        .in(sin16),
        .out(out_lowpass)
        );

//band-pass
wire [55:0]out_bandpass;
fir #( .TAPS(25) ) fir_bp_inst(
        .clk(tb_clk),
        .coefs( {
               -32'd801,
               -32'd1026,
               -32'd210,
                32'd1914,
                32'd4029,
                32'd3905,
                32'd330,
               -32'd5174,
               -32'd8760,
               -32'd7040,
```

```
                -32'd152,
                 32'd7700,
                 32'd11130,
                 32'd7700,
                -32'd152,
                -32'd7040,
                -32'd8760,
                -32'd5174,
                 32'd330,
                 32'd3905,
                 32'd4029,
                 32'd1914,
                -32'd210,
                -32'd1026,
                -32'd801
                } ),
        .in(sin16),
        .out(out_bandpass)
        );

integer i;
real f;

initial
begin
        $dumpfile("out.vcd");
        $dumpvars(0,testbench);
        f=100000;
```

```
        for(i=0; i<4000; i=i+1)
        begin
                set_freq(f);
                #1000;
                f=f+1000;
        end
        $finish;
end


endmodule
```

Симуляцию проводим в icarus-verilog.

Компилируем:

> iverilog -o qqq testbench.v fir.v

Симулируем:

> vvp qqq

Просмотр временных диаграмм в GtkWave:

> gtkwave out.vcd

В результате симуляции программой **vvp** получается выходной файл с временными диаграммами сигналов **out.vcd**. Их можно посмотреть в программе GtkWave.

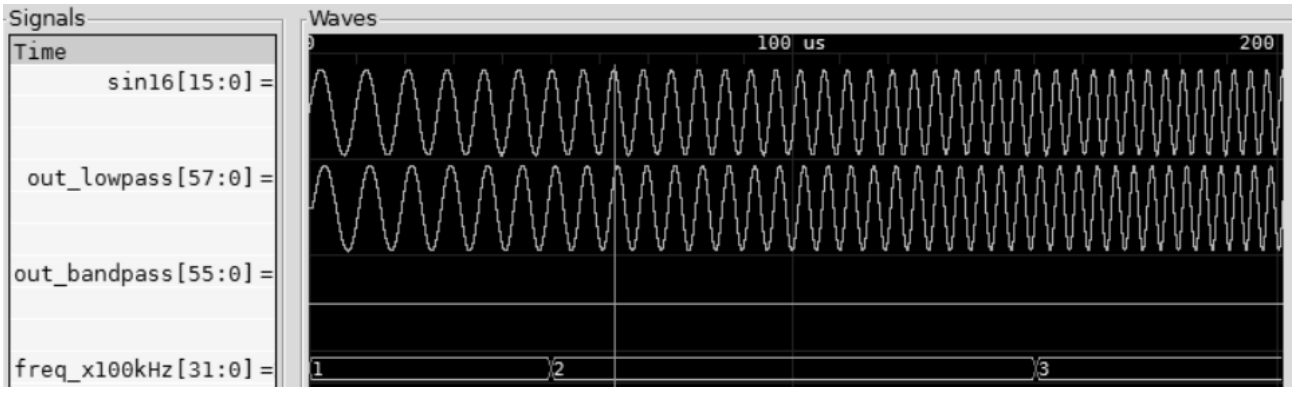

Рисунок 11 – Временные диаграммы

что видно: входной сигнал sin16 для обоих фильтров меняет частоту,

Чтобы внимательно рассмотреть получившуюся картину нужно использовать специальные возможности просмотра в программе GtkWave:

- для знаковых сигналов, как sin16 или out_lowpass, out_bandpass кликнуть правой кнопкой мыши по сигналу и выбрать сперва Data Format => Sogned Decimal, а потом Data Format => Analog => Step;
- чтобы отобразить аналоговый сигнал на графике в большую высоту использовать правый клик мыши на сигнале и потом пункт popup меню Insert Analog Height Extension.

Заключение

Анализируя полученные при моделировании графики можно сделать следующие выводы:

- входной сигнал sin16 для обоих фильтров меняет частоту, она растет;
- на широком диапазоне частот от сотен килогерц до нескольких мегагерц видно какие частоты пропускаются фильтрами, а какие подавляются;
- входной сигнал для sin16 равномерен по амплитуде во всей полосе частот.
- а вот выходные сигналы, такие какие и положены быть на выходе фильтров. Фильтр низких частот от 500кГц начинается плавный спад амлитуды синусоиды out_lowpass и после 2х МГц фильтр уже практически не пропускает. Полосовой фильтр хорошо пропускает где-то в полосе 2-3МГц, а по краям пологие спады (сигнал out_bandpass).

Таким образом, видно, что разрабатываемый фильтр выполняет свои функции и можно сделать вывод о реализации поставленной задачи в полном объеме.

Список использованных источников

1. Основы цифровой обработки сигналов. Курс лекций / А. И. Солонина, Д. А. Улахович, С. М. Арбузов, Е. Б. Соловьёва – Изд. 2-е испр. и перераб. – СПб.: БХВ-Петербург, 2005, – 768 стр.

2. Рабинер Л. Теория и применение цифровой обработки сигналов перевод с английского под ред. Ю. Н. Александрова / Л. Рабинер, Б. Гоулд – М.: Мир, 1978, – 848 стр.

3. Википедия - свободная энциклопедия. ПЛИС -[Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/%D0%9F%D0%9B%D0%98%D0%A1 Дата обращения: 02.12.2018

4. Википедия - свободная энциклопедия. Icarus Verilog- [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Icarus_Verilog Дата обращения: 03.12.2018

5. Соловьев В. В. Логическое проектирование цифровых систем на основе программируемых логических интегральных схем / В. В. Соловьев, А. Климович – М.: Горячая линия – Телеком, 2008. – 376 с.

6. Черемисин А. Г. Оценка эффективности применения ПЛИС и процессоров DSP для задач цифровой обработки сигналов, – Научная электронная библиотека «КиберЛенинка» – https://cyberleninka.ru/article/v/otsenkaeffektivnosti-primeneniya-plis-i-protsessorov-dsp-dlya-zadach-tsifrovoyobrabotki-signalov

7. Документация Altera FIR II URL: https://www.altera.com/documentation/hco1421694595728.html Дата обращения: 16.12.2018