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

«Разработка цифрового устройства на основе микроконтроллера»

Выполнил: студент

Группы 933

Соловьев Д.А.

Проверил:

Борисов А.Г.

Рязань 2022

# Содержание

[Содержание 2](#_Toc105199161)

[Используемые сокращения 3](#_Toc105199162)

[Введение 4](#_Toc105199163)

[Элементы теории 5](#_Toc105199164)

[Способы реализации цифровых фильтров: 5](#_Toc105199165)

[Моделирование фильтра в MATLAB 8](#_Toc105199166)

[Программная реализация фильтра 17](#_Toc105199167)

[Проверка работы фильтра 24](#_Toc105199168)

[Заключение 31](#_Toc105199169)

[Список литературы 32](#_Toc105199170)

[Приложение 33](#_Toc105199171)

# Используемые сокращения

АЦП – аналого-цифровой преобразователь,

ЦАП – цифро-аналоговый преобразователь,

УВХ - устройство выборки хранения,

БИХ – бесконечная импульсная характеристика.

# 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. Для реализации данного фильтра будем использовать среды программирования MATLAB 2022a и CLion 2022.1.1. Проверку результатов курсового проекта будем выполнять на отладочной плате STM32F407VG6.

# 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 (рис.1).

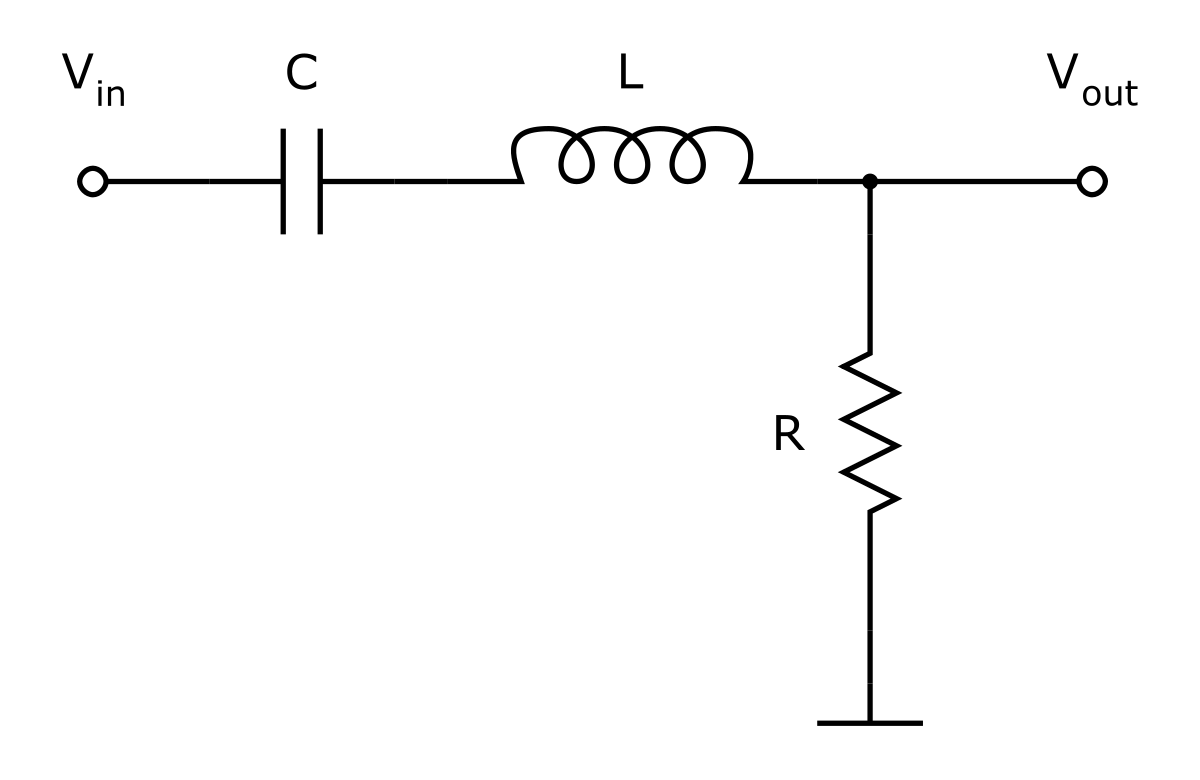


Рис.1. Простейший полосовой фильтр

АЧХ такого фильтра имеет вид (рис.2).

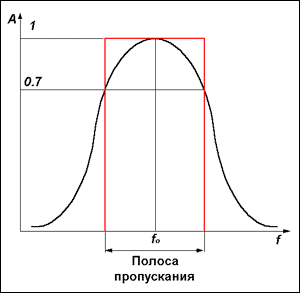


Рис.2. АЧХ полосового фильтра

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 (рис.3).

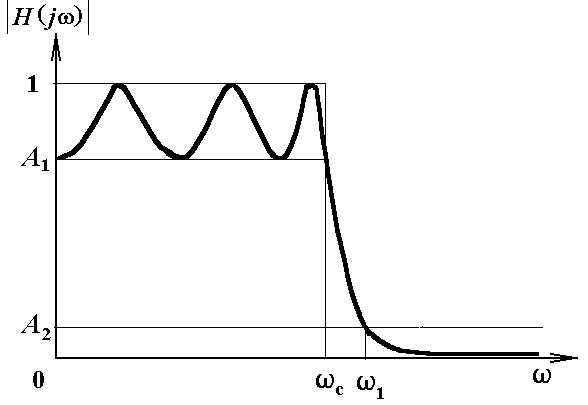


Рис.3. АЧХ фильтра Чебышёва

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.

# Моделирование фильтра в MATLAB

**Исходные данные для проектирования:**

* Band Pass
* Direct Form Ⅱ Transposed
* Chebyshev Type Ⅰ, IIR (Рекурсивный фильтр)
* Specify order = 4
* Section 1
* Fs = 16kHz
* Fc1 = 400 Hz
* Fc2 = 700 Hz

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 (см. рис.4).

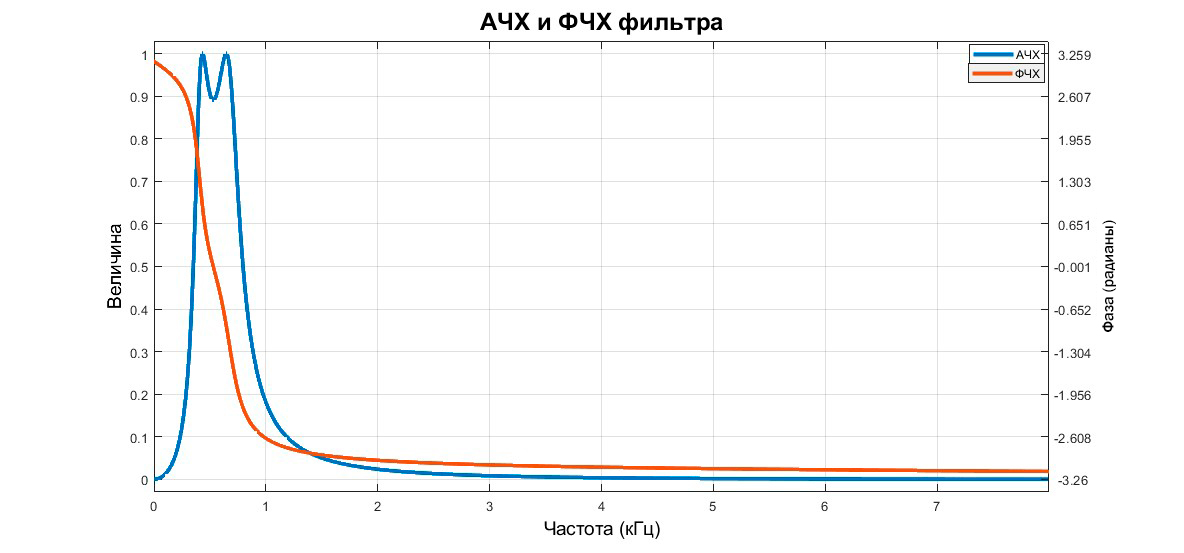


Рис.4. АЧХ и ФЧХ проектируемого фильтра.

Получаем коэффициенты фильтра для разработки проекта (см. рис.5).

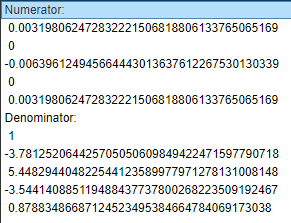


Рис. 5. Коэффициенты фильтра

Чтобы продемонстрировать работоспособность фильтра, в Simulink была создана его принципиальная схема и модель для проверки (рис.6 и рис. 7).

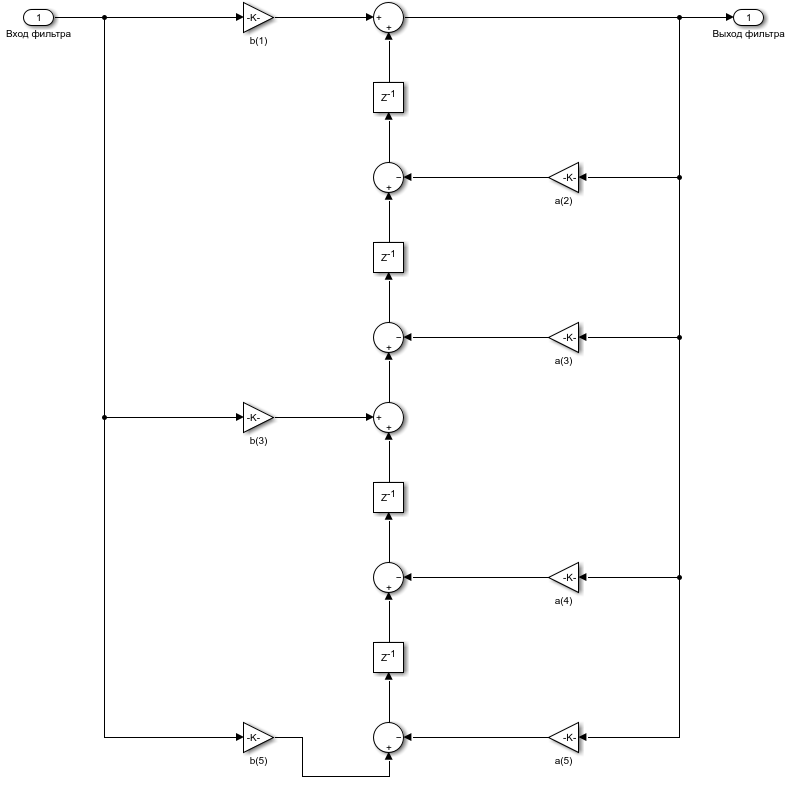


Рис.6. Принципиальная схема фильтра

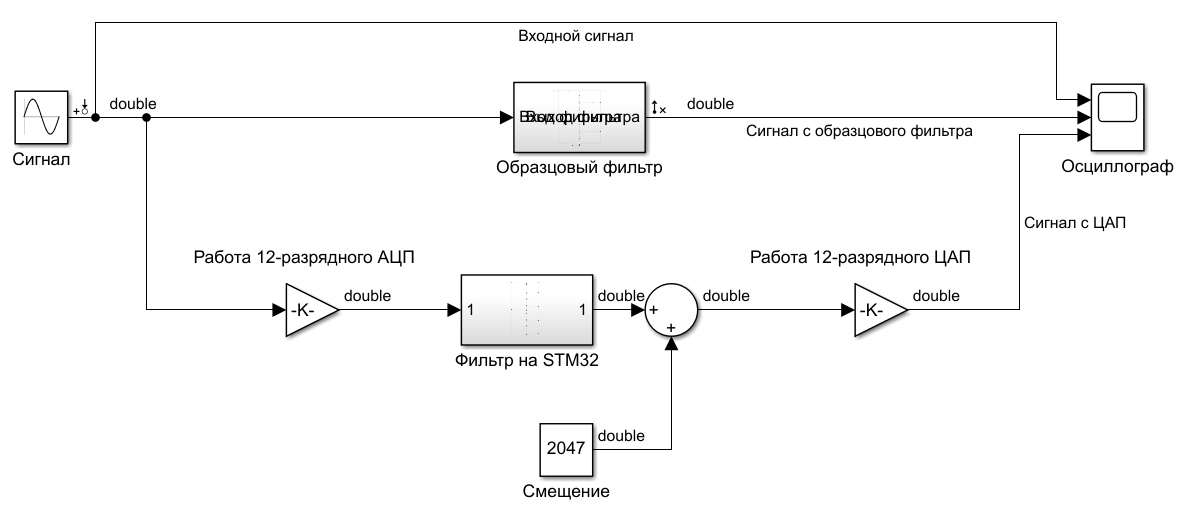


Рис.7. Модель для проверки фильтра

Суть модели для проверки фильтра состоит в следующем. Имеется некий аналоговый сигнал (синусоида с частотой 500 Гц, амплитудой 1 вольт и смещением 1 вольт).

Данный сигнал проходит параллельно через 2 фильтра - образцовый фильтр и фильтр на STM32. Пройдя через образцовый фильтр, сигнал попадает на осциллограф.

Сигнал также попадает на вход 12-ти разрядного АЦП STM32. С выхода АЦП на вход фильтра поступают коды сигнала, и проходят через фильтр.

Так как STM32 может на своих портах может выдать напряжение только в пределах от 0 до 3.3 вольт, а сигнал после фильтрации может быть отрицательным, то к кодам с фильтра на STM32 добавляется постоянное смещение 212/2 = 2047. (где 12 – разрядность АЦП). Далее смещенный сигнал проходит через ЦАП и в виде напряжения попадает на осциллограф.

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 1 В и смещением 1 В на различных частотах: 300 Гц, 400 Гц, 500 Гц, 700 Гц, 1000 Гц (см. рис. 8, 9, 10, 11, 12). На рисунке частоты идут последовательно сверху-вниз, начиная с 300 Гц.

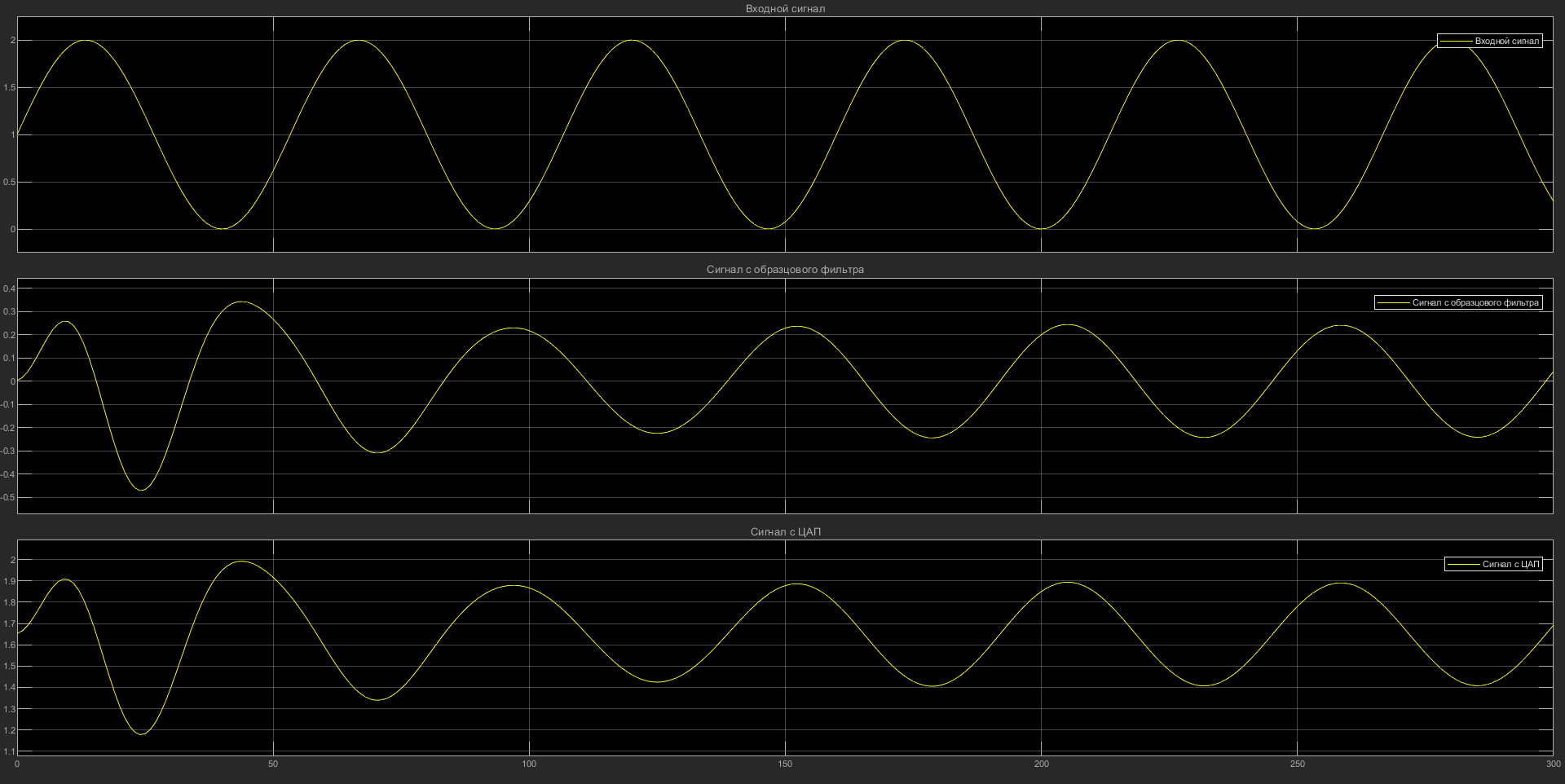


Рис. 8. Сигнал с частотой 300 Гц

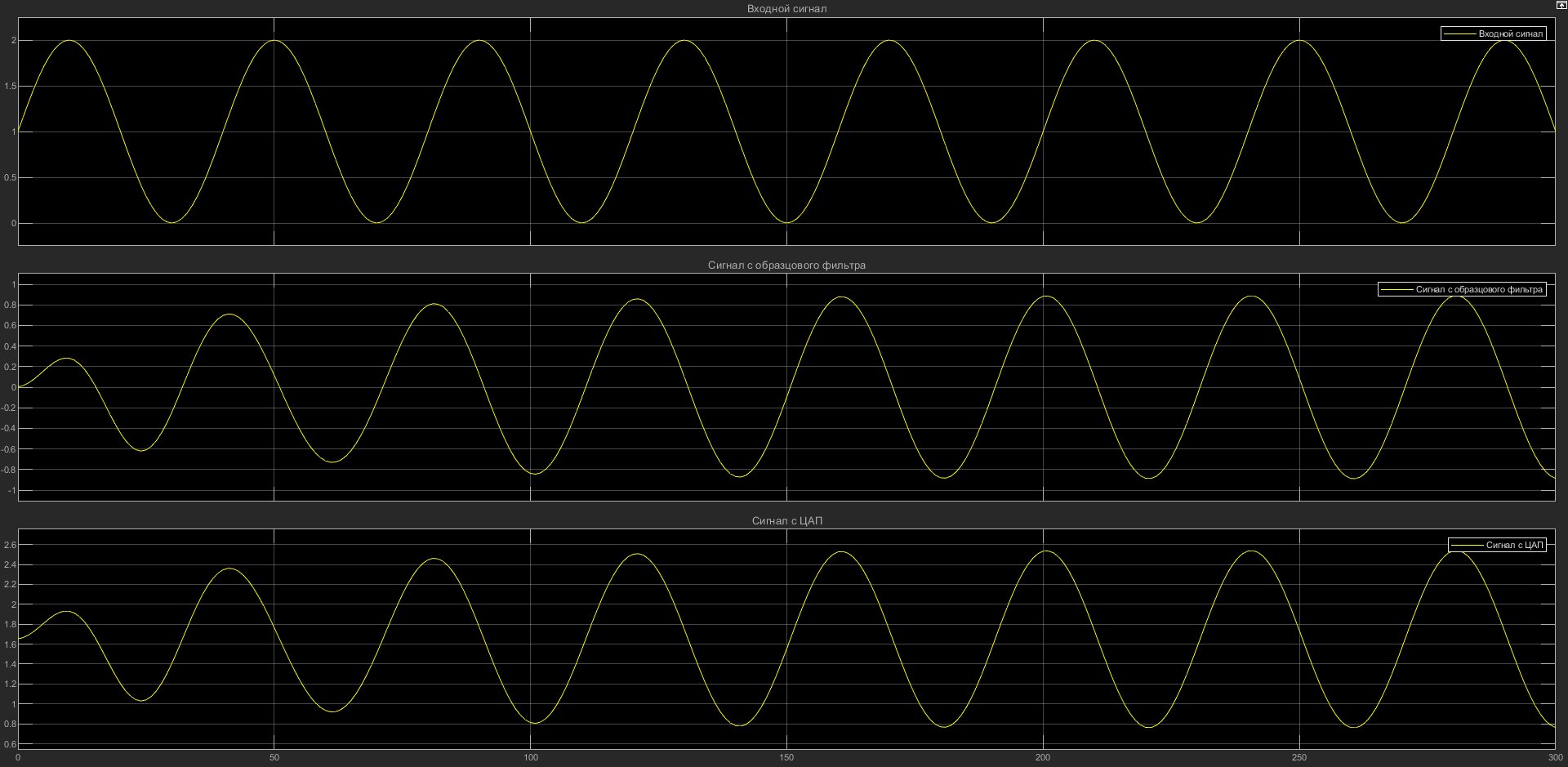


Рис. 9. Сигнал с частотой 400 Гц

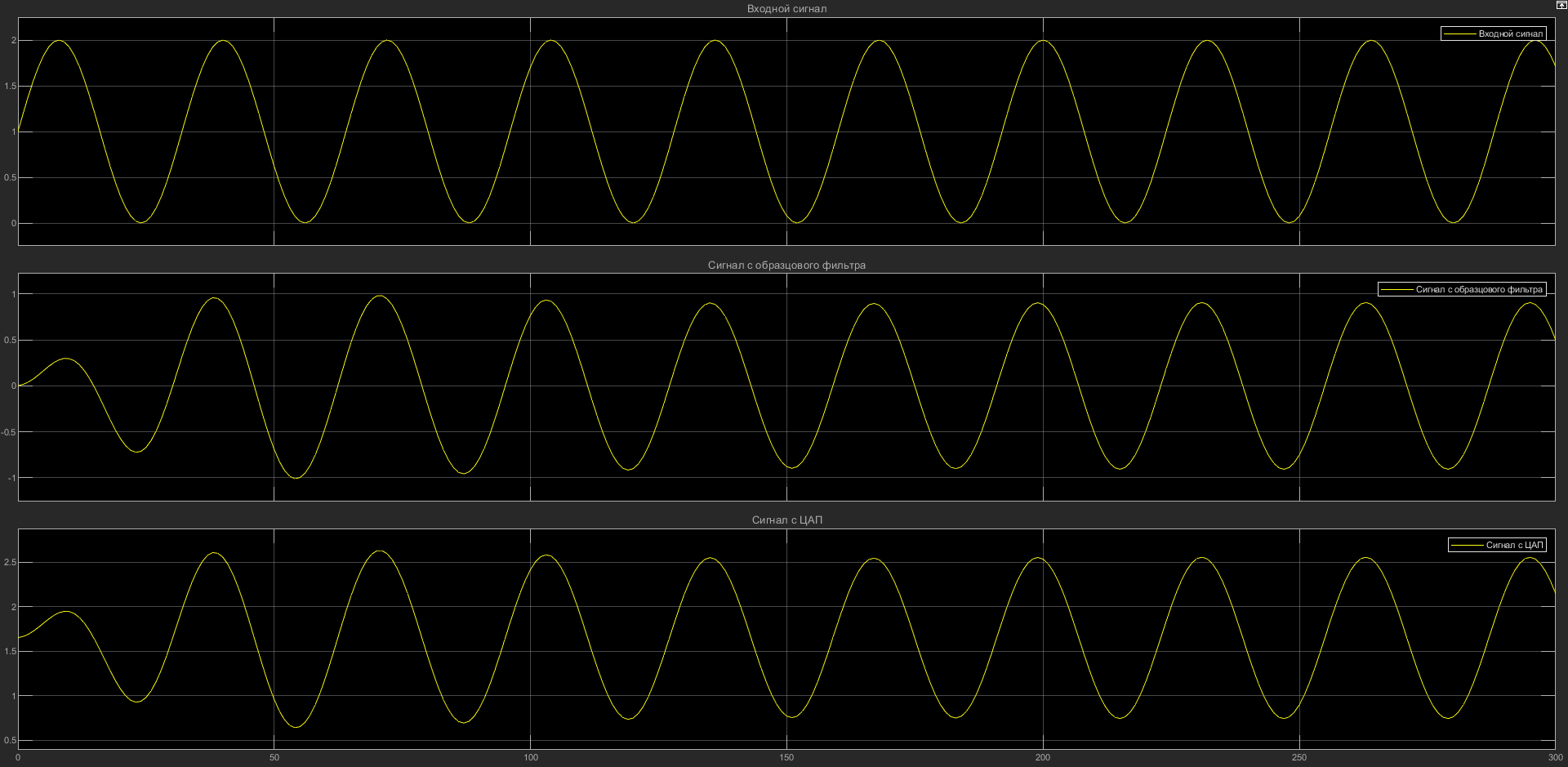


Рис. 10. Сигнал с частотой 500 Гц

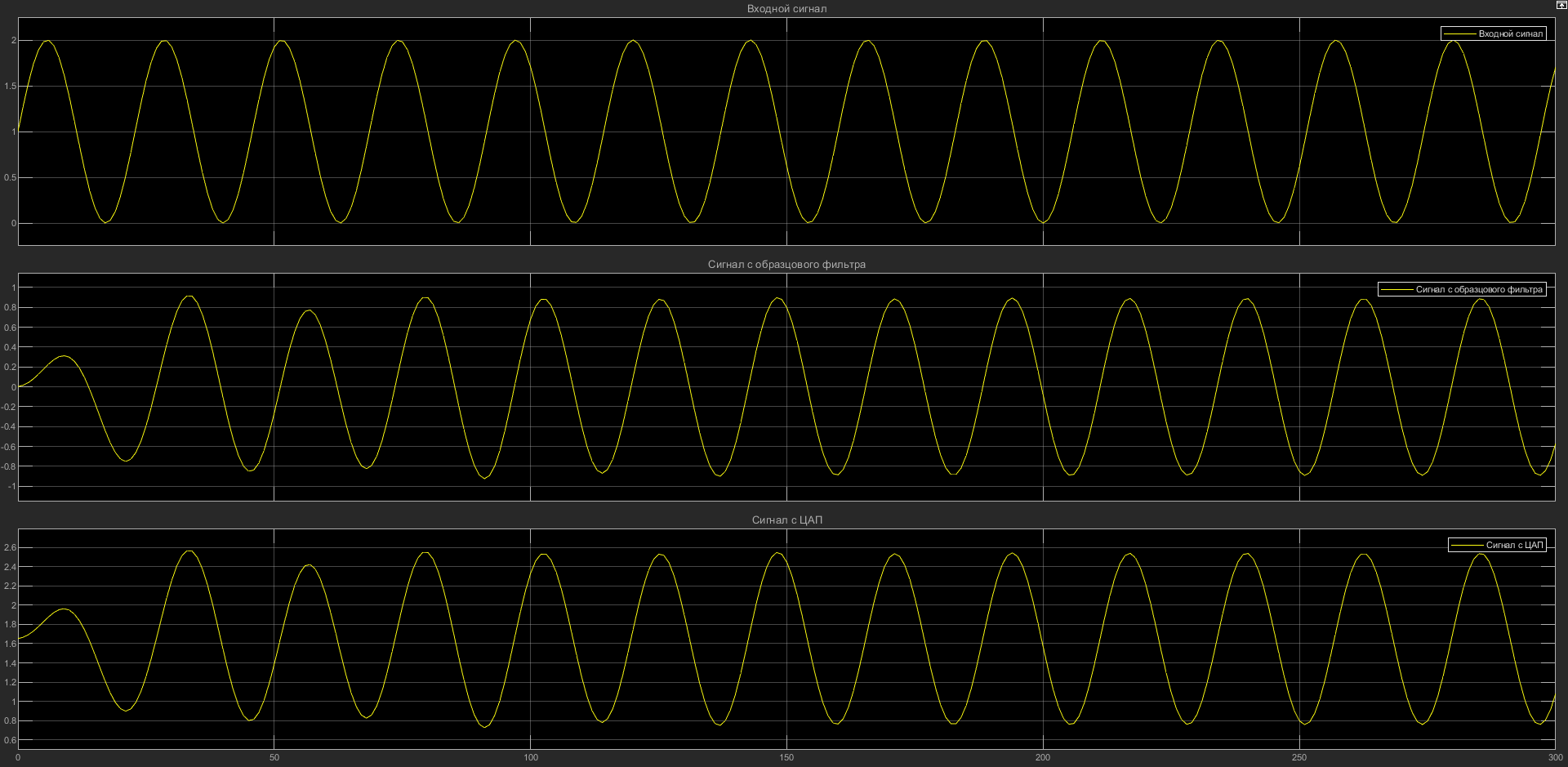


Рис. 11. Сигнал с частотой 700 Гц

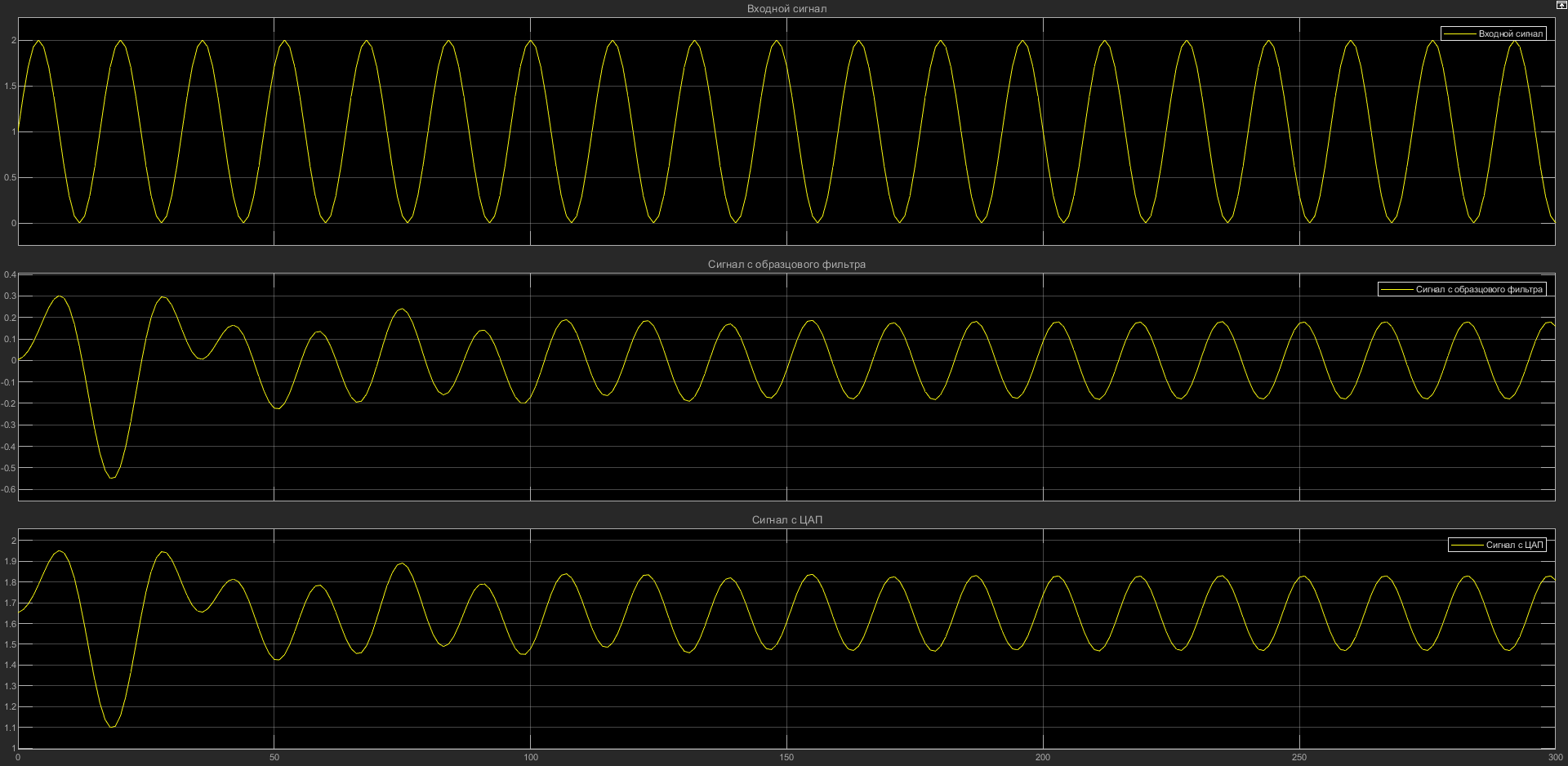


Рис. 12. Сигнал с частотой 1000 Гц

Из полученных осциллограмм видно, что за пределами полосы пропускания сигнал существенно ослабевает. Сделаем вывод, что коэффициенты фильтра 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 Си. В качестве среды разработки будем использовать CLion 2022.1.1. Для написания кода использовали библиотеку libopencm3. Это библиотека низкоуровневого аппаратного обеспечения с открытым исходным кодом для микроконтроллеров ARM Cortex-M4, -M3, -M0.

Структура программы состоит из 8 функций:

1. void setup\_dac() – инициализация ЦАП
2. void setup\_adc() - инициализация АЦП
3. uint16\_t first\_sample() - Снятие первого отсчета с АЦП, для правильной работы
4. uint16\_t adc() – работа АЦП
5. void dac(uint16\_t outputData) – работа ЦАП
6. void setup\_timer(uint16\_t Fs) – настройка таймера
7. double IIRFilter(double x) – цифровая фильтрация
8. int main() – основная функция, в ней также происходит разгон процессора

Принцип работы фильтра на микроконтроллере продемонстрирован на рисунке (см. рис. 13).

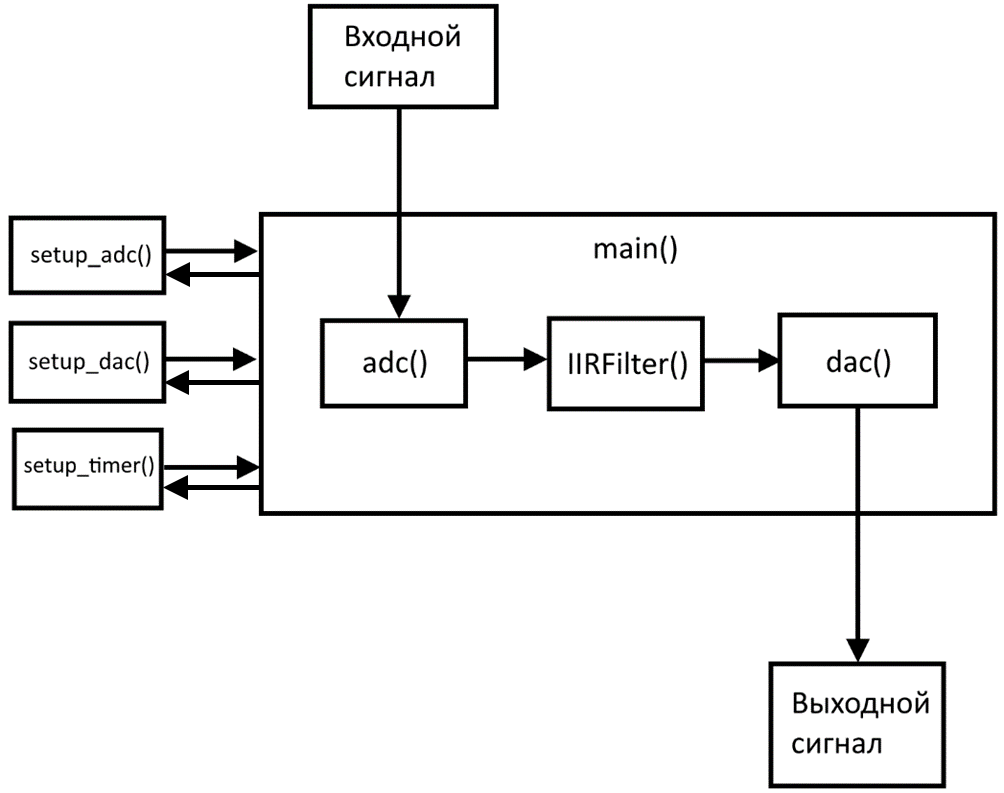


Рис. 13. Принцип работы фильтра

После запуска программы происходит запуск функции main(), которая запускает единожды функции setup\_adc(), setup\_dac, setup\_timer(). Эти функции настраивают работу АЦП, ЦАП и таймера соответственно. В функции main() также происходит тактирование портов и разгон процессора.

В функции main() имеется бесконечный цикл, в котором и происходят все основные процессы работы фильтра.

Функция adc() считывает данные с 12-ти разрядного АЦП. Эти данные попадают в IIRFilter(), где происходит фильтрация. Отфильтрованные данные округляются до ближайшего целого функцией round() и, складываясь с постоянным смещением, в виде кода поступают на вход 12-ти разрядного ЦАП.

# Программная реализация фильтра

Код программы представлен в приложении (см. Приложение 1).

Следует отметить, что программа была создана и проверена на работоспособность только в среде CLion 2022.1.1.

Разберём подробно каждую из основных функций программы:

* подключаем необходимые C библиотеки, и части библиотеки libopencm3

#include <libopencm3/stm32/rcc.h>

#include <libopencm3/stm32/gpio.h>

#include <libopencm3/stm32/dac.h>

#include <libopencm3/stm32/adc.h>

#include <libopencm3/stm32/timer.h>

#include <stdio.h>

#include <stdint.h>

#include <math.h>

* Задаем частоту дискретизации фильтра, она понадобится для дальнейшей настройки таймера

#define FS 16000

* Обозначаем прототипы используемых функций

void setup\_dac();

void setup\_adc();

uint16\_t first\_sample();

uint16\_t adc();

void dac(uint16\_t outputData);

void setup\_timer (uint16\_t Fs);

double IIRFilter(double x);

* Объявляем массив и задаем размер массива коэффициентов, это необходимо для фильтрации

double v[4] = {0, 0, 0, 0};

int N = 5;

* Объявляем структуру, описывающую информацию для разгона процессора. Следует отметить, что мы используем внешний кварцевый резонатор, так как он более стабилен, по сравнению с внутренним

const struct rcc\_clock\_scale\* clocks = rcc\_hse\_8mhz\_3v3[RCC\_CLOCK\_3V3\_84MHZ];

* Объявляем массивы коэффициентов

double b[] = {0.00319806247283222151, 0, -0.00639612494566444301, 0, 0.00319806247283222151};

double a[] = {1, -3.78125206442570505, 5.44829440482254412, -3.54414088511948755, 0.878834866871245235};

**Основная функция main()**

int main () {

* Выполняем разгон процессора. Следует отметить, что процедура фильтрации занимает довольно длительное время, и стандартной частоты работы процессора может не хватать, что может привести к некорректным результатам фильтрации. Для этого и производится разгон процессора минимум в 2-3 раза от частоты дискретизации фильтра.

rcc\_clock\_setup\_pll(clocks);

* Выполняем тактирование портов A

rcc\_periph\_clock\_enable(RCC\_GPIOA);

* Производим предварительную настройку таймера, АЦП и ЦАП

setup\_dac();

setup\_adc();

setup\_timer(FS);

* Выполняем тактирование портов D

rcc\_periph\_clock\_enable(RCC\_GPIOD);

* Настраиваем порт ввода-вывода D15

gpio\_mode\_setup (GPIOD, GPIO\_MODE\_OUTPUT, GPIO\_PUPD\_NONE, GPIO15);

* Объявляем рабочие переменные для: записи данных с АЦП, записи данных с фильтра, передачи данных в АЦП

uint16\_t data\_adc = first\_sample();

double data\_filtered\_double = 0;

uint16\_t data\_filtered\_12 = 0;

* Основной бесконечный цикл

while (1) {

if (timer\_get\_flag(TIM6, TIM\_SR\_UIF)) {

* Устанавливаем порт D15

gpio\_set(GPIOD, GPIO15);

* Забираем данные с АЦП

data\_adc = adc();

* Записываем данные с фильтра

data\_filtered\_double = IIRFilter(data\_adc);

* Округляем и добавляем смещение

data\_filtered\_12 = round(data\_filtered\_double) + 2047;

* Передаем в ЦАП

dac(data\_filtered\_12);

timer\_clear\_flag(TIM6, TIM\_SR\_UIF);

* Блокирующий оператор - появился результат АЦП

while (! adc\_eoc(ADC1));

* Очищаем порт D15

gpio\_clear(GPIOD, GPIO15);

}

}

}

**Функция цифровой фильтрации**

* В данной функции представлен алгоритм фильтрации Direct Form 2 Transposed

double IIRFilter(double x){

double y = 0; // выход

int i = 0;

y = v[0] + b[0]\*x;

for (; i < N-2; i++)

v[i] = v[i+1] + b[i+1]\*x - a[i+1]\*y;

v[N-2] = b[N-1]\*x - a[N-1]\*y;

return y;

}

**Функция инициализации ЦАП**

void setup\_dac() {

* Настройка порта A4 для выхода цифрового сигнала (подключения осциллографа)

gpio\_mode\_setup (GPIOA, GPIO\_MODE\_ANALOG, GPIO\_PUPD\_NONE, GPIO4);

* Тактирование ЦАП

rcc\_periph\_clock\_enable(RCC\_DAC);

* Задание канала ЦАП

dac\_enable(CHANNEL\_1);

}

**Функция инициализации АЦП**

void setup\_adc() {

* Настройка порта A0 для входа аналогового сигнала (подключения генератора)

gpio\_mode\_setup (GPIOA, GPIO\_MODE\_ANALOG, GPIO\_PUPD\_NONE, GPIO0);

* Тактирование АЦП

rcc\_periph\_clock\_enable(RCC\_ADC1);

* Переход в режим одноканального сканирования каналов

adc\_disable\_scan\_mode(ADC1);

* Одиночное преобразование

adc\_set\_single\_conversion\_mode(ADC1);

* Тактирования блока преобразователя УВХ, примерно 15 тактов (в тактах АЦП)

adc\_set\_sample\_time(ADC1, ADC\_CHANNEL0, ADC\_SMPR\_SMP\_15CYC);

* Подаем опорное напряжение на внутренний АЦП

adc\_power\_on(ADC1);

}

**Функция снятия первого отсчета с АЦП, для правильной работы**

uint16\_t first\_sample() {

* Запуск одиночного преобразования

adc\_start\_conversion\_regular(ADC1);

* Блокирующий оператор - появился результат АЦП

while (! adc\_eoc(ADC1));

* Записываем результат с АЦП

uint16\_t adc\_retult = adc\_read\_regular(ADC1);

* Запуск следующего преобразования

adc\_start\_conversion\_regular(ADC1);

return adc\_retult;

}

**Функция снятия данных с АЦП**

uint16\_t adc() {

* Записываем результат с АЦП

uint16\_t temp\_data\_from\_adc = adc\_read\_regular(ADC1);

* Запуск следующего преобразования

adc\_start\_conversion\_regular(ADC1);

return temp\_data\_from\_adc;

}

**Функция передачи отфильтрованных данных на ЦАП**

void dac(uint16\_t outputData) {

dac\_load\_data\_buffer\_single(outputData, RIGHT12, CHANNEL\_1 );

}

**Функция настройки таймера**

* Каждый из базовых таймеров включает в себя: 16-битные регистры деления частоты PSC и регистр переполнения с автоматической перезагрузкой ARR, а также схему синхронизации для запуска DAC и схему генерации прерывания. Частота тактового сигнала равна 84МГц, исходя из формулы частота дискретизации будет определяться:

F

Частота дискретизации задаётся путём записи соответствующих значений в регистры ARR и PSC счетчика. Зная частоту дискретизации, можем рассчитать значения соответствующих регистров.

void setup\_timer (uint16\_t Fs) {

* Тактирование таймера

rcc\_periph\_clock\_enable(RCC\_TIM6);

* Задание PSC и ARR

timer\_set\_prescaler(TIM6, 1 - 1);

timer\_set\_period(TIM6, 5250 - 1);

timer\_enable\_irq(TIM6, TIM\_DIER\_UIE);

* Включение таймера

timer\_enable\_counter(TIM6);

}

# Проверка работы фильтра

Для проверки работы фильтра будем использовать отладочную плату Analog Discovery и программу Digilent WaveForms.

C помощью генератора сигналов будем подавать синусоидальный сигнал с различными частотами на вход фильтра – контакт PA0, c амплитудой 1 В со смещением 1 В. Результат будем снимать осциллографом с выхода – контакта PA4.

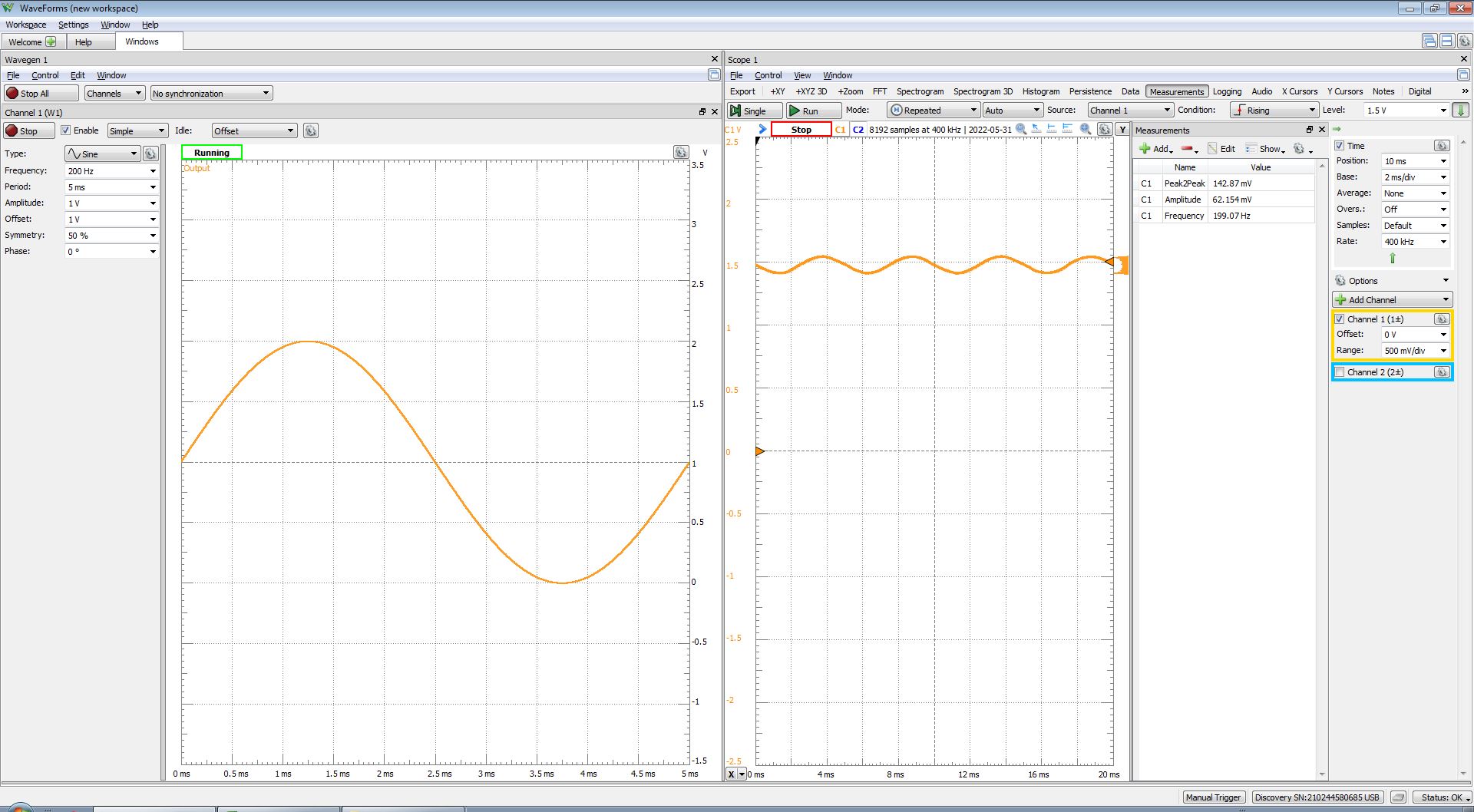
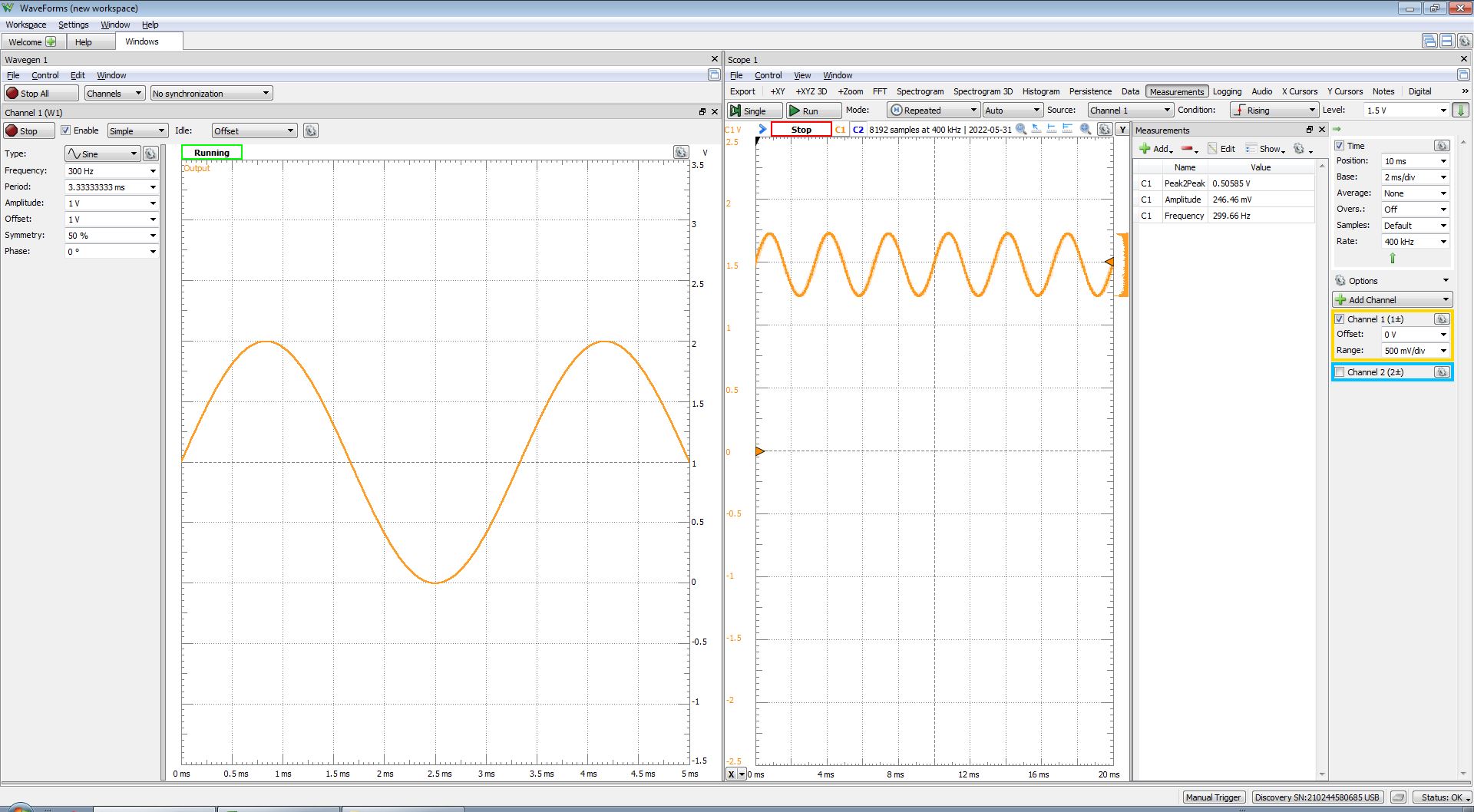
Подаем сигнал с частотой 200 Гц и 300 Гц. Частота данного сигнала лежит за пределами полосы пропускания, что мы можем видеть на соответствующем рисунке (см.рис. 14, 15).

Рис. 14. Сигнал 200 Гц. График слева – генератор, справа – осциллограф.

Рис. 15. Сигнал 300 Гц. График слева – генератор, справа – осциллограф.

Подадим сигнал в полосе пропускания – 500 Гц, 600 Гц. По картинке видно, что сигнал проходит практически без изменений. (см.рис. 16, 17)

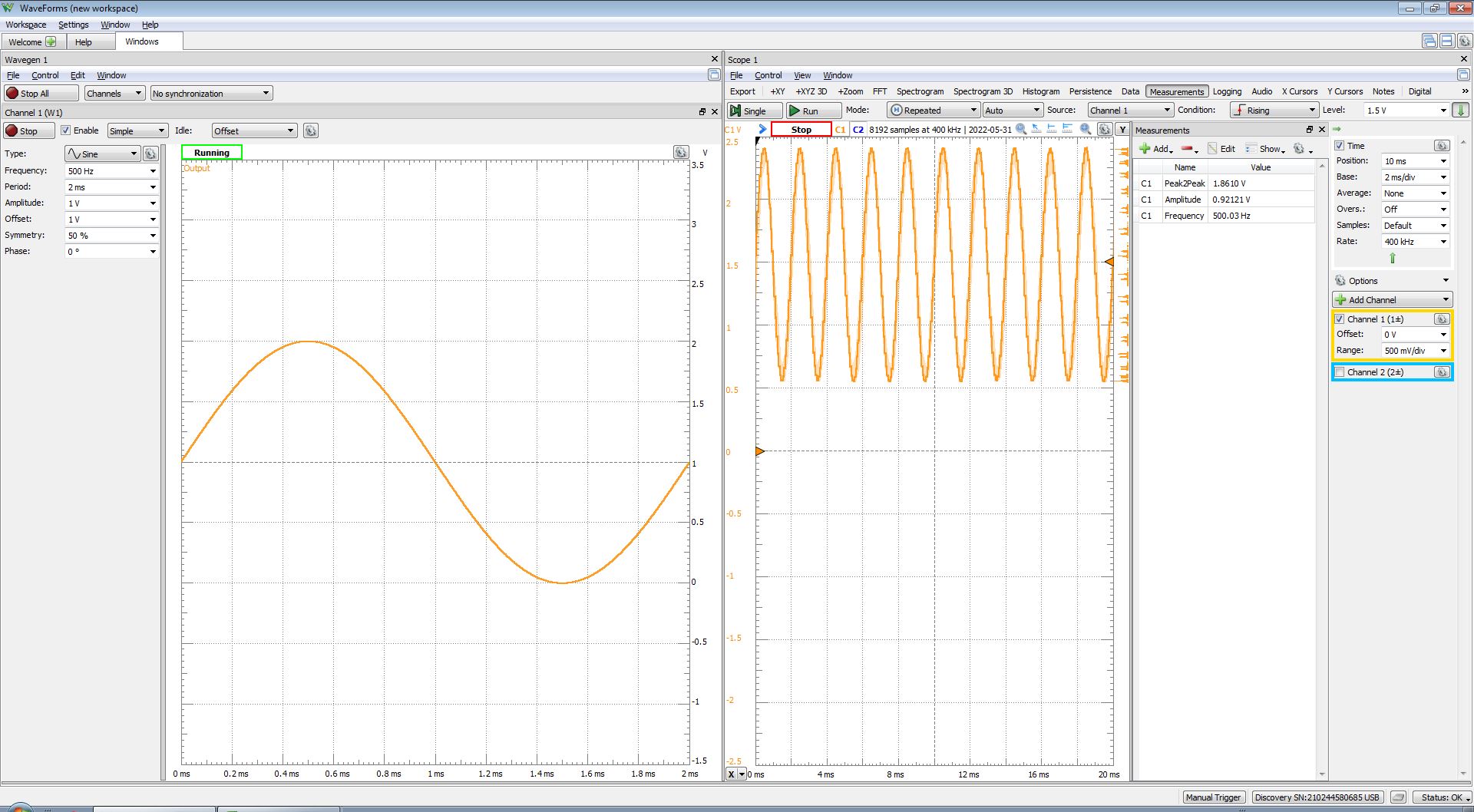


Рис. 16. Сигнал 500 Гц. График слева – генератор, справа – осциллограф.

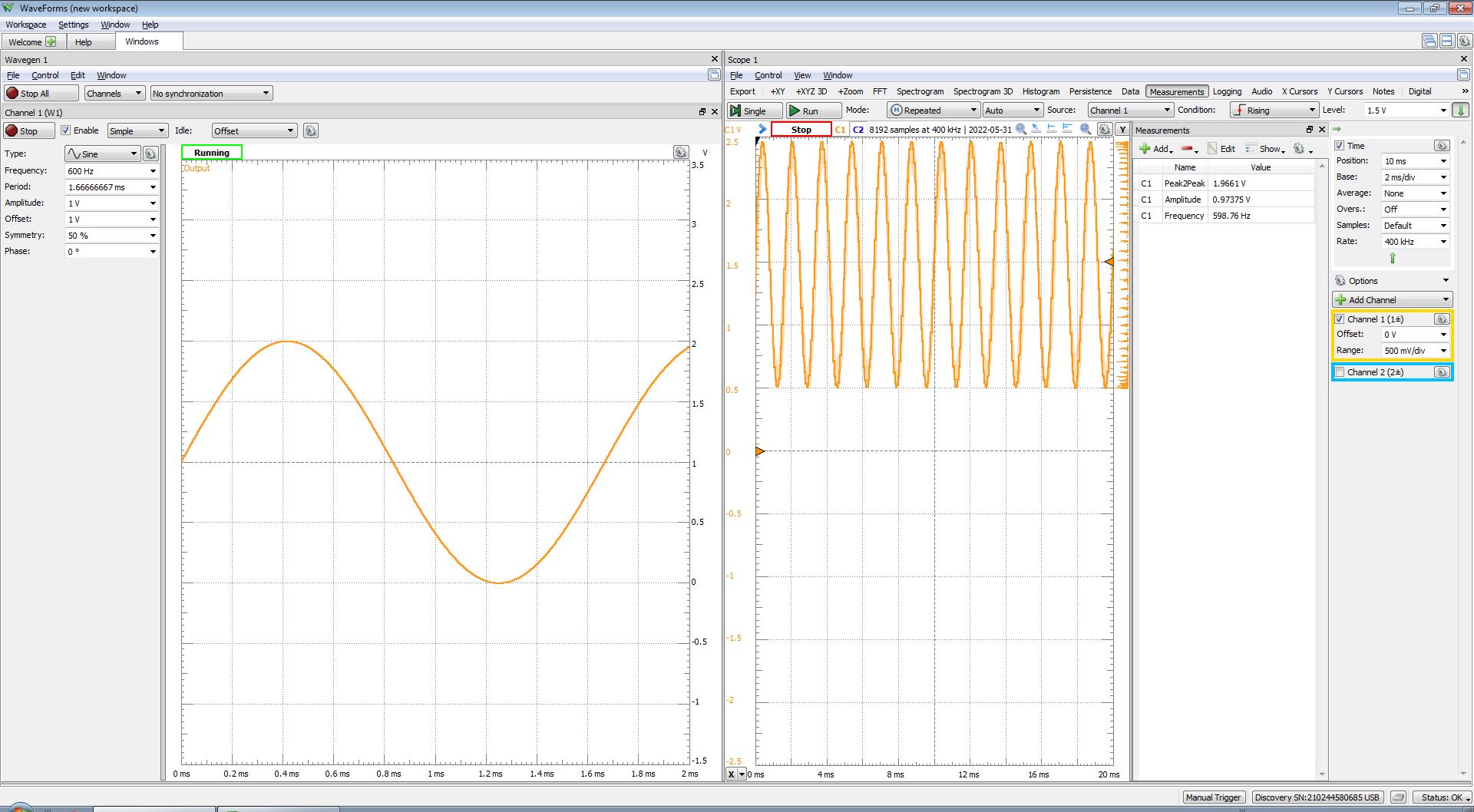


Рис. 17. Сигнал 600 Гц. График слева – генератор, справа – осциллограф.

Подадим за полосой пропускания – 1000 Гц, 2000 Гц. По картинке видно, что сигнал существенно ослабляется. (см.рис. 18, 19)

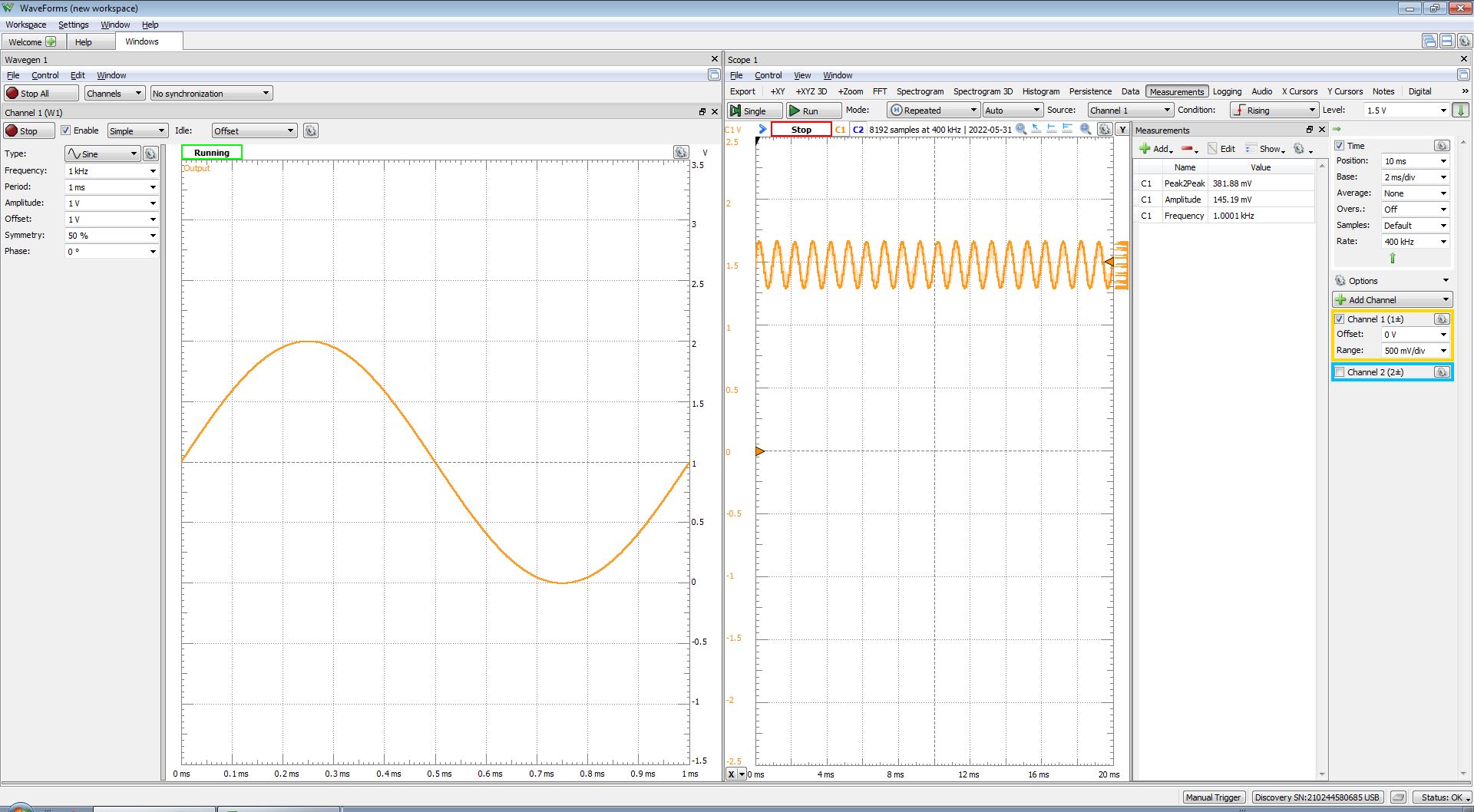


Рис. 18. Сигнал 1000 Гц. График слева – генератор, справа – осциллограф.

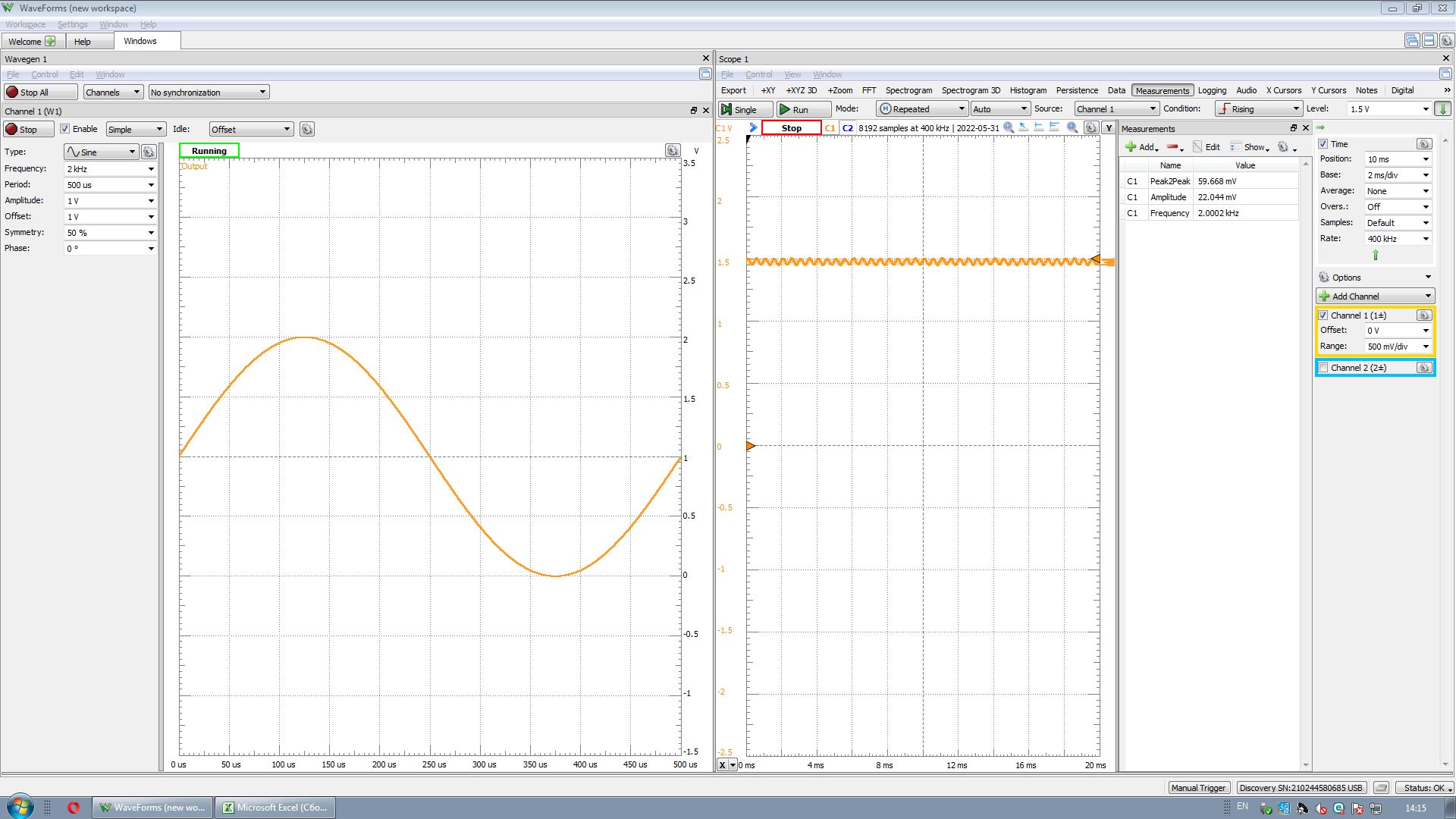


Рис. 19. Сигнал 2000 Гц. График слева – генератор, справа – осциллограф.

Из полученных осциллограмм видно, что в полосе пропускания амплитуда сигнала равна приблизительно 0.9 от исходной, а вне полосы пропускания сигнал существенно ослабевает. Зависимости, полученные экспериментальным путём, соответствуют данным, полученным на основе моделирования фильтра в Simulink.

Сравним АЧХ фильтра из Matlab и Waveform. (см. рис. 20, 21)



Рис. 20. АЧХ фильтра в Matlab

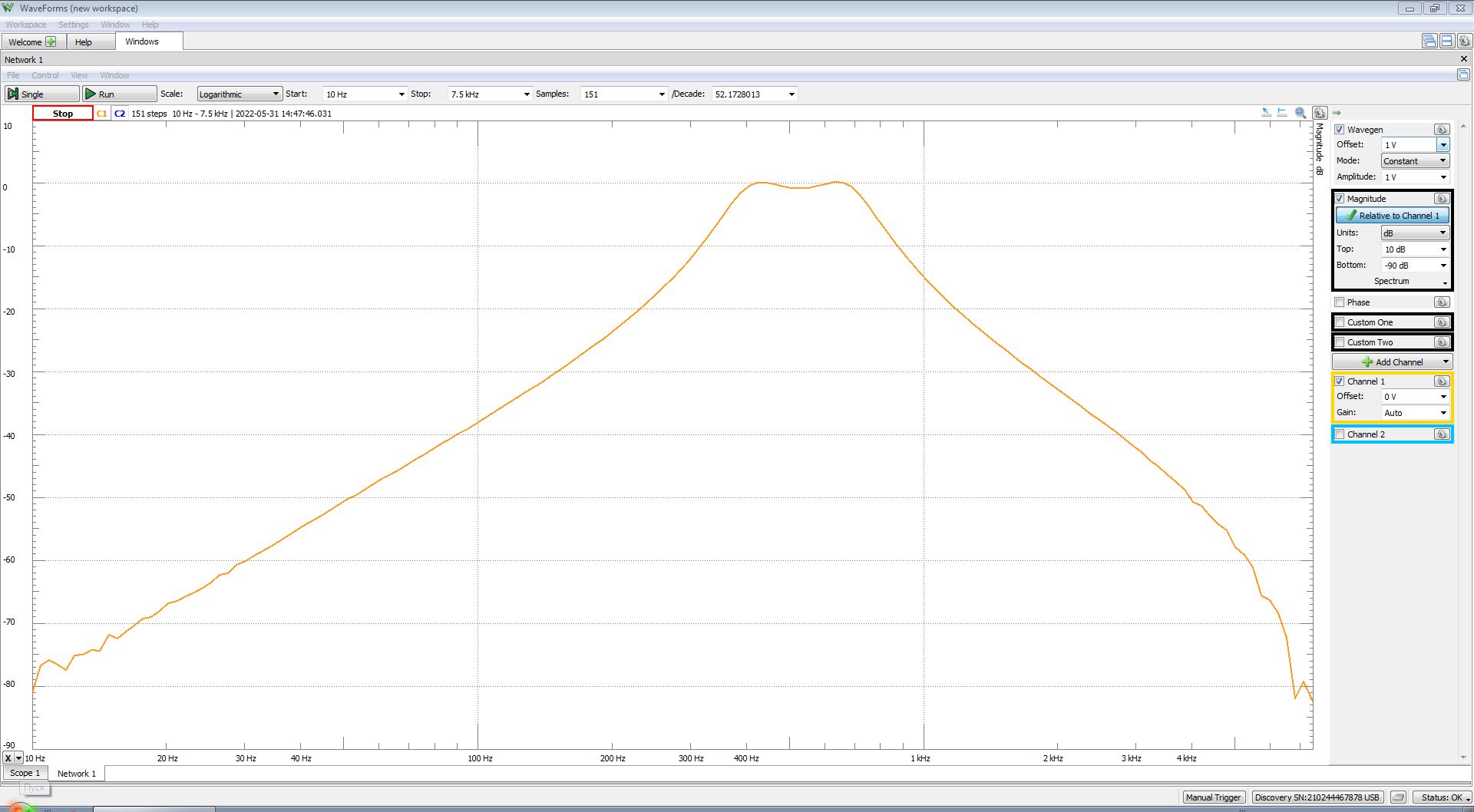


Рис. 21. АЧХ фильтра в Waveform

Отметим следующее:

* АЧХ в Waveform начинает вести себя нестабильно при высоких и низких частотах сигнала (больше ~7 кГц и меньше ~20 Гц).
* Частоты среза и усиления по амплитуде обеих АЧХ практически совпадают.
* В полосе пропускания обоих АЧХ видны характерные для фильтра Чебышева 1 рода пульсации.

Сравним коэффициенты ослабления из Matlab и из Waveform на 6 разных частотах (4 в полосе заграждения и 2 в полосе пропускания). Частоты будут следующие: 200 Гц, 300 Гц, 500 Гц, 600 Гц, 1000 Гц, 2000 Гц. (см. рис. 14-19; 22)

Коэффициент ослабления можно по АЧХ или по формуле:

Результаты расчета коэффициент ослабления приведены ниже.

|  |  |  |
| --- | --- | --- |
| Частоты (Гц) | Коэффициенты ослабления Matlab | Коэффициент ослабления Waveform |
| 200 | 0.06467 | 0.062142 / 1 = 0.062142 |
| 300 | 0.24460 | 0.24646/ 1 = 0.24646 |
| 500 | 0.90496 | 0.92121 / 1 = 0.92121 |
| 600 | 0.96561 | 0.97375 / 1 = 0.97375 |
| 1000 | 0.18065 | 0.14519 / 1 = 0.14519 |
| 2000 | 0.02341 | 0.022044 / 1 = 0.022044 |

По таблице видно, что результаты коэффициентов ослабления из Matlab и из Waveform очень схожи.



Рис. 21. АЧХ фильтра в Matlab

Исходя из вышеперечисленного делаем вывод, что расчеты проведены правильно и фильтр работает.

# Заключение

В данной работе были получены навыки работы с микроконтроллером, изучен принцип построения и проектирования односекционного рекурсивного полосового фильтра. Спроектированный фильтр был протестирован в графической среде имитационного моделирования Simulink и в среде программирования CLion 2022.1.1, так же была проведена проверка фильтра опытным путём на отладочной плате STM32F407VG6. Считаю, что поставленные задачи были выполнены.

# 

# Список литературы

1. 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.
2. 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.
3. Радиотехнические цепи и сигналы/ Под ред. К. А. Самойло. - М.: Радио и связь, 1982. - 528 с.
4. Казиева Г. С. Основы цифровой обработки сигналов в телекоммуникационных системах. Конспект лекций. - Алматы: АИЭС, 2006. - 46 с.
5. 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/
4. АЦП – wikipedia [электронный ресурс]. - https://ru.wikipedia.org/wiki/аналого-цифровой\_преобразователь

# Приложение

Приложение 1.

Код программы в CLion 2022.1.1:

#include <libopencm3/stm32/rcc.h>  
#include <libopencm3/stm32/gpio.h>  
#include <libopencm3/stm32/dac.h>  
#include <libopencm3/stm32/adc.h>  
#include <libopencm3/stm32/timer.h>  
  
#include <stdio.h>  
#include <stdint.h>  
#include <math.h>  
#include <stdlib.h>  
  
#define FS 16000  
  
*void* setup\_dac();  
*void* setup\_adc();  
uint16\_t first\_sample();  
uint16\_t adc();  
*void* dac(uint16\_t outputData);  
*void* setup\_timer (uint16\_t Fs);  
*double* IIRFilter(*double* x);  
  
*double* v[4] = {0, 0, 0, 0};  
*int* N = 5; *// размер массива кэфов  
  
const struct* rcc\_clock\_scale\* clocks = &rcc\_hse\_8mhz\_3v3[RCC\_CLOCK\_3V3\_84MHZ];  
  
*// ===============================================  
  
double* b[] = {0.00319806247283222151, 0, -0.00639612494566444301, 0, 0.00319806247283222151};  
*double* a[] = {1, -3.78125206442570505, 5.44829440482254412, -3.54414088511948755, 0.878834866871245235};  
  
*// ===============================================  
  
int* main () {  
  
 *// Разгон процессора* rcc\_clock\_setup\_pll(clocks);  
  
 *// тактирование портов A* rcc\_periph\_clock\_enable(RCC\_GPIOA);  
  
 setup\_dac();  
 setup\_adc();  
 setup\_timer(FS);  
  
 rcc\_periph\_clock\_enable(RCC\_GPIOD);  
 gpio\_mode\_setup (GPIOD, GPIO\_MODE\_OUTPUT, GPIO\_PUPD\_NONE, GPIO15);  
  
  
 uint16\_t data\_adc = first\_sample();  
 *double* data\_filtered\_double = 0;  
 uint16\_t data\_filtered\_12 = 0;  
  
 *while* (1) {  
  
 *if* (timer\_get\_flag(TIM6, TIM\_SR\_UIF)) {  
  
 gpio\_set(GPIOD, GPIO15);  
   
 data\_adc = adc();  
  
 data\_filtered\_double = IIRFilter(data\_adc);  
 data\_filtered\_12 = round(data\_filtered\_double) + 2047;  
  
 dac(data\_filtered\_12);  
  
 timer\_clear\_flag(TIM6, TIM\_SR\_UIF);  
   
 *while* (! adc\_eoc(ADC1));  
 gpio\_clear(GPIOD, GPIO15);  
 }  
  
 }  
}  
  
  
  
*// ===================== Цифровая фильтрация =====================  
  
double* IIRFilter(*double* x){  
 *double* y = 0; *// выход  
 int* i = 0;  
 y = v[0] + b[0]\*x;  
  
 *// обновляем буфер для следующей итерации  
 for* (; i < N-2; i++)  
 v[i] = v[i+1] + b[i+1]\*x - a[i+1]\*y;  
  
 v[N-2] = b[N-1]\*x - a[N-1]\*y;  
  
 *return* y;  
}  
  
*// ===================== Инициализация ЦАП =====================  
  
void* setup\_dac() {  
  
 *// Настройка порта A4 для выхода цифрового сигнала (осциллограф)* gpio\_mode\_setup (GPIOA, GPIO\_MODE\_ANALOG, GPIO\_PUPD\_NONE, GPIO4);  
  
 *// Тактирование ЦАП* rcc\_periph\_clock\_enable(RCC\_DAC);  
  
 *// Задание канала ЦАП* dac\_enable(CHANNEL\_1);  
}  
  
*// ===================== Инициализация АЦП =====================  
  
void* setup\_adc() {  
  
 *// Настройка порта A0 для входа аналогового сигнала (генератор)* gpio\_mode\_setup (GPIOA, GPIO\_MODE\_ANALOG, GPIO\_PUPD\_NONE, GPIO0);  
  
 *// Тактирование АЦП* rcc\_periph\_clock\_enable(RCC\_ADC1);  
  
 *// Переход в режим одноканального сканирования каналов* adc\_disable\_scan\_mode(ADC1);  
  
 *// Одиночное преобразование* adc\_set\_single\_conversion\_mode(ADC1);  
  
 *// Тактирования блока преобразователя УВХ, примерно 15 тактов (в тактах АЦП)* adc\_set\_sample\_time(ADC1, ADC\_CHANNEL0, ADC\_SMPR\_SMP\_15CYC);  
  
 *// Подаем опорное напряжение на внутренний АЦП* adc\_power\_on(ADC1);   
  
}  
  
*// ===================== Снятие первого отсчета с АЦП, для правильной работы =====================*uint16\_t first\_sample() {  
   
 *// Запуск одиночного преобразования* adc\_start\_conversion\_regular(ADC1);  
   
 *// Блокирующий оператор - появился результат АЦП  
 while* (! adc\_eoc(ADC1));  
  
 *// Записываем результат с АЦП* uint16\_t adc\_retult = adc\_read\_regular(ADC1);  
  
 *// Запуск следующего преобразования* adc\_start\_conversion\_regular(ADC1);   
   
 *return* adc\_retult;  
}  
  
*// ===================== Снятие данных с АЦП =====================*uint16\_t adc() {  
  
 *// Записываем результат с АЦП* uint16\_t temp\_data\_from\_adc = adc\_read\_regular(ADC1);  
   
 *// Запуск следующего преобразования* adc\_start\_conversion\_regular(ADC1);  
  
 *return* temp\_data\_from\_adc;  
}  
  
*// ===================== Передача отфильтрованных данных на ЦАП =====================  
  
void* dac(uint16\_t outputData) {  
  
 dac\_load\_data\_buffer\_single(outputData, RIGHT12, CHANNEL\_1 );  
  
}  
  
*// ===================== Настройка таймера =====================  
  
void* setup\_timer (uint16\_t Fs) {  
   
 rcc\_periph\_clock\_enable(RCC\_TIM6);  
  
 timer\_set\_prescaler(TIM6, 1 - 1);  
  
 timer\_set\_period(TIM6, 5250 - 1);  
  
 timer\_enable\_irq(TIM6, TIM\_DIER\_UIE);  
  
 timer\_enable\_counter(TIM6);  
  
}