МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ имени В.Ф. Уткина

кафедра информационно-измерительной и биомедицинской техники

Курсовая работа

на тему

«Цифровой модуль робота на основе ПЛИС»

Выполнил: студент

Группы 933

Соловьев Д.А.

Проверил:

Голь С.А.

Рязань 2022

# Содержание

[Содержание 2](#_Toc105704610)

[Используемые сокращения 3](#_Toc105704611)

[Список обозначений 4](#_Toc105704612)

[Введение 5](#_Toc105704613)

[Элементы теории 6](#_Toc105704614)

[Моделирование фильтра в MATLAB 10](#_Toc105704615)

[Структура программы 23](#_Toc105704616)

[Фильтр на 9 умножителях 24](#_Toc105704617)

[Фильтр на 3 умножителях 32](#_Toc105704618)

[Заключение 39](#_Toc105704619)

[Список литературы 40](#_Toc105704620)

[Приложение 41](#_Toc105704621)

# Используемые сокращения

АЦП – аналого-цифровой преобразователь,

ЦАП – цифро-аналоговый преобразователь,

ПЛИС – программируемая логическая интегральная схема,

БИХ – бесконечная импульсная характеристика.

КИХ – конечная импульсная характеристика.

кГц, Гц – килогерц, герц.

В – вольт.

# Список обозначений

 – время.

fn – n-ое входное значение,

bk – k-ый коэффициент b фильтра,

ak – k-ый коэффициент a фильтра,

N – количество коэффициентов a или b,

yn - n-ое выходное значение.

# Введение

Цифровые фильтры на сегодняшний день применяются практически везде, где требуется обработка сигналов, в частности в спектральном анализе, обработке изображений, обработке видео, обработке звука и речи и многих других приложениях.

В данном курсовом проекте требуется разработать модуль односекционного рекурсивного полосового фильтра по канонической схеме.

В проектирование данного устройства будут применяться средства разработки, такие как, пакет программ Matlab и Simulink, а также интегрированная среда разработки цифровых устройств на базе программируемых логических интегральных схем фирмы Altera - Quaruts II version 13.0.1.

# Элементы теории

Разберём преимущества и недостатки цифровых фильтров по сравнению с аналоговыми.

К преимуществам цифровых фильтров по сравнению с аналоговыми можно отнести:

* Высокая точность (точность аналоговых фильтров ограничена допусками на элементы)
* Стабильность (в отличие от аналогового фильтра передаточная функция не зависит от дрейфа элементов).
* Гибкость настройки, лёгкость изменения.
* Компактность. Например, аналоговый фильтр на очень низкую частоту (доли герца) потребовал бы чрезвычайно громоздких элементов: конденсаторов и индуктивностей.

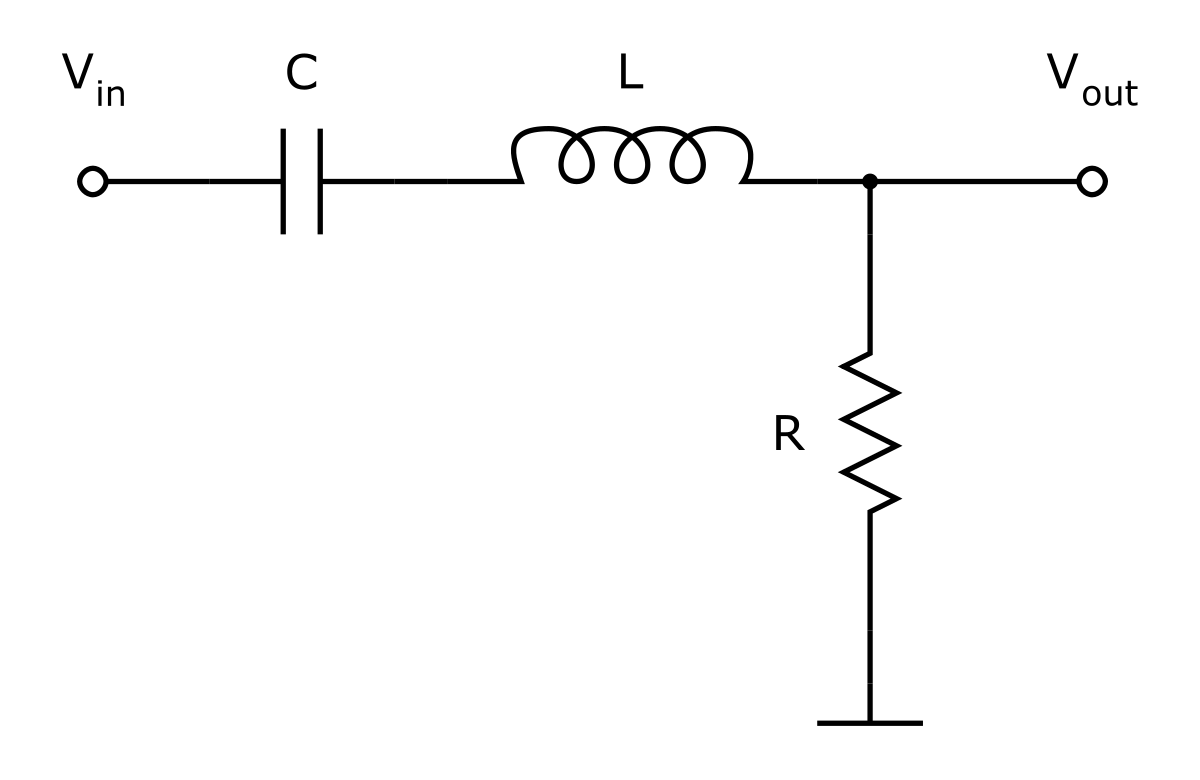
К недостаткам цифровых фильтров по сравнению с аналоговыми можно отнести:

* Трудность работы с высокочастотными сигналами. Полоса частот ограничена частотой Найквиста, равной половине частоты дискретизации сигнала. Поэтому для высокочастотных сигналов применяют аналоговые фильтры, либо, если на высоких частотах нет полезного сигнала, сначала подавляют высокочастотные составляющие с помощью аналогового фильтра, затем обрабатывают сигнал цифровым фильтром.
* Трудность работы в реальном времени – вычисления должны быть завершены в течение периода дискретизации.
* Для большой точности и высокой скорости обработки сигналов требуется не только мощный процессор, но и дополнительное, возможно дорогостоящее, аппаратное обеспечение в виде высокоточных и быстрых ЦАП и АЦП.

**Способы реализации цифровых фильтров:**

Различают два вида реализации цифрового фильтра: аппаратный и программный. Аппаратные цифровые фильтры реализуются на элементах интегральных схем, тогда как программные реализуются с помощью программ, выполняемых ПЛИС, процессором или микроконтроллером. Преимуществом программных перед аппаратным является лёгкость воплощения, а также настройки и изменений, а также то, что в себестоимость такого фильтра входит только труд программиста. Недостаток — низкая скорость, зависящая от быстродействия процессора, а также трудная реализуемость цифровых фильтров высокого порядка.

Полосовой фильтр – это фильтр, пропускающий составляющие, находящиеся в некоторой полосе частот. Поэтому составляющие сигнала с частотами выше и ниже полосы пропускания, будут ослабляться, в то время, или отфильтровываться, в то время как составляющие с частотами, находящимися в полосе пропускания, проходят с умеренным затуханием.

Простейший полосовой фильтр состоит из последовательно соединённых фильтров верхних (LC фильтр) и нижних (RC фильтр) частот (Рисунок 1).

Рисунок

Рисунок 1. Простейший полосовой фильтр.

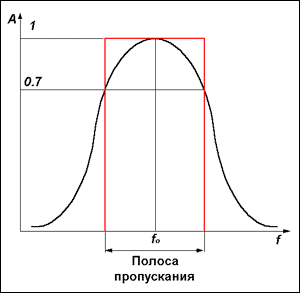
АЧХ такого фильтра имеет вид (Рисунок 2).

Рисунок . АЧХ полосового фильтра.

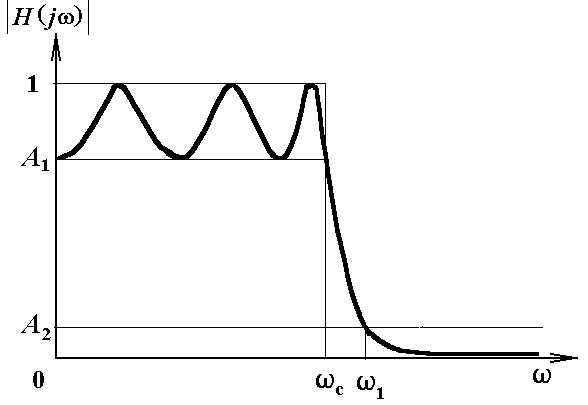
Фильтр Чебышева 1 рода – один из типов линейных аналоговых или цифровых фильтров, отличительной особенностью которого является более крутой спад АЧХ и существенные пульсации на частотах полос пропускания (Рисунок 3).

Рисунок . АЧХ фильтра Чебышёва.

Фильтр с бесконечной импульсной характеристикой (рекурсивный фильтр, БИХ-фильтр) — линейный электронный фильтр, использующий один или более своих выходов в качестве входа, то есть образует обратную связь. Основным свойством таких фильтров является то, что их импульсная переходная характеристика имеет бесконечную длину во временной области, а передаточная функция имеет дробно-рациональный вид. Такие фильтры могут быть как аналоговыми, так и цифровыми.

Существует несколько схем реализации БИХ фильтров. Между собой они отличаются количеством использованных элементов задержки.

Так на рисунке 4 мы видим что в схеме Direct Form I используется в 2 раза больше элементов задержки чем в схеме Direct Form II.

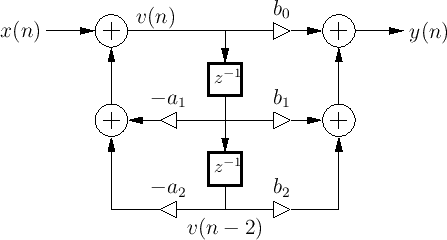
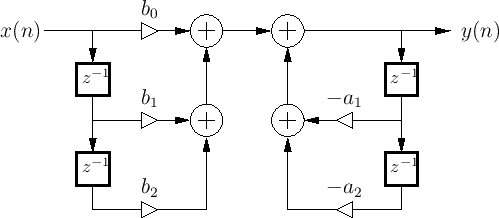


Рисунок . Схемы реализации БИХ фильтров. Сверху - Direct Form I, снизу - Direct Form II.

# Моделирование фильтра в MATLAB

Исходные данные для проектирования:

* Band Pass. (Полосовой фильтр).
* Direct Form Ⅱ Transposed. (Прямая форма 2, транспонированная)
* Chebyshev Type Ⅰ, IIR (Фильтр Чебышева 1, БИХ)
* Specify order = 4 (Порядок фильтра)
* Section 1 (Односекционный фильтр)
* Fs = 16kHz (Частота дискретизации, кГц)
* Fc1 = 400 Hz (Нижняя частота среза, Гц)
* Fc2 = 700 Hz (Верхняя частота среза, Гц)
* Максимальная динамическая погрешность = 2,5 %

Для проектировки цифрового фильтра воспользуемся внутренней библиотекой MATLAB fdatool, которая позволяет смоделировать параметры проектируемого фильтра, путём подстановки исходных данных.

Получаем АЧХ и ФЧХ проектируемого фильтра (Рисунок 5).

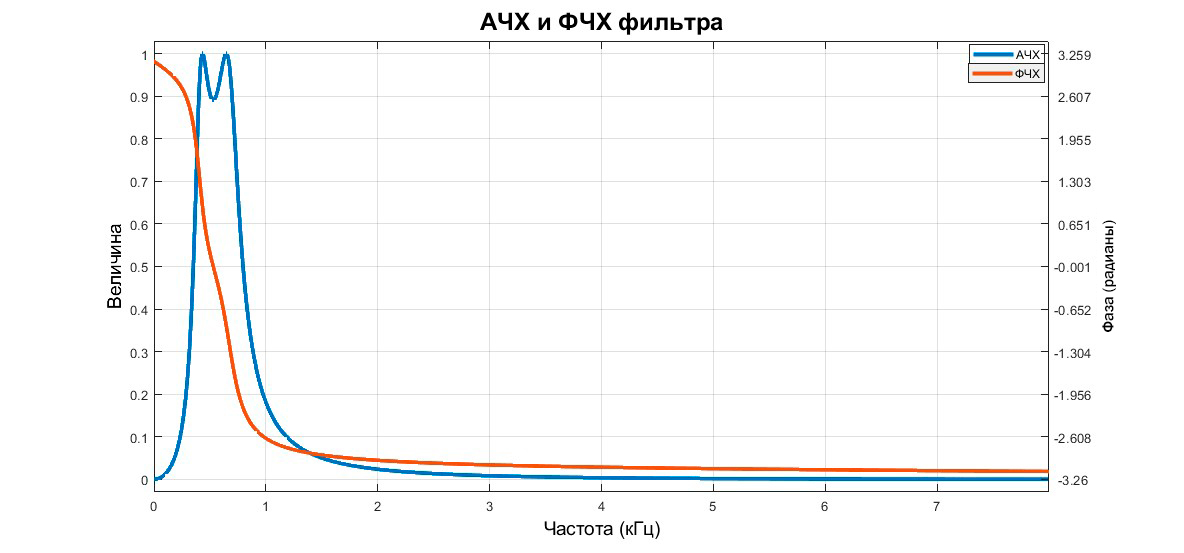


Рисунок . АЧХ и ФЧХ проектируемого фильтра.

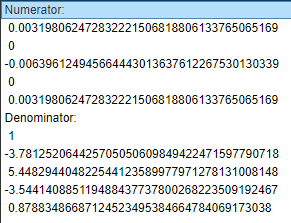
Получаем коэффициенты фильтра для разработки проекта (Рисунок 6).

Рисунок . Коэффициенты фильтра.

Чтобы продемонстрировать работоспособность фильтра, в Matlab и Simulink была создана его структурная схема и модель для проверки (Рисунок 7 и Рисунок 8).

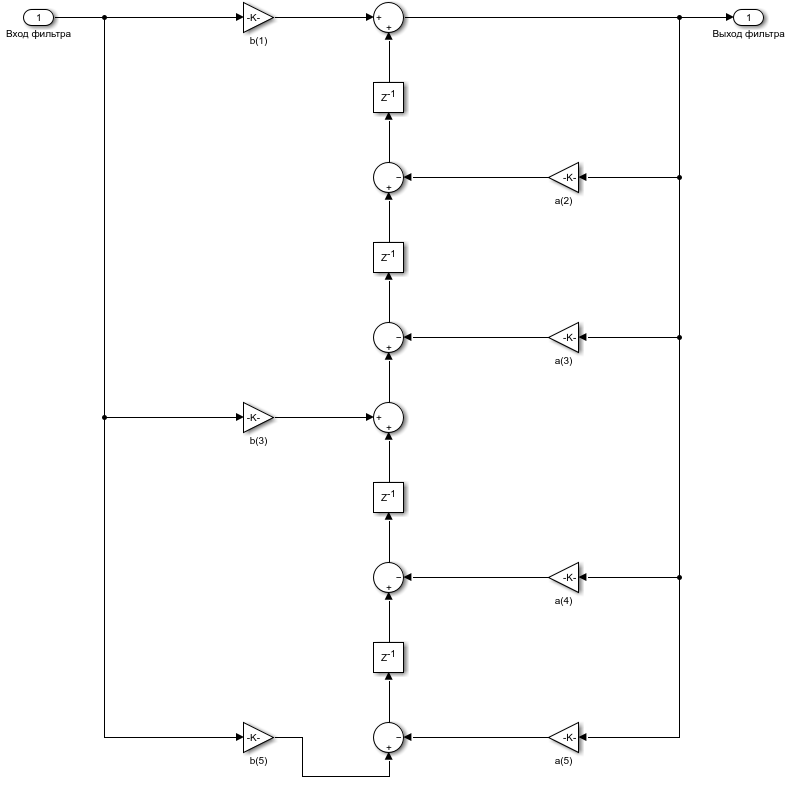


Рисунок . Структурная схема фильтра.

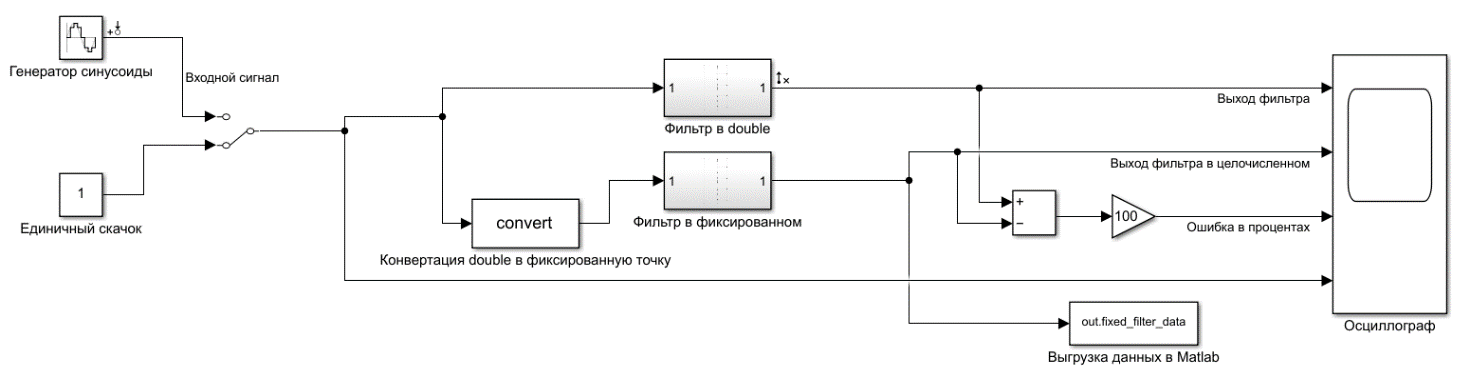


Рисунок . Модель для проверки фильтра.

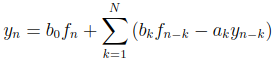
ПЛИС не способна работать с вещественными числами. Для корректной работы переведем все вычисления в целочисленный формат. Это действие также имеет ряд плюсов:

* целые числа всегда представимы своими точными значениями
* операции целочисленной арифметики дают точные результаты
* операции целочисленной арифметики выполняются быстрее, чем операции вещественной («плавающей») арифметики.

Перевод в общем случае заключается в следующих действиях:

1. Определяем наибольшее целое число, которое будет участвовать при вычислениях. Определяем разрядность новой системы исчисления, в которую помещается наибольшее число, участвующее в вычислениях (5 в 3 бита двоичной системы счисления, 24 в 5 бит двоичной системы).
2. Умножаем исходное число на основание новой системы счисления в степени n. Чем больше n, тем больше точность числа и последующих вычислений.
3. Результатом всех вычислений будет число разрядностью N = n + 1 + z, где n количество бит под дробную часть, z количество бит под целую часть числа и 1 под знак.

В ПЛИС не заложена возможность работать с отрицательными числами, поэтому все операции будут проводиться в дополнительном коде. Для получения отрицательных целочисленных значений в дополнительном коде необходимо воспользоваться формулой

Работу цифрового фильтра можно описать следующим выражением:

Где: fn – n-ое входное значение, b0 – 1-ый коэффициент b фильтра, bk – k-ый коэффициент b фильтра, ak – k-ый коэффициент a фильтра, N – количество коэффициентов a или b, yn - n-ое выходное значение.

Так как в реализации фильтра на ПЛИС используется дополнительный код и имеется фиксированный разряд, отвечающий за знак числа, то для упрощения работы есть смысл написать логику фильтра, используя только сумматоры. Для этого перемножим коэффициенты a на -1. Старший бит станет 1, так как число отрицательное. При сложении может происходить переполнение регистра, но так как разрядность фиксирована, то полученный лишний бит не учитывается в ответе. Для примера, иллюстрирующего принцип вычислений в дополнительном коде, возьмем два четырехбитных числа, старший бит будет отвечать за знак.

Двоичный обратный код Десятичный код

0111+ 1011 = 0010 7 – 4 = 3

Подберем оптимальную разрядную сетку для фильтра. Разрядность фильтра зависит от нескольких факторов: от значений коэффициентов, от значения амплитуды входного сигнала, от значения ошибки между образцовым фильтром и фильтром на ПЛИС.

Подавать будем сигнал амплитудой 3.3 вольт, без смещения и с частотой внутри полосы пропускания фильтра с условием того, что на этой частоте фильтр пропускает сигнал практически без искажений. В моем случае это 650 гЦ.

Начнем с 16 битного числа: 3 бита числа + 1 бит под знак = 4, это количество бит под целую часть, следовательно 16 – 4 = 12 бит под дробную. (Рисунок 9)

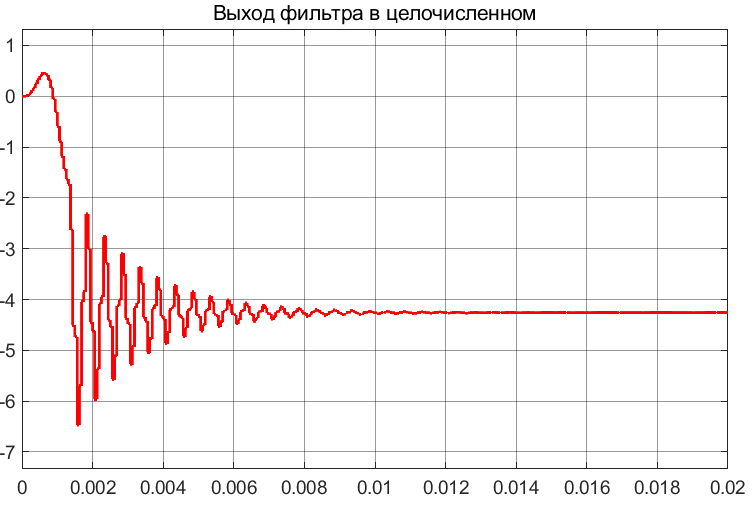


Рисунок . Выход фильтра 16 битной разрядности с переполнением.

И видим переполнение на графике, о чем также предупреждает Simulink (Рисунок 10). Переполнение произошло при умножении на коэффициент a3 = 5.44829440482254412. Он самый большой из коэффициентов. Повысим количество бит под целую часть на 1.



Рисунок . Предупреждение Simulink о переполнении.

В результате переполнения нет и Simulink не выдает ошибку, но значения в некоторых местах графика превышают 200%. (Рисунок 11). Это значение - динамическая погрешность, или же ошибка.

Ошибка берется из-за округления при переходе от вещественного представления числа к целочисленному. В качестве функции округления Matlab предоставляет такие варианты как: fix() - отброс дробной части числа, floor-округление в меньшую сторону, ceil()- округление в большую сторону, round()- округление до ближайшего целого. Для имитации работы ПЛИС будем применять функцию округления умножителя Ceiling (аналог ceil()).

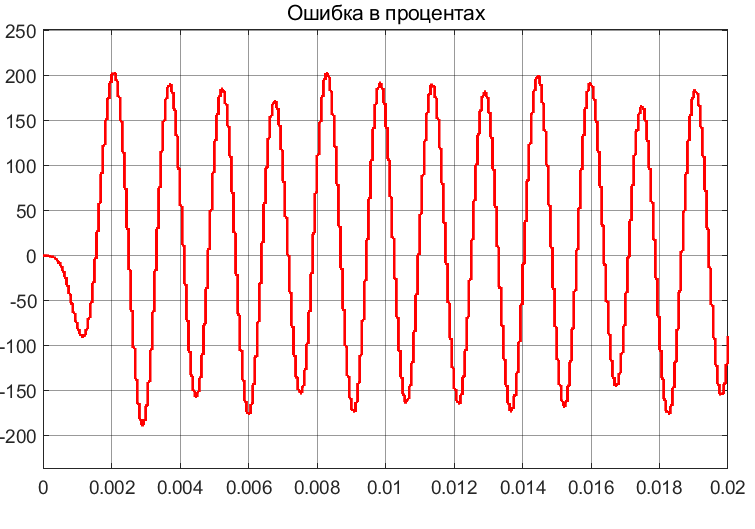


Рисунок . Ошибка фильтра 16 битной разрядности без переполнения.

Будем повышать разрядность и количество бит под целую часть, до тех пор, пока не исчезнет переполнение и динамическая погрешность не станет меньше заданной. В результате получилось 24 разрядное число с 18 бит под дробную часть. В результате, на отрезке времени симуляции T = 0.02 секунд максимальная ошибка составила 1.653%, это меньше 2.5%, следовательно разрядность выбрана верно. (Рисунок 12). Попробуем уменьшить разрядность до 23 бит и 17 бит под дробную часть, и понаблюдаем за ошибкой, она увеличилась до 2.81%, что не вписывается в заданный диапазон. (Рисунок 13)

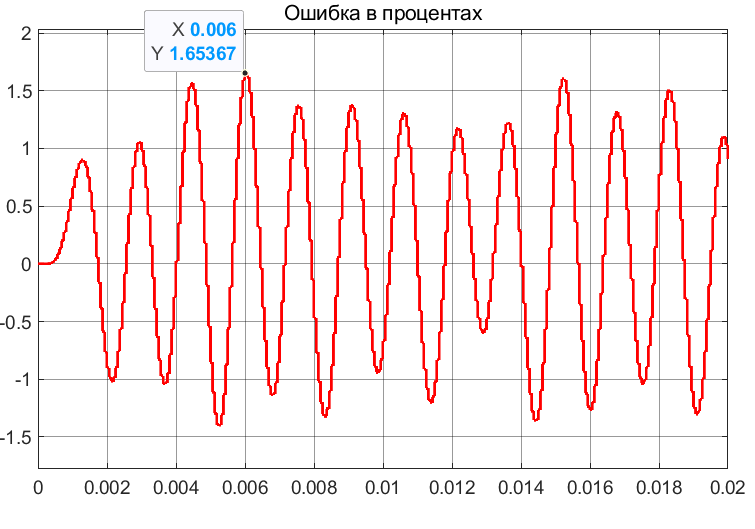


Рисунок . Выход фильтра 24 битной разрядности без переполнения.

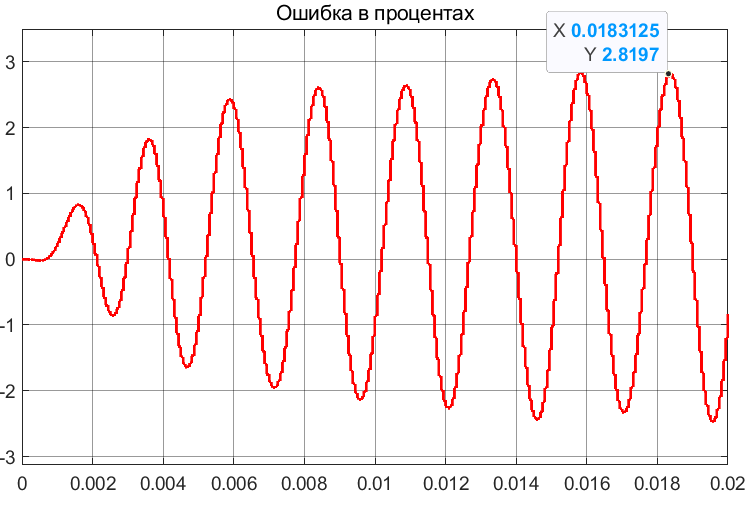


Рисунок . Выход фильтра 23 битной разрядности без переполнения.

Необходимые для дальнейшей работы ПЛИС 24 разрядные коды коэффициентов a и b моей модели представлены в виде таблицы coefficient\_table в рабочем пространстве Matlab (Рисунок 14). Нужно отметить, что в ПЛИС поступают коэффициенты -a, так как используются только сумматоры.

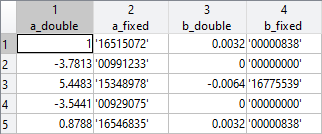


Рисунок . Таблица данных coefficient\_table. Обозначения колонок слева направо: 1 – коэффициенты a в формате double; 2 - коэффициенты -a в целочисленном формате в дополнительном коде; 3 - коэффициенты b в формате double; 4 - коэффициенты b в целочисленном формате в дополнительном коде.

Убедимся в том, что фильтр работает.

Суть модели для проверки фильтра состоит в следующем (Рисунок 8). Имеется некий аналоговый сигнал (например, синусоида с частотой 650 Гц, амплитудой 3.3 вольт и без смещения). Изначально сигнал представлен в вещественном виде (тип с плавающей точкой по стандарту IEEE 754) в формате повышенной точности double.

Данный сигнал проходит параллельно через 2 фильтра - образцовый фильтр и фильтр на ПЛИС. Пройдя через образцовый фильтр, сигнал попадает на осциллограф.

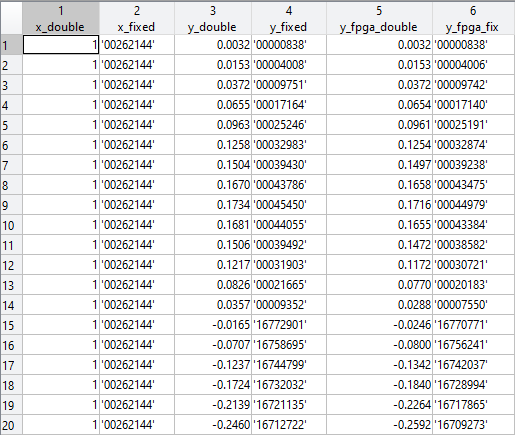
Сигнал также попадает на вход преобразователя из double в фиксированный дополнительный код, с указанной разрядностью (24 бита, 18 бит под дробную часть). С выхода преобразователя коды сигнала поступают на вход фильтра на ПЛИС. С выхода фильтра сигнал попадает на осциллограф и в рабочую область Matlab в виде столбца таблицы данных data\_table (Рисунок 15).

Рисунок . Таблицы данных data\_table. Обозначения колонок слева направо: 1 – сигнал, подаваемый на вход фильтра (Вольт); 2 – сигнал, в фиксированном целочисленном дополнительном коде; 3 – сигнал с выхода образцового фильтра в формате double (Вольт); 4 – сигнал с выхода образцового фильтра выхода в фиксированном целочисленном дополнительном коде; 5 – сигнал с выхода модели фильтра на ПЛИС в формате double (Вольт); 6 - сигнал модели фильтра на ПЛИС в фиксированном целочисленном дополнительном коде.

Также на осциллограф попадают: сигнал ошибки (динамическая погрешность) и входной сигнал. По сигналу ошибки определяется корректность выбранной разрядной сетки.

Протестируем фильтр на разных частотах. Согласно заданию, частоты среза для данного фильтра будут:

* Нижняя частота среза 400 Гц
* Верхняя частота среза 700 Гц

На частотах среза амплитуда сигнала должна составлять 0.9 от исходной величины. Соответственно сигнал в полосе пропускания фильтра должен быть приближен к исходному, а за пределами полосы пропускания сигнал должен существенно ослабевать.

Чтобы проверить работоспособность фильтра будем подавать синусоидальный сигнал c амплитудой 3.3 В и без смещения на различных частотах: 300 Гц, 400 Гц, 500 Гц, 700 Гц, 1000 Гц (Рисунок 16, Рисунок 17, Рисунок 18, Рисунок 19, Рисунок 20).

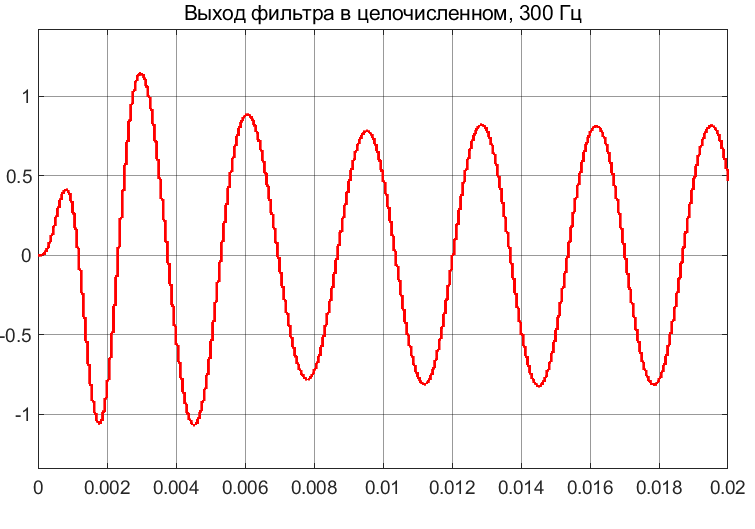


Рисунок . Сигнал с частотой 300 Гц



Рисунок . Сигнал с частотой 400 Гц.

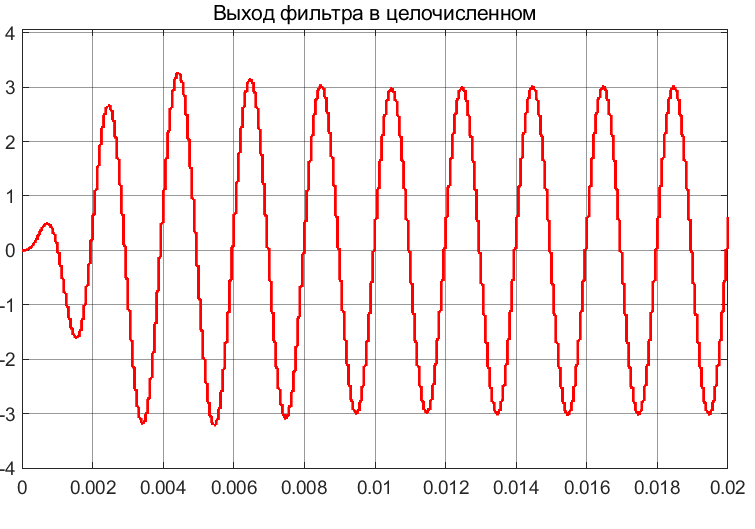


Рисунок . Сигнал с частотой 500 Гц.

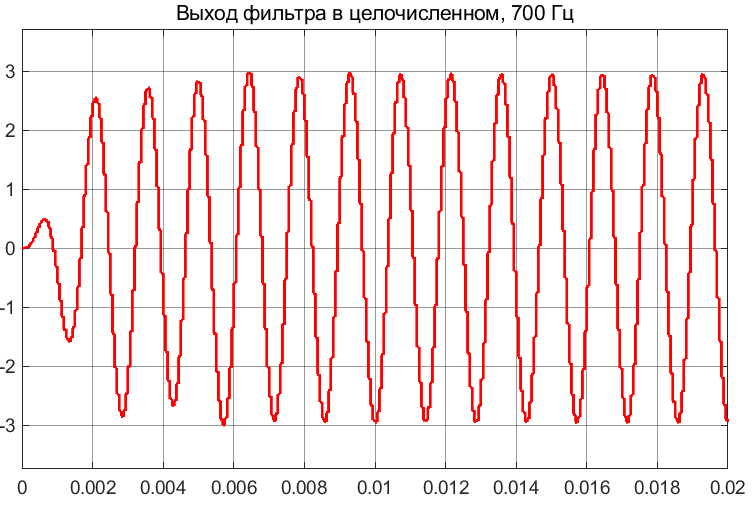


Рисунок . Сигнал с частотой 700 Гц.

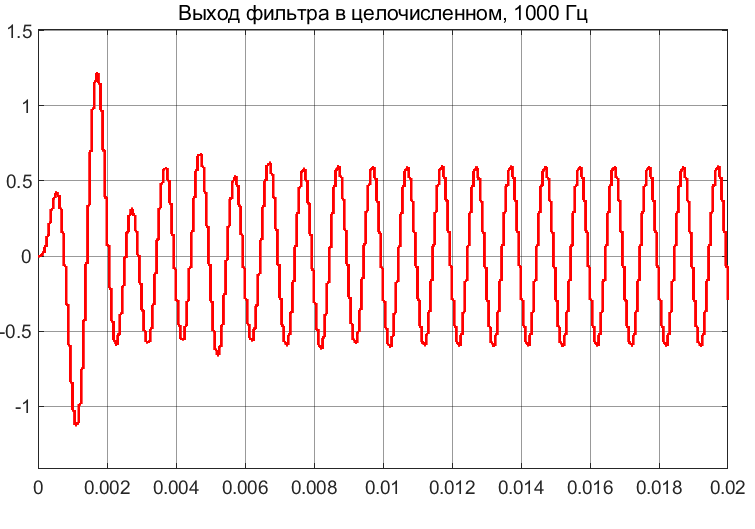


Рисунок . Сигнал с частотой 1000 Гц.

Из полученных осциллограмм видно, что за пределами полосы пропускания сигнал существенно ослабевает. Сделаем вывод, что коэффициенты фильтра и разрядная сетка подобраны верные. Получение на частотах среза амплитуды большей чем 0.9 от исходной связанно с тем, что в фильтре Чебышева первого рода, на частотах пропускания существуют пульсации.

# Структура программы

Для того чтобы проверить работоспособность разработанного фильтра на отладочной плате, напишем программный код на языке AHDL. В качестве среды разработки будем использовать Quaruts II version 13.0.1. По заданию необходимо сделать 2 программы, на 9 умножителях (приложение 1), и на 3 умножителях (приложение 2).

# Фильтр на 9 умножителях

В качестве сумматора будем использовать функцию Altera lpm\_add\_sub, в качестве умножителя Altera lpm\_mult, в качестве задержки - регистр my\_reg. Счетчик Altera lpm\_counter.

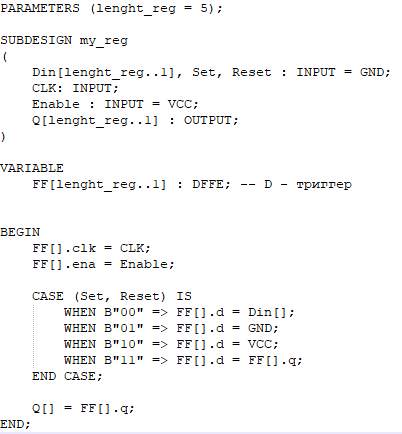
 Регистр будет иметь две шины: входные и выходные значения размерностью «length\_reg», также имеются 4 входа: CLK - тактирование триггеров, Enable - разрешение на запись, SET, RESET, отвечающие за сброс в 0 и установку в 1; в дальнейшем входы SET и RESET не используются. (Рисунок 21)

Рисунок . Описание регистра my\_reg

Перейдем к основной программе course\_9\_mult.tdf

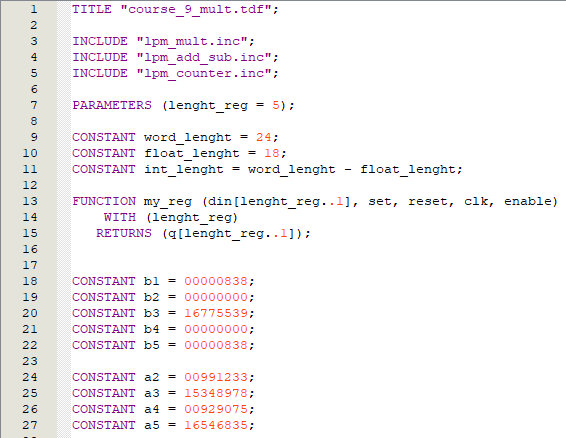


Рисунок . Предварительная настройка

Указываем подключаемые функции, название проекта, разрядности чисел, описываем прототип функции my\_reg, задаем постоянные коэффициенты. Коэффициенты берем из таблицы coefficient\_table в Matlab (Рисунок 22).

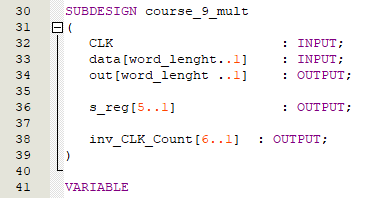


Рисунок . Раздел описания интерфейса модуля: задаем используемые входные и выходные сигналы.

В разделе описания интерфейса модуля Subdesign Section задаем имя модуля и перечисляем его выводы. Имя модуля должно совпадать с именем логического файла, в котором хранится его текстовое описание. Clk[] – тактовый сигнал, data[] – входные данные, out[] – отфильтрованные данные, s\_reg[] – сигнал на перезапись регистров, inv\_CLK\_Cound[] – задает моменты времени для перезаписи регистров. (Рисунок 23)

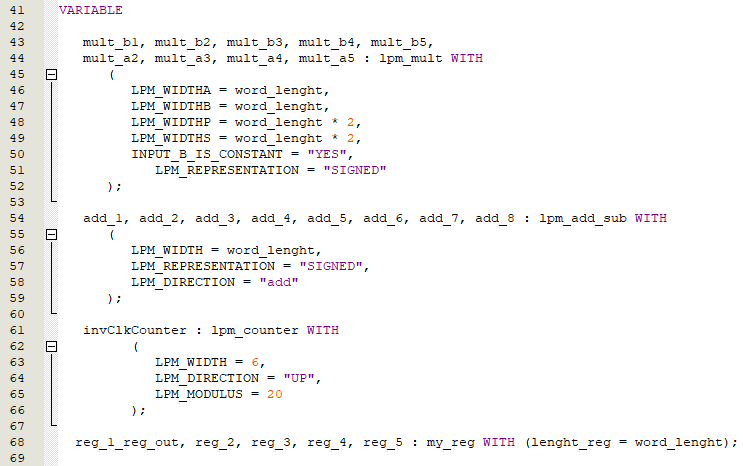
В разделе переменных указываем используемые регистры, сумматоры, счетчики и умножители. Производим их настройку. Количество задержек в программе отличается от количества задержек на схеме, из-за того, что при перезаписи задержек значение на первом сумматоре меняется и выходное значение становится неверным. Для правильной работы программы добавлена пятая задержка (регистр reg\_1\_reg\_out). (Рисунок 24)

Рисунок . Раздел переменных.

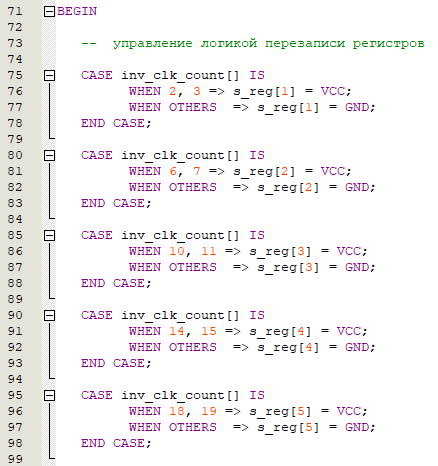
В начале раздела описания логики, произвожу настройку перезаписи регистров при помощи сигнала с счетчика инвертированных фронтов clk invClkCounter. Инвертированные фронты тактового сигнала CLK необходимы для синхронного переключения регистров. (Рисунок 25).

Рисунок . Начало раздела описания логики. Описание моментов времени перезаписи регистров.

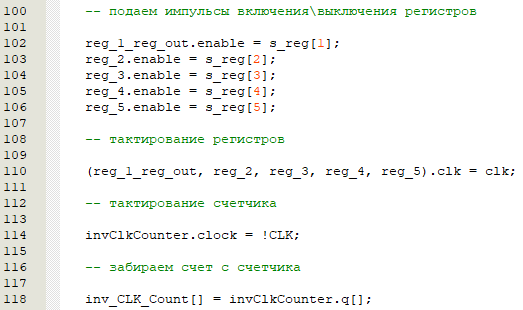


Рисунок . Задание необходимых сигналов.

Произвожу тактирование регистров и счетчиков, забираю данные с счетчика. Раздаю сигналы перезаписи регистрам. (Рисунок 26)

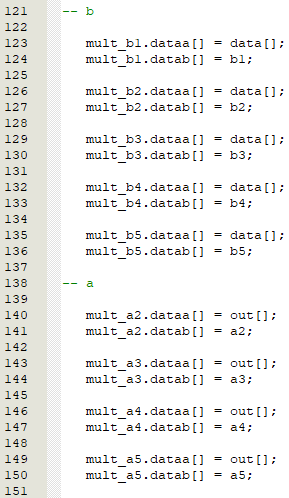


Рисунок . Описание работы 9 умножителей.

Описываю работу умножителей, подключаю их входы. (Рисунок 27)

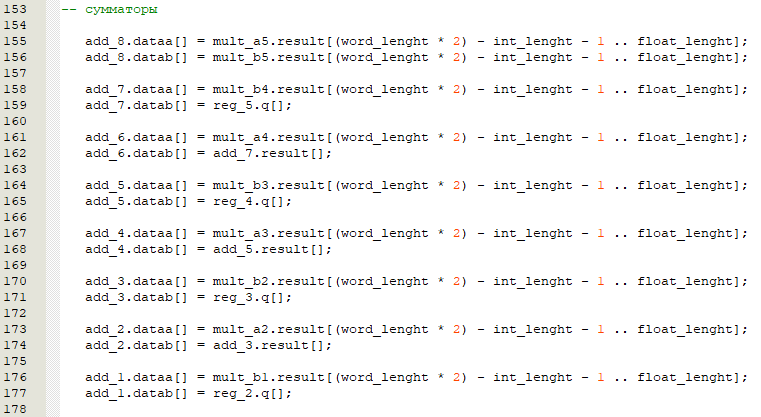


Рисунок . Описание работы сумматоров.

Суммируем результаты умножений. Входы всех элементов имеют разрядность фильтра, а результат умножителей в 2 раза ее превышает, получается число с двойной точностью. Это сделано специально, для предотвращения переполнения. Для перехода в формат с прежней разрядностью требуется отбросить float\_lenght (количество бит дробной части) младших битов, таким образом отказываемся от 2 точности в дробной части числа, и также отбрасываем int\_lenght + 1 (количество бит целой части) старших битов, ликвидируя двойную точность в целой части, не забывая про знак. (Рисунок 28).

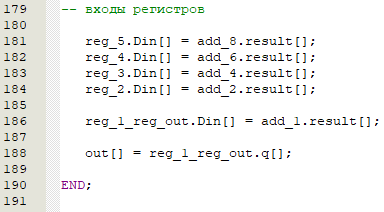


Рисунок . Записываем значения в регистры. Конец программы.

В заключительной части программы описываем входы регистров. Выходной сигнал, или же отфильтрованный сигнал – выход регистра reg\_1\_reg\_out. (Рисунок 29)

Запустим Waveform editor и файл course\_9\_mult.vwf для симуляции процесса работы описанного устройства. Укажем все используемые нами порты ввода/вывода, зададим частоту тактового сигнала clk, такую, чтобы частота сигнала out = 16 кГц (Период 62.5 микросекунды). Период clk = 3.14 микросекунд. К сожалению, Waveform editor программы Quartus II выдает ошибку, когда я пытаюсь выставить время симуляции больше 100 микросекунд. Поэтому симуляцию результата будем проводить в программе фирмы Altera Max+plus II. Выделенная рамкой секция с названием Interval показывает период дискретизации фильтра, и он составляет 62.7 микросекунды или же приблизительно 16 кГц.

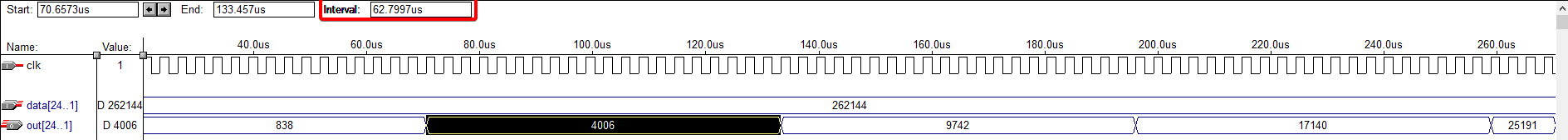


Рисунок . Симуляция работы фильтра в Max+plus II.

Сверим значения с моделью в Matlab, для этого выберем таблицу data\_table, и посмотрим на пятый столбец. Первые 20 значений сходятся точь-в-точь, что свидетельствует о правильной работе фильтра на девяти умножителях. (Рисунок 30 и Рисунок 15).

Также проверим работу фильтра при отрицательном значении входа. Подадим скачок с амплитудой -1 вольт (Рисунок 31) и сравним реакцию фильтра на AHDL с таблицей значений data\_table. Результаты первых 20 значений идентичны. (Рисунок 32)

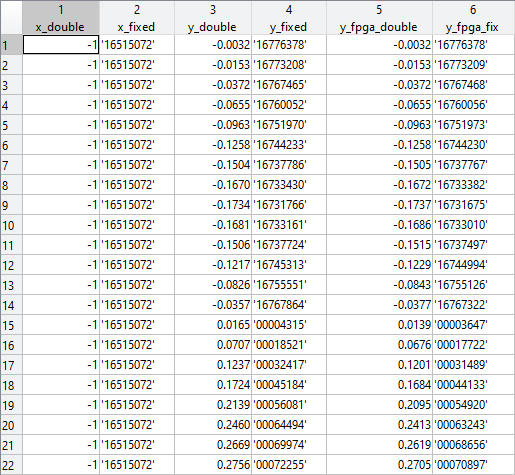


Рисунок . Таблица data\_table с отрицательными значениями данных входа. Амплитуда скачка -1 вольт.

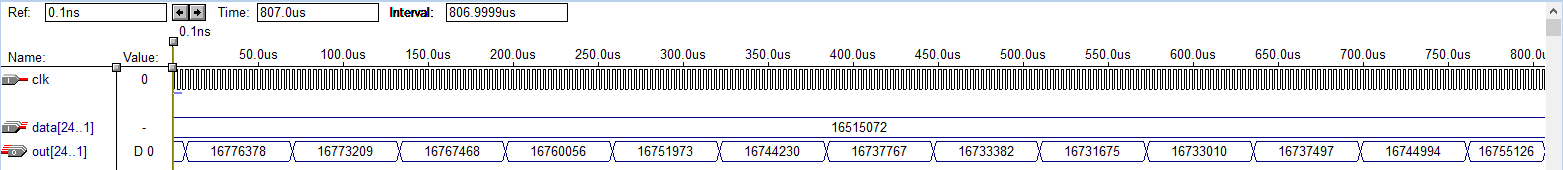


Рисунок . Симуляция работы фильтра в Max+plus II с отрицательными значениями на входе.

# Фильтр на 3 умножителях

Структурная схема фильтра на трех умножителях представлена на рисунке 33.

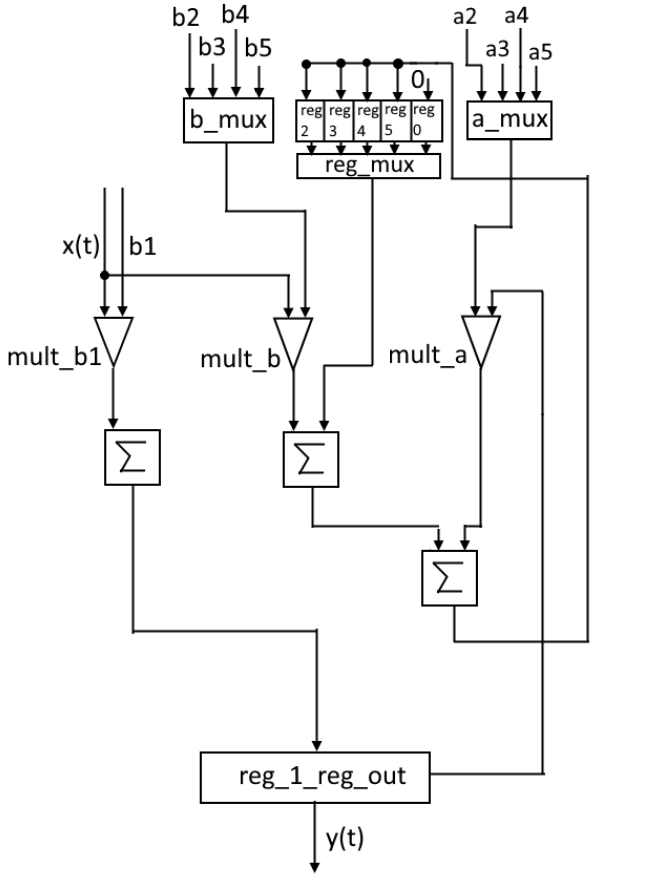


Рисунок . Структурная схема фильтра на трех умножителях.

Данная схема имеет ряд отличий от фильтра на девяти умножителях:

1. Добавились 3 мультиплексора b\_mux, a\_mux, reg\_mux. Они отвечают за своевременную смену умножаемых коэффициентов и регистров.
2. Добавился новый регистр reg\_0, который хранит в себе значение нуля, на скорость работы схемы не влияет, так как регистр не перезаписывается.
3. Уменьшилось количество умножителей и сумматоров, следовательно ресурсы ПЛИС освободились.

Идея схемы заключается в следующем: в отличии от схемы на 9 умножителях, схема на 3 умножителях работает по 4 условным шагам, которые повторяются друг за другом на всем протяжении работы фильтра. (Шаг 1, шаг 2, шаг 3, шаг 4, шаг 1, шаг 2, и т.д.) Каждый шаг состоит из нескольких простых операций, которые идут параллельно. (Выбор – подача необходимого адресного сигнала на мультиплексор, перезапись – включение и выключение записи выбранного регистра). Опишем эти шаги и операции:

Шаг 1. Выбор регистра reg\_2; Перезапись регистра reg\_1\_reg\_out; Выбор reg\_3, b2, a2; Перезапись регистра reg\_2.

Шаг 2. Выбор регистра reg\_4, переключение мультиплексоров в положения, где записаны коэффициенты b3 и a3; Перезапись регистра reg\_3.

Шаг 3. Выбор регистра reg\_5, переключение мультиплексоров в положения, где записаны коэффициенты b4 и a4; Перезапись регистра reg\_4.

Шаг 4. Выбор регистра reg\_0, переключение мультиплексоров в положения, где записаны коэффициенты b5 и a5; Перезапись регистра reg\_5.

Результатом этих 4 шагов является отфильтрованное значение на выходе out. Далее новые данные на входе data поступают на вход шага 1 и процесс циклически повторяется.

Для определения последовательности этих шагов и операций используется тот же метод, что и в реализации на 9 умножителях: создается счетчик invClkCounter, который считает инвертированные импульсы сигнала clk. Только, в отличие от 9 умножителей, где нам приходилось управлять лишь моментами времени перезаписи регистров, в схеме на 3 умножителях нам приходится позаботится также о своевременном переключении мультиплексора и хранении его в нужном состоянии некоторый момент времени, необходимый для расчета промежуточных значении и запоминании их в нужных регистрах.

Перейдем к рассмотрению самой программы.

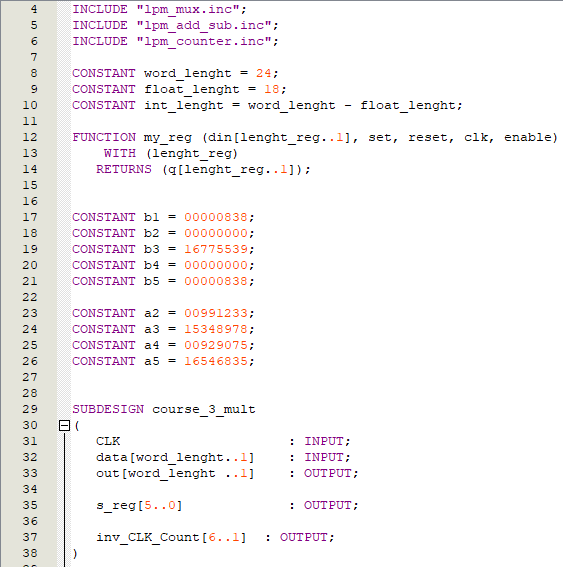


Рисунок . Настройка фильтра на 3 умножителях.

Аналогично, как и в схеме на 9 умножителях описываем коэффициенты, a и b, задаем используемые модули (в качестве нового модуля появился мультиплексор фирмы Altera lpm\_mux), описываем входы и выходы фильтра. (Рисунок 34).

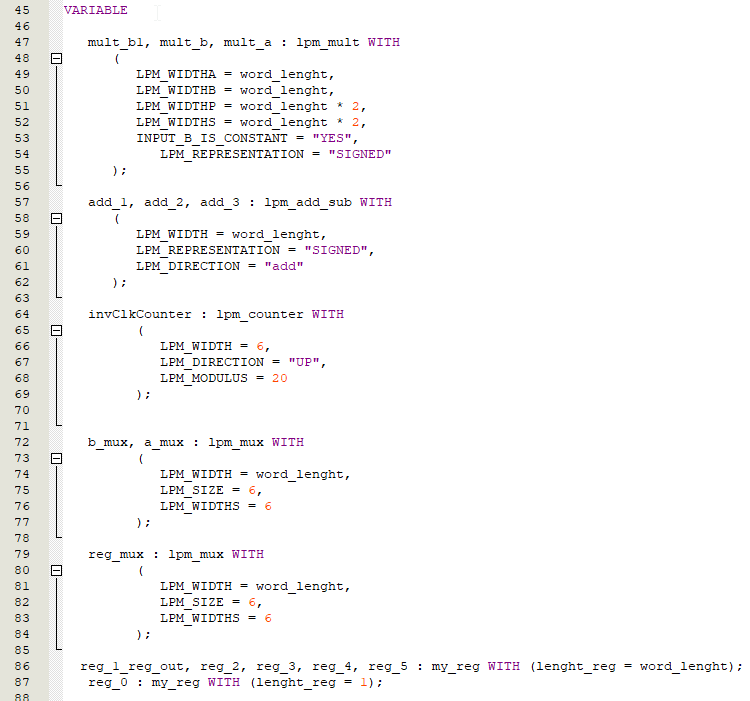


Рисунок . Раздел переменных фильтра на 3 умножителях.

Задаем используемые сумматоры, регистры мультиплексоры. (Рисунок 35).

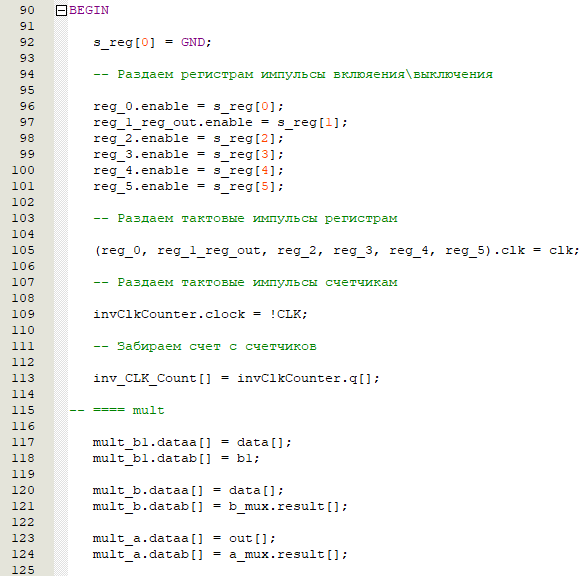


Рисунок . Задание тактовых импульсов, входов мультиплексора, сигналов записи регистров.

Задание тактовых импульсов счетчику и регистрам, описываем входы умножителей. (Рисунок 36)

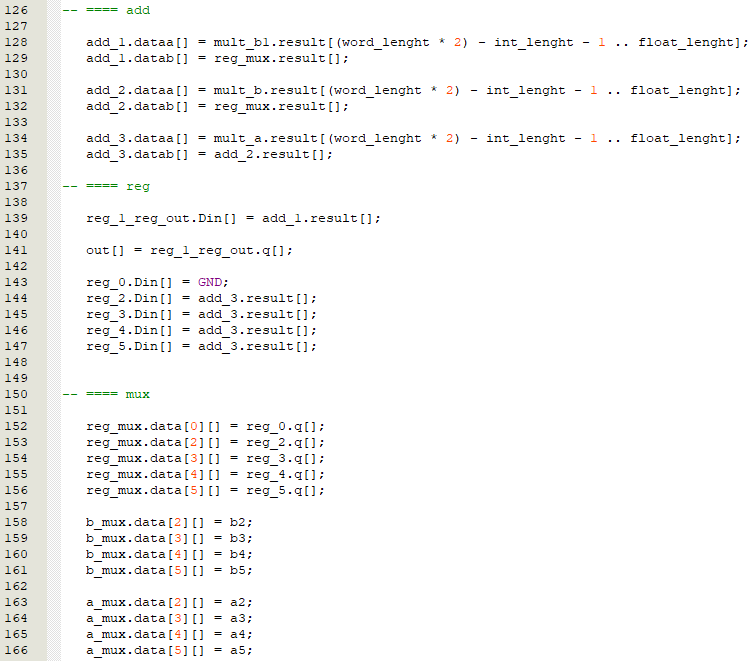


Рисунок . Описываем входы регистров и мультиплексоров, задаем входы-выходы сумматоров.

На рисунке 37 представлены взаимосвязи выводов регистров, мультиплексоров и сумматоров.

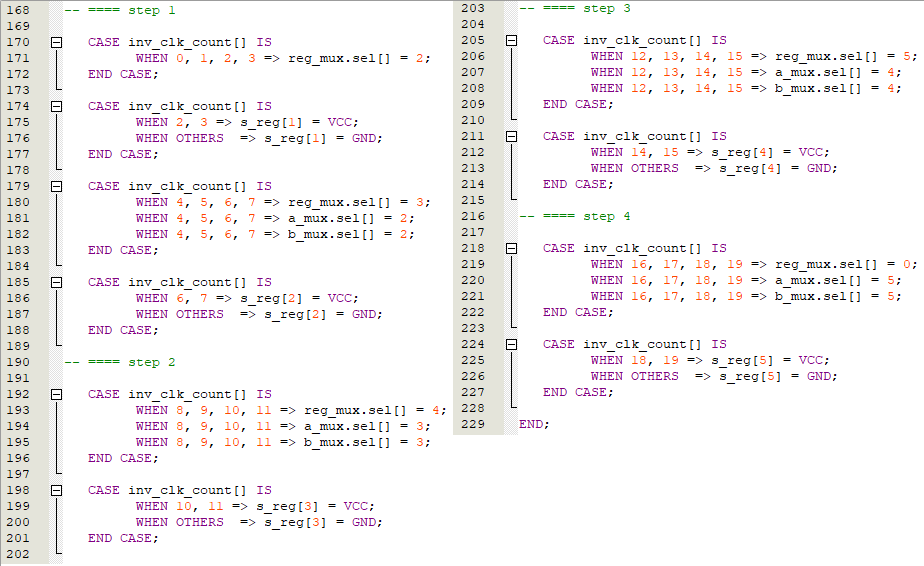


Рисунок . Описание шагов работы фильтра.

Описываем логику работы фильтра - описываем шаги. (Рисунок 38).

Симуляцию результата работы фильтра будем проводить в Max+plus II. Запустим Waveform editor и файл course\_3\_mult.scf для симуляции процесса работы описанного устройства. Период clk = 3.14 микросекунды.

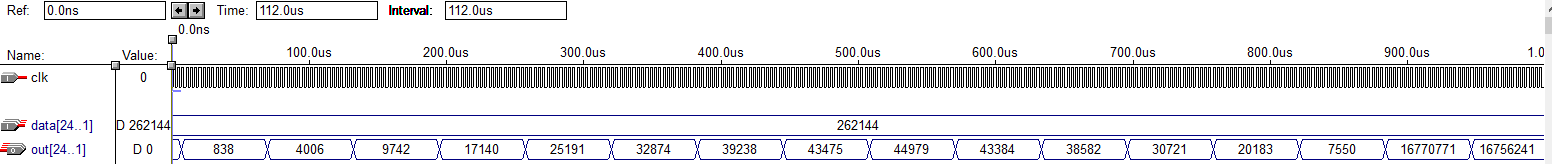


Рисунок . Результат работы фильтра на 3 умножителях.

Как мы видим, результат работы фильтров на 3 и на 9 умножителях совпадают с результатом модели Simulink, значит фильтры разработаны верно. (Рисунок 39). Также проверим работу фильтра при отрицательном скачке (амплитуда -1 вольт). Результаты сходятся, что с моделью Simulink, что с фильтром на 9 умножителях. (Рисунок 40)

# 

Рисунок . Симуляция работы фильтра на 3 умножителях в Max+plus II с отрицательными значениями на входе.

# Заключение

В данной работе были получены навыки работы с системами разработки для проектирования программируемых логических устройств, изучен принцип построения и проектирования односекционного рекурсивного полосового фильтра. Спроектированный фильтр был протестирован в графической среде имитационного моделирования Simulink и в программах Max+plus II, Quaruts II. Считаю, что поставленные задачи были выполнены.

# 

# Список литературы

1. Антонов А. П. Язык описания цифровых устройств Altera HDL Практический курс. – Мск.: РадиоСофт, 2002. – 221 с.
2. Харрис Дэвид М., Харрис Сара Л. Цифровая схемотехника и архитектура компьютера. - ДМК-Пресс, 2018. – 792 с.
3. Бойко В. И., Гуржий А. Н., Жуйков В. Я., Зорн А. А., Спивак В. М., Багрийй В. В. Схемотехника электронных систем. Цифровые устройства. - СПб.: БХВ-Петербург, 2004. - 512 с.
4. А. И. Солонина, Д. А. Улахович, С. М. Арбузов, Е. Б. Соловьева, И. И. Гук. Основы цифровой обработки сигналов. Курс лекций. - СПб.: БХВ-Петербург, 2003.
5. Радиотехнические цепи и сигналы/ Под ред. К. А. Самойло. - М.: Радио и связь, 1982. - 528 с.
6. Казиева Г. С. Основы цифровой обработки сигналов в телекоммуникационных системах. Конспект лекций. - Алматы: АИЭС, 2006. - 46 с.
7. Гольденберг Л. М., Матюшкин Б. Д., Поляк М. Н. Цифровая обработка сигналов. Учебное пособие. - М.: Радио и связь, 1988. - 368 с.

Электронные ресурсы.

1. Цифровые фильтры - https://rucont.ru/efd/206253
2. Transposed Direct-Forms – stanford edu [электронный ресурс]. – https://ccrma.stanford.edu/~jos/fp/Transposed\_Direct\_Forms.html
3. Band Pass Filter: Circuit & Transfer Function (Active & Passive) – electrical 4 u [электронный ресурс]. – https://www.electrical4u.com/band-pass-filter/

# Приложение

**Приложение 1. Фильтр на 9 умножителях.**

TITLE "course\_9\_mult.tdf";

INCLUDE "lpm\_mult.inc";

INCLUDE "lpm\_add\_sub.inc";

INCLUDE "lpm\_counter.inc";

PARAMETERS (lenght\_reg = 5);

CONSTANT word\_lenght = 24;

CONSTANT float\_lenght = 18;

CONSTANT int\_lenght = word\_lenght - float\_lenght;

FUNCTION my\_reg (din[lenght\_reg..1], set, reset, clk, enable)

WITH (lenght\_reg)

RETURNS (q[lenght\_reg..1]);

CONSTANT b1 = 00000838;

CONSTANT b2 = 00000000;

CONSTANT b3 = 16775539;

CONSTANT b4 = 00000000;

CONSTANT b5 = 00000838;

CONSTANT a2 = 00991233;

CONSTANT a3 = 15348978;

CONSTANT a4 = 00929075;

CONSTANT a5 = 16546835;

SUBDESIGN course\_9\_mult

(

CLK : INPUT;

data[word\_lenght..1] : INPUT;

out[word\_lenght ..1] : OUTPUT;

s\_reg[5..1] : OUTPUT;

inv\_CLK\_Count[6..1] : OUTPUT;

)

VARIABLE

mult\_b1, mult\_b2, mult\_b3, mult\_b4, mult\_b5,

mult\_a2, mult\_a3, mult\_a4, mult\_a5 : lpm\_mult WITH

(

LPM\_WIDTHA = word\_lenght,

LPM\_WIDTHB = word\_lenght,

LPM\_WIDTHP = word\_lenght \* 2,

LPM\_WIDTHS = word\_lenght \* 2,

INPUT\_B\_IS\_CONSTANT = "YES",

LPM\_REPRESENTATION = "SIGNED"

);

add\_1, add\_2, add\_3, add\_4, add\_5, add\_6, add\_7, add\_8 : lpm\_add\_sub WITH

(

LPM\_WIDTH = word\_lenght,

LPM\_REPRESENTATION = "SIGNED",

LPM\_DIRECTION = "add"

);

invClkCounter : lpm\_counter WITH

(

LPM\_WIDTH = 6,

LPM\_DIRECTION = "UP",

LPM\_MODULUS = 20

);

reg\_1\_reg\_out, reg\_2, reg\_3, reg\_4, reg\_5 : my\_reg WITH (lenght\_reg = word\_lenght);

BEGIN

-- управление логикой перезаписи регистров

CASE inv\_clk\_count[] IS

WHEN 2, 3 => s\_reg[1] = VCC;

WHEN OTHERS => s\_reg[1] = GND;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 6, 7 => s\_reg[2] = VCC;

WHEN OTHERS => s\_reg[2] = GND;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 10, 11 => s\_reg[3] = VCC;

WHEN OTHERS => s\_reg[3] = GND;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 14, 15 => s\_reg[4] = VCC;

WHEN OTHERS => s\_reg[4] = GND;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 18, 19 => s\_reg[5] = VCC;

WHEN OTHERS => s\_reg[5] = GND;

END CASE;

-- подаем импульсы включения\выключения регистров

reg\_1\_reg\_out.enable = s\_reg[1];

reg\_2.enable = s\_reg[2];

reg\_3.enable = s\_reg[3];

reg\_4.enable = s\_reg[4];

reg\_5.enable = s\_reg[5];

-- тактирование регистров

(reg\_1\_reg\_out, reg\_2, reg\_3, reg\_4, reg\_5).clk = clk;

-- тактирование счетчика

invClkCounter.clock = !CLK;

-- забираем счет с счетчика

inv\_CLK\_Count[] = invClkCounter.q[];

-- b

mult\_b1.dataa[] = data[];

mult\_b1.datab[] = b1;

mult\_b2.dataa[] = data[];

mult\_b2.datab[] = b2;

mult\_b3.dataa[] = data[];

mult\_b3.datab[] = b3;

mult\_b4.dataa[] = data[];

mult\_b4.datab[] = b4;

mult\_b5.dataa[] = data[];

mult\_b5.datab[] = b5;

-- a

mult\_a2.dataa[] = out[];

mult\_a2.datab[] = a2;

mult\_a3.dataa[] = out[];

mult\_a3.datab[] = a3;

mult\_a4.dataa[] = out[];

mult\_a4.datab[] = a4;

mult\_a5.dataa[] = out[];

mult\_a5.datab[] = a5;

-- сумматоры

add\_8.dataa[] = mult\_a5.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_8.datab[] = mult\_b5.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_7.dataa[] = mult\_b4.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_7.datab[] = reg\_5.q[];

add\_6.dataa[] = mult\_a4.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_6.datab[] = add\_7.result[];

add\_5.dataa[] = mult\_b3.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_5.datab[] = reg\_4.q[];

add\_4.dataa[] = mult\_a3.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_4.datab[] = add\_5.result[];

add\_3.dataa[] = mult\_b2.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_3.datab[] = reg\_3.q[];

add\_2.dataa[] = mult\_a2.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_2.datab[] = add\_3.result[];

add\_1.dataa[] = mult\_b1.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_1.datab[] = reg\_2.q[];

-- входы регистров

reg\_5.Din[] = add\_8.result[];

reg\_4.Din[] = add\_6.result[];

reg\_3.Din[] = add\_4.result[];

reg\_2.Din[] = add\_2.result[];

reg\_1\_reg\_out.Din[] = add\_1.result[];

out[] = reg\_1\_reg\_out.q[];

END;

**Приложение 2. Фильтр на 3 умножителях.**

TITLE "course\_3\_mult.tdf";

INCLUDE "lpm\_mult.inc";

INCLUDE "lpm\_mux.inc";

INCLUDE "lpm\_add\_sub.inc";

INCLUDE "lpm\_counter.inc";

CONSTANT word\_lenght = 24;

CONSTANT float\_lenght = 18;

CONSTANT int\_lenght = word\_lenght - float\_lenght;

FUNCTION my\_reg (din[lenght\_reg..1], set, reset, clk, enable)

WITH (lenght\_reg)

RETURNS (q[lenght\_reg..1]);

CONSTANT b1 = 00000838;

CONSTANT b2 = 00000000;

CONSTANT b3 = 16775539;

CONSTANT b4 = 00000000;

CONSTANT b5 = 00000838;

CONSTANT a2 = 00991233;

CONSTANT a3 = 15348978;

CONSTANT a4 = 00929075;

CONSTANT a5 = 16546835;

SUBDESIGN course\_3\_mult

(

CLK : INPUT;

data[word\_lenght..1] : INPUT;

out[word\_lenght ..1] : OUTPUT;

s\_reg[5..0] : OUTPUT;

inv\_CLK\_Count[6..1] : OUTPUT;

)

VARIABLE

mult\_b1, mult\_b, mult\_a : lpm\_mult WITH

(

LPM\_WIDTHA = word\_lenght,

LPM\_WIDTHB = word\_lenght,

LPM\_WIDTHP = word\_lenght \* 2,

LPM\_WIDTHS = word\_lenght \* 2,

INPUT\_B\_IS\_CONSTANT = "YES",

LPM\_REPRESENTATION = "SIGNED"

);

add\_1, add\_2, add\_3 : lpm\_add\_sub WITH

(

LPM\_WIDTH = word\_lenght,

LPM\_REPRESENTATION = "SIGNED",

LPM\_DIRECTION = "add"

);

invClkCounter : lpm\_counter WITH

(

LPM\_WIDTH = 6,

LPM\_DIRECTION = "UP",

LPM\_MODULUS = 20

);

b\_mux, a\_mux : lpm\_mux WITH

(

LPM\_WIDTH = word\_lenght,

LPM\_SIZE = 6,

LPM\_WIDTHS = 6

);

reg\_mux : lpm\_mux WITH

(

LPM\_WIDTH = word\_lenght,

LPM\_SIZE = 6,

LPM\_WIDTHS = 6

);

reg\_1\_reg\_out, reg\_2, reg\_3, reg\_4, reg\_5 : my\_reg WITH (lenght\_reg = word\_lenght);

reg\_0 : my\_reg WITH (lenght\_reg = 1);

BEGIN

s\_reg[0] = GND;

-- Раздаем регистрам импульсы вклюяения\выключения

reg\_0.enable = s\_reg[0];

reg\_1\_reg\_out.enable = s\_reg[1];

reg\_2.enable = s\_reg[2];

reg\_3.enable = s\_reg[3];

reg\_4.enable = s\_reg[4];

reg\_5.enable = s\_reg[5];

-- Раздаем тактовые импульсы регистрам

(reg\_0, reg\_1\_reg\_out, reg\_2, reg\_3, reg\_4, reg\_5).clk = clk;

-- Раздаем тактовые импульсы счетчикам

invClkCounter.clock = !CLK;

-- Забираем счет с счетчиков

inv\_CLK\_Count[] = invClkCounter.q[];

-- ==== mult

mult\_b1.dataa[] = data[];

mult\_b1.datab[] = b1;

mult\_b.dataa[] = data[];

mult\_b.datab[] = b\_mux.result[];

mult\_a.dataa[] = out[];

mult\_a.datab[] = a\_mux.result[];

-- ==== add

add\_1.dataa[] = mult\_b1.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_1.datab[] = reg\_mux.result[];

add\_2.dataa[] = mult\_b.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_2.datab[] = reg\_mux.result[];

add\_3.dataa[] = mult\_a.result[(word\_lenght \* 2) - int\_lenght - 1 .. float\_lenght];

add\_3.datab[] = add\_2.result[];

-- ==== reg

reg\_1\_reg\_out.Din[] = add\_1.result[];

out[] = reg\_1\_reg\_out.q[];

reg\_0.Din[] = GND;

reg\_2.Din[] = add\_3.result[];

reg\_3.Din[] = add\_3.result[];

reg\_4.Din[] = add\_3.result[];

reg\_5.Din[] = add\_3.result[];

-- ==== mux

reg\_mux.data[0][] = reg\_0.q[];

reg\_mux.data[2][] = reg\_2.q[];

reg\_mux.data[3][] = reg\_3.q[];

reg\_mux.data[4][] = reg\_4.q[];

reg\_mux.data[5][] = reg\_5.q[];

b\_mux.data[2][] = b2;

b\_mux.data[3][] = b3;

b\_mux.data[4][] = b4;

b\_mux.data[5][] = b5;

a\_mux.data[2][] = a2;

a\_mux.data[3][] = a3;

a\_mux.data[4][] = a4;

a\_mux.data[5][] = a5;

-- ==== step 1

CASE inv\_clk\_count[] IS

WHEN 0, 1, 2, 3 => reg\_mux.sel[] = 2;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 2, 3 => s\_reg[1] = VCC;

WHEN OTHERS => s\_reg[1] = GND;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 4, 5, 6, 7 => reg\_mux.sel[] = 3;

WHEN 4, 5, 6, 7 => a\_mux.sel[] = 2;

WHEN 4, 5, 6, 7 => b\_mux.sel[] = 2;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 6, 7 => s\_reg[2] = VCC;

WHEN OTHERS => s\_reg[2] = GND;

END CASE;

-- ==== step 2

CASE inv\_clk\_count[] IS

WHEN 8, 9, 10, 11 => reg\_mux.sel[] = 4;

WHEN 8, 9, 10, 11 => a\_mux.sel[] = 3;

WHEN 8, 9, 10, 11 => b\_mux.sel[] = 3;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 10, 11 => s\_reg[3] = VCC;

WHEN OTHERS => s\_reg[3] = GND;

END CASE;

-- ==== step 3

CASE inv\_clk\_count[] IS

WHEN 12, 13, 14, 15 => reg\_mux.sel[] = 5;

WHEN 12, 13, 14, 15 => a\_mux.sel[] = 4;

WHEN 12, 13, 14, 15 => b\_mux.sel[] = 4;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 14, 15 => s\_reg[4] = VCC;

WHEN OTHERS => s\_reg[4] = GND;

END CASE;

-- ==== step 4

CASE inv\_clk\_count[] IS

WHEN 16, 17, 18, 19 => reg\_mux.sel[] = 0;

WHEN 16, 17, 18, 19 => a\_mux.sel[] = 5;

WHEN 16, 17, 18, 19 => b\_mux.sel[] = 5;

END CASE;

CASE inv\_clk\_count[] IS

WHEN 18, 19 => s\_reg[5] = VCC;

WHEN OTHERS => s\_reg[5] = GND;

END CASE;

END;