

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №41

ОЦЕНКА РЕФЕРАТА
РУКОВОДИТЕЛЬ

доц., канд. тех. наук
должность, уч. степень, звание

подпись, дата

О. О. Жаринов
инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

Разработка цифрового фильтра с фиксированными параметрами

по дисциплине: методы и устройства цифровой обработки сигналов

РЕФЕРАТ ВЫПОЛНИЛk

СТУДЕНТ ГР. _____ 4711

подпись, дата

Хасанов Б. Р,
инициалы, фамилия

Санкт-Петербург 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ:
Заведующий кафедрой №41

Зав. каф., д-р техн. наук, проф _____
должность, уч. степень, звание

подпись, дата

Г.А. Коржавин _____
инициалы, фамилия

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

по дисциплине Методы и устройства цифровой обработки сигналов

выполняемую в осеннем семестре 2020/2021 учебного года

студенту Хасанову Булату Рифкатовичу

Ф.И.О. студента полностью в родительном падеже

группы 4711 института № 4 Вычислительных систем и программирования

1. Тема работы: Разработка цифрового фильтра с фиксированными параметрами

2. Дата выдачи задания «03» сентября 2020 г.

3. Техническое задание на курсовую работу «09» сентября 2020 г.

4. Сроки сдачи студентом оконченной работы «24» декабря 2020 г.

5. Исходная технико-экономическая информация по работе (требования к функциональности, параметры сигналов, требования к элементной базе): $f_n = 390$ Гц, $f_v = 410$ Гц, полосовой фильтр, глубина затухания Z должна быть не менее 60 дБ на частотах 39 Гц и 4100 Гц, микропроцессор 8051, АЦП AD1674, ЦАП AD5310.

6 Состав и объем работы

6.1 Чертежи, схемы, диаграммы: структурная схема устройства, схема компьютерного моделирования

6.2 Программа расчетов на ЭВМ: MathCAD (рекомендовано), Proteus версии не ниже 7.7, Keil uVision, Python 3.7 (при необходимости)

6.3 Расчетно-пояснительная записка не менее 30 стр.

7 Календарный план выполнения курсовой работы*

Этап	Краткое содержание	Срок выполнения*	
		по плану	фактически
1	Уточнение технического задания	09.09.2020	
2	Аналитический обзор по теме курсовой работы	17.09.2020	
3	Определение требований к алгоритму цифровой обработки сигнала. Теоретические сведения	27.09.2020	
4	Моделирование алгоритмов ЦОС и обоснование подходящего для решения поставленной задачи	14.10.2020	
5	Разработка структурной схемы устройства	16.10.2020	
6	Изучение особенностей элементной базы: микропроцессорный вычислитель (обращение к цифровым портам ввода-вывода)	23.10.2020	
7	Разработка схемы и программы в части сопряжения микропроцессора с элементами ввода информационного сигнала (ЦАП)	02.11.2020	
8	Разработка схемы и программы в части сопряжения микропроцессора с элементами ввода информационного сигнала (АЦП)	18.11.2020	
9	Разработка схемы компьютерного моделирования и отладка сквозного тракта передачи сигнала от входа до выхода	23.11.2020	
10	Разработка и отладка программы для микропроцессора в составе устройства	15.12.2020	
11	Компьютерное моделирование разработанного микропроцессорного фильтра	19.12.2020	
12	Оформление пояснительной записки к курсовой работе	22.12.2020	
13	Защита курсовой работы	24.12.2020	


* Преподаватель фиксирует выполнение этапа, с указанием даты его выполнения. Нарушение сроков выполнения этапов является основанием для снижения баллов за курсовую работу:

- 1) при нарушении сроков выполнения двух и более этапов работы максимально возможная оценка за работу - "хорошо";
- 2) при нарушении сроков выполнения пяти и более этапов работы максимально возможная оценка за работу - "удовлетворительно".

8. Дополнительные указания по курсовой работе Допускается разработка устройства на элементной базе, отличающейся от заданной по варианту, в этом случае максимальная оценка за работу – "хорошо"

Задание принято к исполнению:

Студент


дата, подпись

Руководитель

дата, подпись

Хасанов Б. Р.

ФИО

О.О. Жаринов

ФИО

Реферат

Пояснительная записка 65 с., 33 рис., 12 источников, 4 табл.

Ключевые слова: цифровой фильтр, полосовой фильтр, 8051, AD5310, AD1674.

Объектом курсовой работы является цифровой полосовой фильтр, с фиксированными параметрами.

Цель работы: разработать и смоделировать цифровой фильтр с фиксированными параметрами, функционирующего в соответствии с заданными требованиями.

Результат: в среде программирования Keil была разработана программа, смоделирована работа фильтра в среде компьютерного моделирования Proteus.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
ОСНОВНАЯ ЧАСТЬ.....	7
1. Аналитический обзор по теме курсовой работы.....	7
1.1 Цифровой фильтр.....	7
1.2 Классификации цифровых фильтров.....	9
1.3 Проектирование фильтра.....	10
2. Определение требований к алгоритму цифровой обработки сигнала.....	11
3. Моделирование алгоритмов ЦОС и обоснование подходящего для решения поставленной задачи.....	17
4. Разработка структурной схемы устройства.....	23
5. Изучение особенностей элементной базы: микропроцессорный вычислитель.....	24
5.1 Сведения о микроконтроллере.....	24
5.3 Написание и симуляция программы.....	28
6. Разработка схемы и программы в части сопряжения микропроцессора с элементами ввода информационного сигнала (ЦАП).....	30
6.1 Сведения о ЦАП.....	30
6.2 Сведения о передатчике значений.....	33
6.3 Написание программы для работы с ЦАП.....	34
7. Разработка схемы и программы в части сопряжения микропроцессора с элементами ввода информационного сигнала (АЦП).....	36
7.1 Сведения об АЦП.....	36
7.2 Написание программы и моделирование работы АЦП.....	40
8. Разработка схемы компьютерного моделирования и отладка сквозного тракта передачи сигнала от входа до выхода.....	43
ЗАКЛЮЧЕНИЕ.....	46
Список литературы:.....	48
ПРИЛОЖЕНИЕ 1.....	50
ПРИЛОЖЕНИЕ 2.....	54
ПРИЛОЖЕНИЕ 3.....	60
ПРИЛОЖЕНИЕ 4.....	62
ПРИЛОЖЕНИЕ 5.....	64

ВВЕДЕНИЕ

В наши дни, развитие цифровых устройств происходит гигантскими шагами. Очевидно и преимущество применения цифровой обработки сигнала наряду с аналоговым: улучшается помехозащищенность канала связи, бесконечные возможности кодирования информации. Применение микропроцессоров в радиотехнических системах существенно улучшает их массогабаритные, технические и экономические показатели, открывает широкие возможности реализации сложных алгоритмов цифровой обработки сигналов.

Микропроцессоры находят применение при решении широкого круга радиотехнических задач, таких как построение радиотехнических измерителей координат, сглаживающих и экстраполирующих фильтров, устройств вторичной обработки сигналов, специализированных вычислительных устройств бортовых навигационных комплексов, устройств кодирования и декодирования сигналов, весовой обработки пачечных сигналов в радиолокации и в других устройствах. В данном курсовом проекте микропроцессор используется для построения цифрового фильтра.

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.

ОСНОВНАЯ ЧАСТЬ

1. Аналитический обзор по теме курсовой работы.

1.1 Цифровой фильтр

Цифровой фильтр представляет собой устройство обработки сигнала, преобразующее одну последовательность чисел (называемую входной) в другую (называемую выходной). Многие теоретические принципы цифровой фильтрации были известны еще со времен Лапласа. Однако существующий уровень техники не позволял реализовать эти знания, и только появление ЦВМ привело к широкому распространению цифровых фильтров. Сейсмологи успешно применили принципы цифровой фильтрации для решения многих интересных проблем. Использование цифровой фильтрации для обработки фотоснимков, полученных от удаленных источников, межпланетной связи и рентгеновских пленок, позволило значительно улучшить их качество. Она нашла также применение в обработке речи, картографии, радио- и звуколокации и медицинской аппаратуре.[1]

Основными целями фильтрации являются улучшение качества сигнала (например, устранение или снижение помех), извлечение из сигналов информации или разделение нескольких сигналов, объединённых ранее для, например, эффективного использования доступного канала связи.

Упрощённая схема цифрового фильтра реального времени с аналоговым входом и выходом приведена на рисунке 1. Узкополосный аналоговый сигнал периодически выбирается и конвертируется в набор цифровых выборок, цифровой процессор производит фильтрацию, отображая входную последовательность в выходную согласно вычислительному алгоритму фильтра. ЦАП конвертирует отфильтрованный цифровой образ выход в аналоговые значения, которые затем проходят аналоговую фильтрацию для сглаживания и устранения нежелательных высокочастотных компонентов.[2]



Рисунок 1 – Упрощенная схема цифрового фильтра реального времени с аналоговым входом и выходом

Интенсивное развитие цифровой техники не означает, что должны быть полностью отброшены и забыты аналоговые устройства для обработки сигналов. Каждому из типов фильтров, каждому методу обработки сигналов присущи свои преимущества и недостатки и в зависимости от конкретных условий следует применять тот или иной тип фильтра.

Преимущества цифровых фильтров:

- Возможность реализации сложных алгоритмов обработки сигналов, которые неосуществимы с помощью аналоговой техники, например адаптивных алгоритмов, изменяющихся при изменении параметров входного сигнала;
- Точность обработки сигнала цифровыми фильтрами определяется точностью выполняемых расчетов. Она может быть несоизмеримо выше точности обработки сигнала в аналоговых фильтрах;
- Отсутствие источников погрешности вызываемых дрейфом нуля, колебаний температуры, старением, изменением питающих напряжений;
- Отсутствие задачи согласования нагрузок;
- Относительная компактность цифровых фильтров при обработке низкочастотных и инфранизкочастотных сигналов.

Недостатки цифровых фильтров:

- Большая сложность по сравнению с аналоговыми и более высокая стоимость;
- Не очень высокое быстродействие;
- Специфические погрешности, вызванные дискретизацией, квантованием

сигнала и округлением значений обрабатываемого сигнала в процессе вычислений.[3]

1.2 Классификации цифровых фильтров

Фильтры вообще делятся на следующие 4 вида:

1. Фильтры низкой частоты (ФНЧ) - фильтры, пропускающие сигналы с частотой от 0 до f_n (f_n - граничная частота пропускания) и подавляющие все остальные.
2. Фильтры высокой частоты (ФВЧ) - фильтры, пропускающие сигналы с частотой от f_n до бесконечности и подавляющие все остальные.
3. Полосовые фильтры (ПФ) - фильтры, пропускающие сигналы в диапазоне частот от $f_{п1}$ до $f_{п2}$ и подавляющие все остальные.
4. Заграждающие фильтры (ЗФ) - фильтры, подавляющие сигналы в диапазоне частот от $f_{з1}$ до $f_{з2}$ и пропускающие все остальные.

Конкретно цифровые фильтры делятся на два вида: фильтры с конечной импульсной характеристикой (КИХ) и фильтры с бесконечной импульсной характеристикой (БИХ). Как следует из терминологии, эта классификация относится к импульсным характеристикам фильтров. Изменяя веса коэффициентов и число звеньев КИХ-фильтра, можно реализовать практически любую частотную характеристику. КИХ-фильтры могут иметь такие свойства, которые невозможно достичь методами аналоговой фильтрации (в частности, совершенную линейную фазовую характеристику). Но высокоэффективные КИХ-фильтры строятся с большим числом операций умножения с накоплением и поэтому требуют использования быстрых и эффективных процессоров DSP. С другой стороны, БИХ-фильтры имеют тенденцию имитировать принцип действия традиционных аналоговых фильтров с обратной связью. Поэтому их импульсная характеристика имеет бесконечную длительность. Благодаря использованию обратной связи, БИХ-фильтры могут быть реализованы с меньшим количеством коэффициентов, чем КИХ-фильтры просто другим способом. Цифровые фильтры применяются в приложениях адаптивной

фильтрации, благодаря своему быстродействию и простоте изменения характеристик воздействием на его коэффициенты.

Нерекурсивные фильтры по сравнению с рекурсивными обладают следующими достоинствами:

1. Они всегда устойчивы при любых коэффициентах фильтра.
2. Для НФ проще вычисление коэффициентов.
3. На их основе возможно получение фильтров с передаточной характеристикой, аргумент которой линейно зависит от частоты, т.е. со строго линейной фазовой характеристикой.
4. Мощность собственных шумов НФ, как правило, гораздо меньше, чем у РФ. Она равна нулю, т. е. у НФ отсутствуют собственные шумы в том случае, если операции сложения и умножения выполняются точно.
5. В системах с изменением частоты дискретизации применение НФ сокращает необходимое число арифметических операций.

Недостатки нерекурсивных фильтров по отношению к рекурсивным будут следующие:

1. Менее резкий спад модуля передаточной характеристики.
2. При одинаковых требованиях к АЧХ, отсутствии требований к линейности ФЧХ и постоянной частоте дискретизации они требуют выполнения существенно большего числа операций.
3. Схемная реализация их оказывается намного сложнее.

С учетом этих свойств выбирают фильтр того, или иного типа. Так, если требуется фильтр со строго линейной фазовой характеристикой, то используют нерекурсивный фильтр. Если же необходима большая крутизна спада модуля передаточной характеристики, то применяют рекурсивный фильтр.[4]

1.3 Проектирование фильтра

Разработка цифрового фильтра проходит в пять этапов.

1. Спецификация требований к фильтру.
2. Вычисление подходящих коэффициентов фильтра.

3. Представление фильтра подходящей структурой.
4. Анализ влияния конечной разрядности на производительность фильтра.
5. Реализация фильтра на программном и/или аппаратном уровне.

Названные пять этапов не всегда независимы; кроме того, они не всегда располагаются в указанном порядке. Фактически существуют методы, которые позволяют объединить второй этап и некоторые аспекты третьего и четвертого. Чтобы получить эффективный фильтр, иногда приходится проводить данный процесс в несколько итераций, особенно, если спецификации не являются совершенно определёнными (как обычно и бывает), или же разработчик желает исследовать альтернативные структуры.[5]

2. Определение требований к алгоритму цифровой обработки сигнала.

В работе будет использован БИХ фильтр. Причина выбора именно этого типа в том, что на фильтрацию входного сигнала уходит меньше процессорного времени, по сравнению с КИХ фильтром.

Общая идея всех методов расчёта рекурсивных фильтров состоит в том, чтобы, используя передаточную функцию аналогового фильтра-прототипа $W(p)$, получить дискретную передаточную функцию ЦФ $D(z)$. При этом заранее нужно выбрать значение периода дискретизации T_d , исходя из требований теоремы Котельникова, и, кроме того, желательно, чтобы верхняя граничная частота полосы пропускания фильтра была бы как минимум в несколько раз (лучше – на порядок) меньше частоты Найквиста.

Переход от $W(p)$ к $D(z)$ рекурсивного ЦФ можно осуществить с использованием любого из трех основных методов:

- 1) метод инвариантности переходной характеристики;
- 2) метод на основе формул дискретного интегрирования;
- 3) метод согласованного Z-преобразования.

Наибольшее распространение на практике получил метод на основе формул дискретного интегрирования, который использует связь между переменными p и z . [6] Именно таким мы и собираемся воспользоваться, а

конкретнее – методом обобщённого билинейного преобразования. Он заключается в замене p в передаточной функции нормированного аналогового ФНЧ прототипа формулой с z оператором, причём для каждого типа фильтра формала будет отличаться. Формулы замены переменной в таблице 1

Таблица 1 – Формулы замены переменной p , применяемые при преобразовании передаточной функции нормированного аналогового ФНЧ-прототипа в передаточную функцию цифрового фильтра заданного типа методом обобщенного билинейного преобразования.

Тип	Параметры результирующего ЦФ	Формула для замены переменной*
ФНЧ	Частота среза ω_{cp}	$p \rightarrow \frac{2}{\omega_{cp} T_{\Delta}} \frac{z-1}{z+1}$
ФВЧ	Частота среза ω_{cp}	$p \rightarrow \frac{\omega_{cp} T_{\Delta}}{2} \frac{z+1}{z-1}$
ПФ	Граничные частоты полосы пропускания ω_n и ω_b	$p \rightarrow \gamma \frac{z^2 - 2\zeta z + 1}{z^2 - 1}$
РФ	Граничные частоты полосы пропускания ω_n и ω_b	$p \rightarrow \frac{z^2 - 1}{\gamma(z^2 - 2\zeta z + 1)}$

* Примечание $\gamma = \operatorname{ctg} \left(\frac{T_{\Delta}}{2} (\omega_b - \omega_n) \right)$, $\zeta = \frac{\cos \left(\frac{T_{\Delta}}{2} (\omega_b + \omega_n) \right)}{\cos \left(\frac{T_{\Delta}}{2} (\omega_b - \omega_n) \right)}$.

Помимо прямой замены есть способ посчитать и АЧХ фильтра и его коэффициенты по всё тому же методу билинейного преобразования. В таблице 2 содержатся формулы, которые позволят после расчёта осуществить фильтрацию по формуле 1

$$y_n = \sum_{k=0}^q b_k x_{n-k} - \sum_{m=1}^r a_m y_{n-m} \quad (1)$$

Таблица 2 – Формулы сомножителей $D_k(z)$, получающиеся при применении формул обобщённого билинейного преобразования к выражению (2)

Тип ЦФ	Формула для $D_k(z)$	Формулы* для коэффициентов $D_k(z)$
ФНЧ с частотой среза ω_{cp}	$\frac{\beta_0 z^2 + \beta_1 z + \beta_2}{\alpha_0 z^2 + \alpha_1 z + \alpha_2}$	$\beta_0 = (\omega_{cp} T_\Delta)^2, \beta_1 = 2\beta_0, \beta_2 = \beta_0,$ $\alpha_0 = a_2 (\omega_{cp} T_\Delta)^2 + 2a_1 \omega_{cp} T_\Delta + 4a_0,$ $\alpha_1 = 2a_2 (\omega_{cp} T_\Delta)^2 - 8a_0,$ $\alpha_2 = a_2 (\omega_{cp} T_\Delta)^2 - 2a_1 \omega_{cp} T_\Delta + 4a_0$
ФВЧ с частотой среза ω_{cp}	$\frac{\beta_0 z^2 + \beta_1 z + \beta_2}{\alpha_0 z^2 + \alpha_1 z + \alpha_2}$	$\beta_0 = 4, \beta_1 = -8, \beta_2 = 4,$ $\alpha_0 = a_0 (\omega_{cp} T_\Delta)^2 + 2a_1 \omega_{cp} T_\Delta + 4a_2,$ $\alpha_1 = 2a_0 (\omega_{cp} T_\Delta)^2 - 8a_2,$ $\alpha_2 = a_0 (\omega_{cp} T_\Delta)^2 - 2a_1 \omega_{cp} T_\Delta + 4a_2$
ПФ с частотами среза ω_n и ω_b	$\frac{\beta_0 z^4 + \beta_1 z^3 + \beta_2 z^2 + \beta_3 z + \beta_4}{\alpha_0 z^4 + \alpha_1 z^3 + \alpha_2 z^2 + \alpha_3 z + \alpha_4}$	$\beta_0 = 1, \beta_1 = 0, \beta_2 = -2, \beta_3 = 0, \beta_4 = 1,$ $\alpha_0 = \gamma^2 a_0 + \gamma a_1 + a_2,$ $\alpha_1 = -4\gamma^2 \zeta a_0 - 2\gamma \zeta a_1,$ $\alpha_2 = 4\gamma^2 \zeta^2 a_0 + 2\gamma a_0 - 2a_2,$ $\alpha_3 = -4\gamma^2 \zeta a_0 + 2\gamma \zeta a_1,$ $\alpha_4 = \gamma^2 a_2 - \gamma a_1 + a_0$
РФ с частотами среза ω_n и ω_b	$\frac{\beta_0 z^4 + \beta_1 z^3 + \beta_2 z^2 + \beta_3 z + \beta_4}{\alpha_0 z^4 + \alpha_1 z^3 + \alpha_2 z^2 + \alpha_3 z + \alpha_4}$	$\beta_0 = \gamma^2, \beta_1 = -4\gamma^2 \zeta, \beta_2 = 4\gamma^2 \zeta^2 + 2\gamma^2,$ $\beta_3 = \beta_1, \beta_4 = \beta_0,$ $\alpha_0 = \gamma^2 a_2 - \gamma a_1 + a_2,$ $\alpha_1 = 4\gamma^2 \zeta a_2 - 2\gamma \zeta a_1,$ $\alpha_2 = 4\gamma^2 \zeta^2 a_2 + 2\gamma a_2 - 2a_0,$ $\alpha_3 = -4\gamma^2 \zeta a_2 + 2\gamma \zeta a_1,$ $\alpha_4 = \gamma^2 a_2 - \gamma a_1 + a_0$

Для того, чтобы посчитать коэффициенты выше понадобятся а-

коэффициенты. Они получаются из коэффициентов передаточной функции нормированного ФНЧ прототипа. Можно выделить несколько видов фильтров:

- Бесселя
- Чебышева
- Баттерворта

У них есть характеристические отличия, но в данном случае нас интересует только то, что фильтр Баттерворта наиболее просто рассчитывается. Его передаточная функция нормированного ФНЧ-прототипа показана в формуле (2)

$$W_{Bt}(p) = \begin{cases} \frac{1}{\prod_{k=1}^{\frac{n}{2}} p^2 - 2p \cos\left(\frac{2k+n-1}{2n}\pi\right) + 1}, & \text{для чётных } n \\ \frac{1}{\prod_{k=1}^{\frac{n}{2}} \left(p^2 - 2p \cos\left(\frac{2k+n-1}{2n}\pi\right) + 1\right) (p+1)}, & \text{для нечётных } n \end{cases} \quad (2)$$

где n – порядок фильтра

Выдвинем гипотезу, что 2-ой порядок данного типа фильтра соответствует ТЗ. Для проверки этой гипотезы смоделируем фильтр. Программа для моделирования написана на python 3.7 и её полная версия написана в приложении 1. Прокомментирую наиболее важные части – функцию `recursive_transfer_function` и `generate_recursive_filter`. Сначала рассмотрим первую. В ней непосредственно производится обобщённое билинейное преобразование.

На вход функция принимает значение z оператора и нижние и верхние частоты среза в герцах. Первое, что в ней делается – перевод частот в радианы:

```
fn = fn * 2 * np.pi
fv = fv * 2 * np.pi
```

Затем вводятся константы записанные в примечании таблицы 1

```
gamma = 1/np.tan((delta_T/2) * (fv - fn))
zeta = np.cos((delta_T/2) * (fv + fn))/np.cos((delta_T/2) * (fv - fn))
```

После чего выписывается формула замены переменной из таблицы 1 соответствующая ПФ:

$$\text{changed_p} = \text{gamma} * (((z^{**2}) - 2 * \text{zeta} * z + 1)/((z^{**2}) - 1))$$

Ну и наконец передаём это всё в передаточную функцию Баттерворта 2-ого порядка полученной из формулы (2):

$$\text{calculated_transfer_function} = 1/((\text{changed_p}^{**2}) + 1.41421 * \text{changed_p} + 1)$$

Результат этой формулы возвращаем из функции.

Соответственно, чтобы получить коэффициент подавления на конкретной частоте нужно передавать на место z оператора результат формулы $e^{2\pi f T_{\Delta j}}$, где f – частота для которой проводится расчёт, и возвести получившееся комплексное число в модуль. Чтобы отобразить результат в дБ нужно взять от получившегося числа логарифм по основанию 10 и умножить на 20. В результате получается график АЧХ фильтра, показанный на рисунке 2

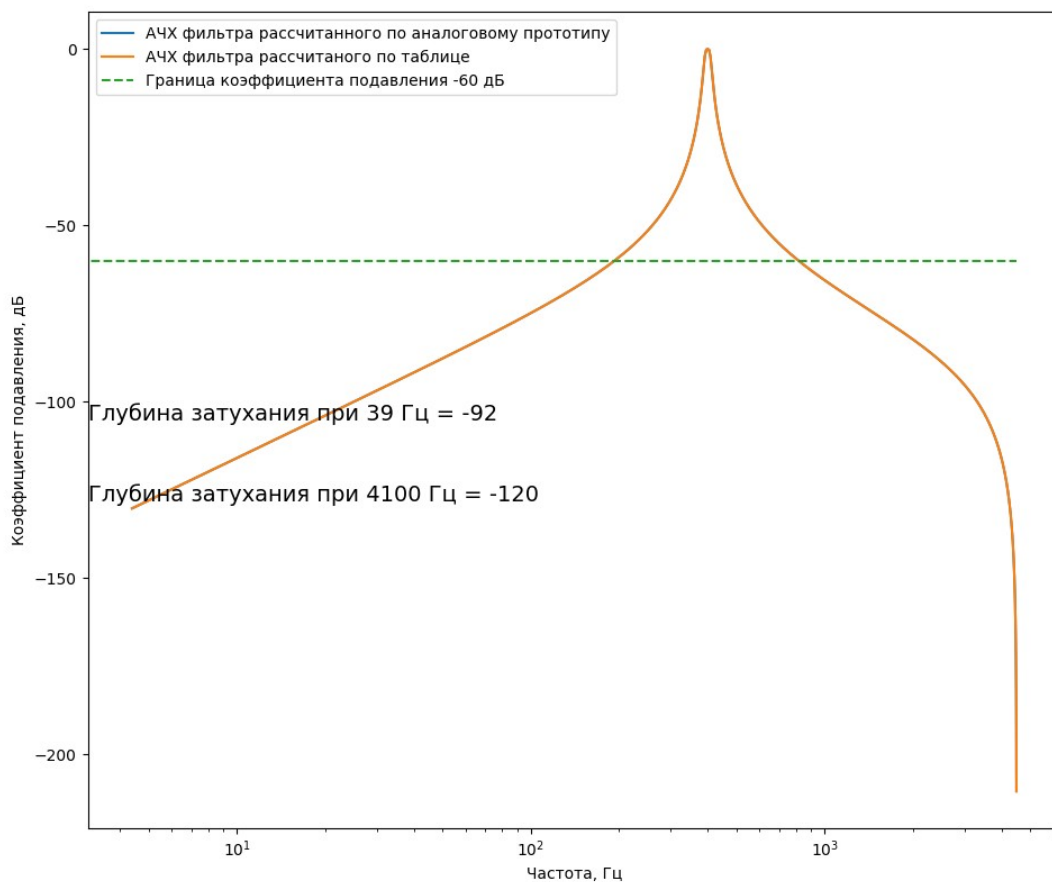


Рисунок 2 – АЧХ фильтра

Далее, рассмотрим функцию `generate_recursive_filter`. Эта функция считает коэффициенты уже по таблице, без непосредственной подстановки. На вход функция принимает значение z оператора и нижние и верхние частоты среза в герцах. Снова осуществляем перевод частот в радианы:

```
fn = fn * 2 * np.pi
fv = fv * 2 * np.pi
```

Затем вводятся константы записанные в примечании таблицы 1

```
gamma = 1/np.tan((delta_T/2) * (fv - fn))
zeta = np.cos((delta_T/2) * (fv + fn))/np.cos((delta_T/2) * (fv - fn))
```

Далее определяются те самые a -коэффициенты

```
a_constants = [1, -2 * np.cos(((2 * order_number + filter_order - 1)/(2 * filter_order)) * np.pi), 1]
```

После чего считаются коэффициенты α и β по таблице 2

```
current_beta = [1, 0, -2, 0, 1]
```

```
current_alpha = [gamma**2 * a_constants[0] + gamma * a_constants[1] +
a_constants[2],
-4 * gamma**2 * zeta * a_constants[0] - 2 * gamma * zeta * a_constants[1],
4 * gamma**2 * zeta**2 * a_constants[0] + 2 * gamma**2 * a_constants[0]
- 2 * a_constants[2],
-4 * gamma**2 * zeta * a_constants[0] + 2 * gamma * zeta * a_constants[1],
gamma**2 * a_constants[0] - gamma * a_constants[1] + a_constants[2]]
```

Теперь считаем $D_k(z)$ с полученными коэффициентами

```
current_beta = [1, 0, -2, 0, 1]
```

```
current_alpha = [gamma**2 * a_constants[0] + gamma * a_constants[1] +
a_constants[2],
-4 * gamma**2 * zeta * a_constants[0] - 2 * gamma * zeta * a_constants[1],
4 * gamma**2 * zeta**2 * a_constants[0] + 2 * gamma**2 * a_constants[0]
- 2 * a_constants[2],
-4 * gamma**2 * zeta * a_constants[0] + 2 * gamma * zeta * a_constants[1],
gamma**2 * a_constants[0] - gamma * a_constants[1] + a_constants[2]]
```

Далее высчитываем дискретную передаточную функцию. С помощью получившихся коэффициентов

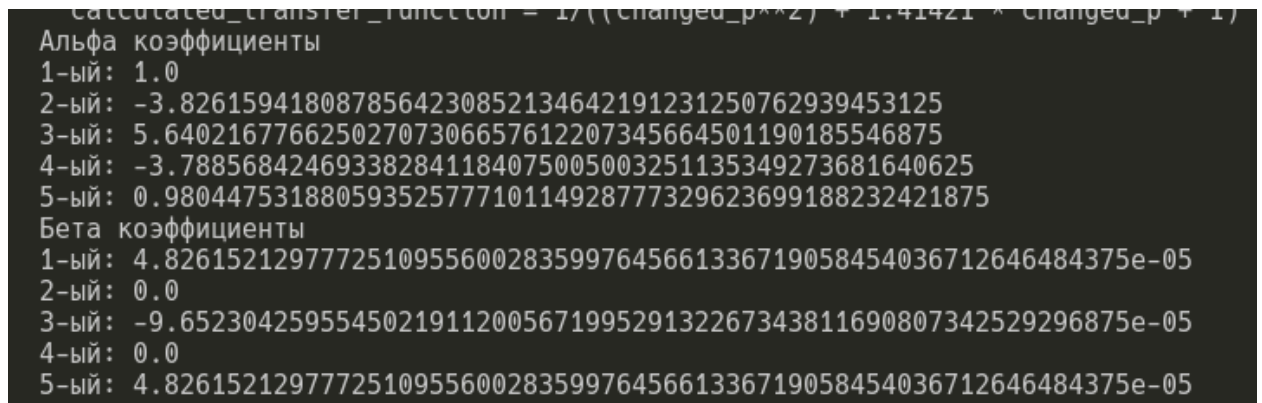
```
our_filter = our_filter * ((current_beta[0] * z**4 + current_beta[1] * z**3 +
```



```
current_betta[2] * z**2 + current_betta[3] * z + current_betta[4])/ \
(current_alpha[0] * z**4 + current_alpha[1] * z**3 + current_alpha[2] * z**2 +
current_alpha[3] * z + current_alpha[4]))
```

Наконец вычисляем коэффициенты α и β для фильтрации. Результат вычисления последних можно увидеть на рисунке 3

```
alphas = np.array(current_alpha)/current_alpha[0]
betas = np.array(current_betta)/current_alpha[0]
```



```
calculated_transfer_function = 1/((changed_p**2) + 1.41421 * changed_p + 1)
Альфа коэффициенты
1-ый: 1.0
2-ый: -3.8261594180878564230852134642191231250762939453125
3-ый: 5.64021677662502707306657612207345664501190185546875
4-ый: -3.788568424693382841184075005003251135349273681640625
5-ый: 0.98044753188059352577710114928777329623699188232421875
Бета коэффициенты
1-ый: 4.82615212977725109556002835997645661336719058454036712646484375e-05
2-ый: 0.0
3-ый: -9.6523042595545021911200567199529132267343811690807342529296875e-05
4-ый: 0.0
5-ый: 4.82615212977725109556002835997645661336719058454036712646484375e-05
```

Рисунок 3 – Коэффициенты фильтра

И без вычислений коэффициента затухания на частотах 39 Гц и 4100 Гц рисунка 1 заметно, что они ниже заданных по заданию -60 дБ, более точные коэффициенты затухания написаны на рисунке. Следовательно, гипотеза подтверждена и фильтр 2-ого порядка нам подходит.

Из рассчитанных коэффициентов α и β , продемонстрированных на рис 2 можно составить разностное уравнение, округляя до количества чисел после запятой, чуть большей минимума (12 знаков), ради простоты записи

$$y_k = 4.82615212978 \cdot 10^{-5} x_k - 9.65230425955 \cdot 10^{-5} x_{k-2} + 4.82615212978 \cdot 10^{-5} x_{k-4} - \\ + 3.82615941809 y_{k-1} - 5.64021677663 y_{k-2} + 3.78856842469 y_{k-3} - 0.98044753188 y_{k-4}$$

3. Моделирование алгоритмов ЦОС и обоснование подходящего для решения поставленной задачи

Для проверки работоспособности выбранного цифрового фильтра нужно промоделировать его работу. Фильтрация производится во временной области по уравнению (3)

$$y_n = \sum_{k=0}^q b_k x_{n-k} - \sum_{m=1}^r a_m y_{n-m} \quad (3)$$

где массивы b и a вычисляются по формулам из таблицы 3

Таблица 3 – Формулы сомножителей $D_k(z)$, получающиеся при применении формул обобщённого билинейного преобразования к выражению (1)

Тип ЦФ	Формула для $D_k(z)$	Формулы* для коэффициентов $D_k(z)$
ФНЧ с частотой среза ω_{cp}	$\frac{\beta_0 z^2 + \beta_1 z + \beta_2}{\alpha_0 z^2 + \alpha_1 z + \alpha_2}$	$\beta_0 = (\omega_{cp} T_{\Delta})^2, \beta_1 = 2\beta_0, \beta_2 = \beta_0,$ $\alpha_0 = a_2 (\omega_{cp} T_{\Delta})^2 + 2a_1 \omega_{cp} T_{\Delta} + 4a_0,$ $\alpha_1 = 2a_2 (\omega_{cp} T_{\Delta})^2 - 8a_0,$ $\alpha_2 = a_2 (\omega_{cp} T_{\Delta})^2 - 2a_1 \omega_{cp} T_{\Delta} + 4a_0$
ФВЧ с частотой среза ω_{cp}	$\frac{\beta_0 z^2 + \beta_1 z + \beta_2}{\alpha_0 z^2 + \alpha_1 z + \alpha_2}$	$\beta_0 = 4, \beta_1 = -8, \beta_2 = 4,$ $\alpha_0 = a_0 (\omega_{cp} T_{\Delta})^2 + 2a_1 \omega_{cp} T_{\Delta} + 4a_2,$ $\alpha_1 = 2a_0 (\omega_{cp} T_{\Delta})^2 - 8a_2,$ $\alpha_2 = a_0 (\omega_{cp} T_{\Delta})^2 - 2a_1 \omega_{cp} T_{\Delta} + 4a_2$
ПФ с частотами среза ω_n и ω_b	$\frac{\beta_0 z^4 + \beta_1 z^3 + \beta_2 z^2 + \beta_3 z + \beta_4}{\alpha_0 z^4 + \alpha_1 z^3 + \alpha_2 z^2 + \alpha_3 z + \alpha_4}$	$\beta_0 = 1, \beta_1 = 0, \beta_2 = -2, \beta_3 = 0, \beta_4 = 1,$ $\alpha_0 = \gamma^2 a_0 + \gamma a_1 + a_2,$ $\alpha_1 = -4\gamma^2 \zeta a_0 - 2\gamma \zeta a_1,$ $\alpha_2 = 4\gamma^2 \zeta^2 a_0 + 2\gamma a_0 - 2a_2,$ $\alpha_3 = -4\gamma^2 \zeta a_0 + 2\gamma \zeta a_1,$ $\alpha_4 = \gamma^2 a_2 - \gamma a_1 + a_0$
РФ с частотами среза ω_n и ω_b	$\frac{\beta_0 z^4 + \beta_1 z^3 + \beta_2 z^2 + \beta_3 z + \beta_4}{\alpha_0 z^4 + \alpha_1 z^3 + \alpha_2 z^2 + \alpha_3 z + \alpha_4}$	$\beta_0 = \gamma^2, \beta_1 = -4\gamma^2 \zeta, \beta_2 = 4\gamma^2 \zeta^2 + 2\gamma^2,$ $\beta_3 = \beta_1, \beta_4 = \beta_0,$ $\alpha_0 = \gamma^2 a_2 - \gamma a_1 + a_2,$ $\alpha_1 = 4\gamma^2 \zeta a_2 - 2\gamma \zeta a_1,$ $\alpha_2 = 4\gamma^2 \zeta^2 a_2 + 2\gamma a_2 - 2a_0,$ $\alpha_3 = -4\gamma^2 \zeta a_2 + 2\gamma \zeta a_1,$ $\alpha_4 = \gamma^2 a_2 - \gamma a_1 + a_0$

* Примечание: $\gamma = \operatorname{ctg} \left(\frac{T_{\Delta}}{2} (\omega_b - \omega_n) \right), \zeta = \frac{\cos \left(\frac{T_{\Delta}}{2} (\omega_b + \omega_n) \right)}{\cos \left(\frac{T_{\Delta}}{2} (\omega_b - \omega_n) \right)}$

Программа для моделирования написана на языке python 3.7 её полное содержание можно найти в приложении 2, я прокомментирую самые важные части. В данном блоке кода задаются характеристики двух сигналов. Один

находиться в середине полосы пропускания и должен пройти через фильтр без изменений, а другой подавиться, т. к. находится вне полосы пропускания фильтра Дискретизация сигнала определена в начале файла и равна 1/9000

$$A1 = 3$$

$$A2 = 2$$

$$f1 = 200$$

$$f2 = 400.33$$

С помощью функции `signal_generate` генерируем сигналы с заданными характеристиками. Это делается для того, чтобы подтвердить, что подавление сигналов соответствует АЧХ. Формула генерации следующая $s = A \cdot \sin(2\pi f t)$, где s – сигнал, A – амплитуда гармоники, f – частота гармоники, t – время. Получившийся сигналы показаны на рисунке 4 и 5, обозначены как нефигьтрованные.

```
signal_in_bandwith = signal_generate(signal_discritisation, {A2:f2})
```

```
signal_out_of_bandwith = signal_generate(signal_discritisation, {A1:f1})
```

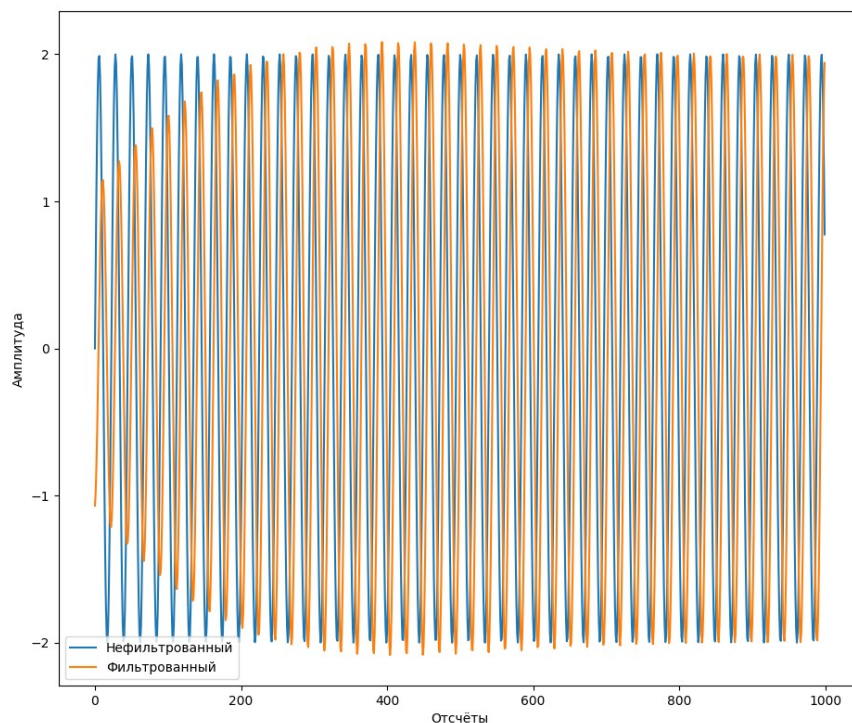


Рисунок 4 – Сравнение фильтрованного и нефигьтрованного синусоидального сигнала с амплитудой 2 и частотой 400

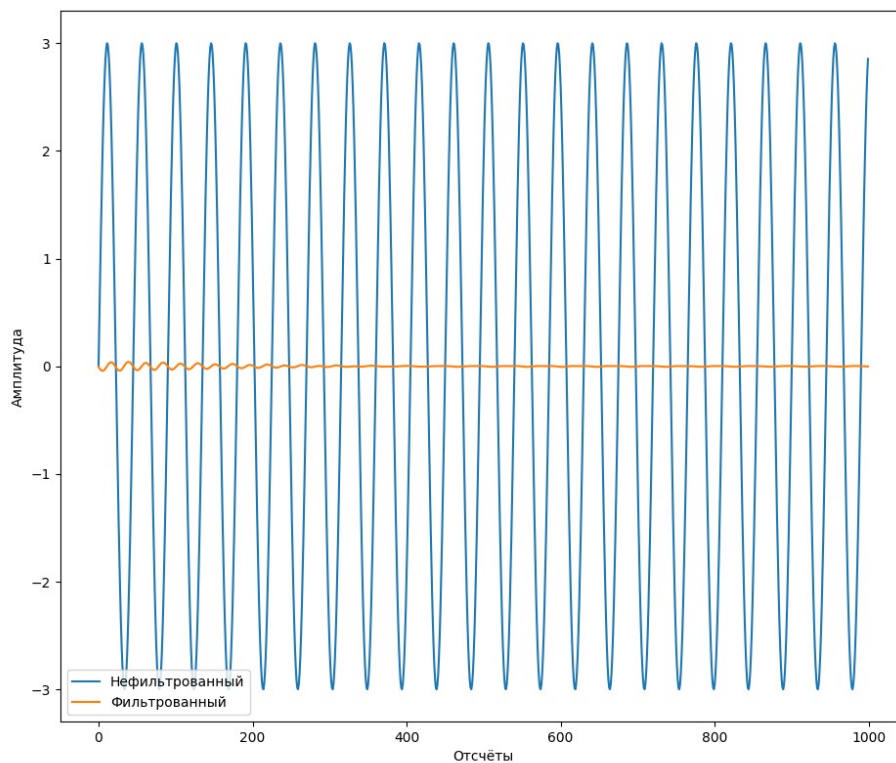


Рисунок 5 – Сравнение фильтрованного и нефильтрованного синусоидального сигнала с амплитудой 3 и частотой 200

Далее с помощью следующей строки добавим к сигналу, находящемуся в полосе пропускания фильтра белый шум с мат. ожиданием 0 и дисперсией 1, чтобы проверить как фильтр справляется непосредственно с задачей фильтрации. Результат на рисунке 6, подписан как нефильтрованный.

```
gaussian_noise_signal = signal_in_bandwidth + lambd * np.random.normal(0, 1, N)
```

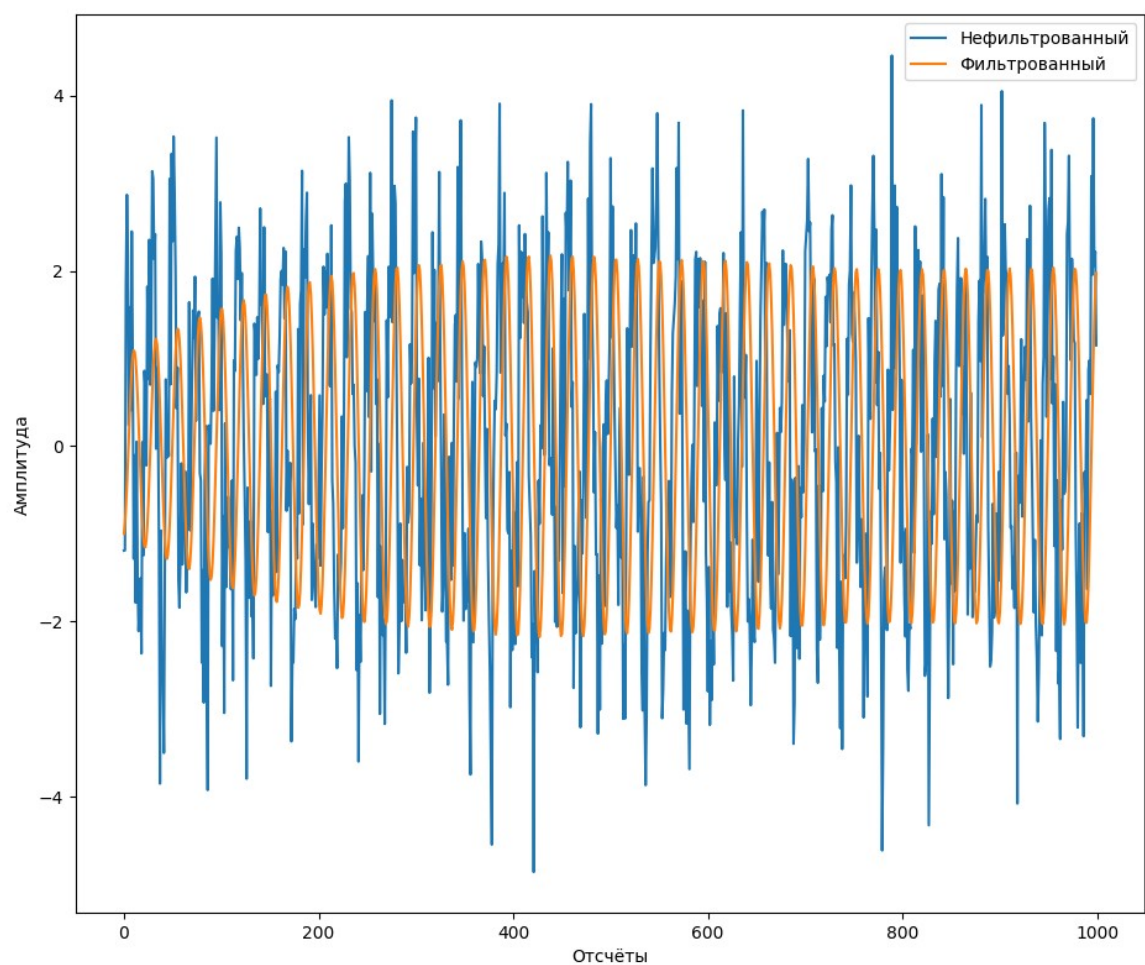


Рисунок 6 – Сравнение фильтрованного и нефильтрованного синусоидального сигнала с амплитудой 2 и частотой 400 и белым шумом

Генерируем АЧХ теоретического рекурсивного фильтра, как в прошлом этапе. Оно показано на рисунке 7.

filter_transfer_function =

recursive_transfer_function(np.exp(np.array(coefficient)), fv=410, fn=390)

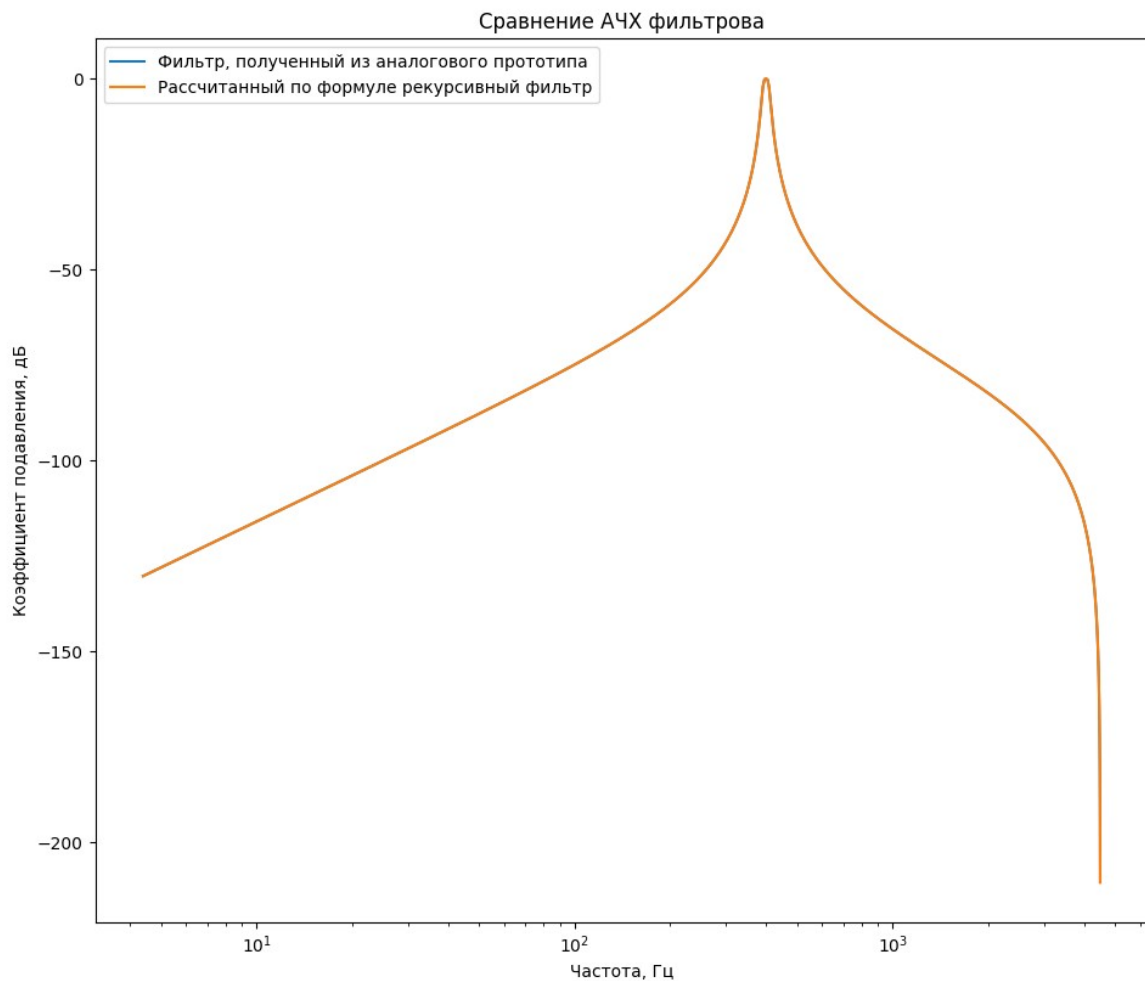


Рисунок 7 – Сравнение двух АЧХ фильтров с одинаковыми параметрами

Затем производится расчёт фильтра для его последующего использования по формулам приведённым в таблице 3, для ПФ. Это делается с помощью функции `generate_recursive_filter`. Возвращаются как АЧХ, показанное на рисунке 3, так и коэффициенты a и b , используемые при фильтрации.

```
recursive_filter, A, B = generate_recursive_filter(np.exp(np.array(coefficient)), fv
= 410, fn=390)
```

После чего производится фильтрация сигнала находящегося в полосе пропускания. Она производится по формуле 3, которая реализована в функции `signal_filtration_with_recursive_filter`. Результат показан на рисунке 4, обозначен как **фильтрованный**. Фильтрованный сигнал смещён влево относительно нефильтрованного на 220 точек, чтобы их удобнее было сравнивать.

filtered_usuall_signal =

signal_filtration_with_recursive_filter(signal_in_bandwith, A, B)

Теперь фильтруем сигнал вне полосы пропускания. Результат обозначен как фильтрованный на рисунке 2.

filtered_harmonic_noise_signal =

signal_filtration_with_recursive_filter(signal_out_of_bandwith, A, B)

Ну и наконец фильтруем сигнал с белым шумом. Результат на рисунке 3, обозначен как фильрованный.

filtered_gaussian_noise_signal =

signal_filtration_with_recursive_filter(gaussian_noise_signal, A, B)

На рисунке 4 видно, что сигнал с частотой входящей в спектр частот в полосе пропускания фильтра минует его без изменений, тогда как сигнал с частотой выходящей за рамки полосы полностью подавляется это видно на 5 рисунке. На 6 рисунке видно, что фильтр так же хорошо справляется с шумами. Из наблюдений можно сделать вывод, что фильтр работает в соответствии с АЧХ

4. Разработка структурной схемы устройства

Абстрактная структурная схема цифрового фильтра показана на рисунке 8

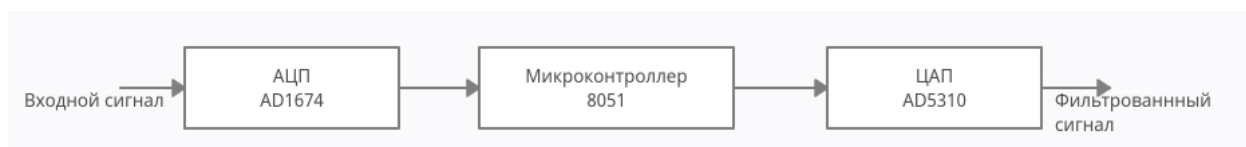


Рисунок 8 – Абстрактная схема цифрового фильтра

Общий принцип заключается в следующих этапах:

1. Входной сигнал поступает на АЦП AD1674, в нём он преобразуется в цифровой и передаётся на выход АЦП
2. Оцифрованный сигнал попадает на вход микроконтроллера 8051, осуществляется фильтрация сигнала программным путём с помощью

фильтра с заданными характеристиками и параметрами. После чего фильтрованный цифровой сигнал подаётся на выход микроконтроллера.

3. ЦАП AD5310 принимает на вход цифровой фильтрованный сигнал и преобразует его в аналоговый, а затем передаёт его на выход.

5. Изучение особенностей элементной базы: микропроцессорный вычислитель

5.1 Сведения о микроконтроллере

AT89C51RC2 – высокопроизводительная КМОП флэш-версия микроконтроллера 8-разрядного однокристального микроконтроллера 80C51. Он содержит 32 кБ флэш-памяти для хранения программы и данных. Он был выбран потому, что имеет наибольшую скорость работы среди всех 8051 микроконтроллеров моделируемых в среде proteus.

Флэш-память размером 32кБ может программироваться как в параллельном режиме, так и в последовательном с внутрисхемным программированием (ISP) или программным. Напряжение программирования генерируется внутри от стандартного источника питания на выводе Vcc.

AT89C51RC2 сохраняет все особенности Atmel 80C52 с 2048 байтами ОЗУ, 10 источниками прерываний и тремя таймерами-счетчиками.

Кроме того, AT89C51RC2 имеет программируемый счетный массив (PCA), расширенное ОЗУ (XRAM) до 1792 байт, аппаратный сторожевой таймер, SPI-интерфейс, клавиатуру, универсальный последовательный канал, облегчающий микропроцессорную связь (EUSART) и механизм изменения быстродействия (режим X2).

Полностью статическая разработка AT89C51RC2 позволяет уменьшать потребляемую мощность за счет уменьшения системной частоты до любого значения, в т.ч. до постоянного тока, при этом данные не теряются.

AT89C51RC2 имеет два программно-настраиваемых режима, уменьшающих энергопотребление и 8-разрядный предделитель для дальнейшего уменьшения потребления. В режиме холостого хода ЦПУ останавливается, при этом периферийные устройства и система прерываний функционируют дальше. В энергосберегающем режиме сохраняется информация в ОЗУ, в то время как все остальные функции не активны.

Отличительные особенности:

- Совместимость с инструкциями 8051

- Шесть 8-разрядных порта ввода-вывода (64-выводная или 68-выводная версии)
- Четыре 8-разрядных порта ввода-вывода (44 –выводная версия)
- Три 16-разрядных таймера-счетчика
- 256 байт сверхоперативной памяти
- 9 источников запроса на прерывание с 4 уровнями приоритета
- Интегрированный контроль питания (POR/PFD) для контроля внутреннего питания
- Внутрисхемное программирование ISP использует стандартное питание Vcc
- Загрузочное ПЗУ содержит процедуры низкого уровня для программирования флэш-памяти и исходный последовательный загрузчик
- Высокопроизводительная архитектура
 - 40 МГц в стандартном режиме
 - 30 МГц режиме X2 (6 тактов в машинном цикле)
- 64 кБ встроенной флэш-памяти программ/данных
 - Побайтная и постраничная (128 байт) очистка и запись
 - 100000 циклов записи
- Встроенное расширенное ОЗУ емкостью 1792 байт (XRAM)
 - Программно выбираемый размер (0, 256, 512, 768, 1024, 1792 байт)
 - 768 байт выбирается при сбросе для совместимости с T89C51RD2
- Встроенные 2048 байт ЭППЗУ для хранения данных (только у AT89C51ED2)
- 100000 циклов записи
- Двойной указатель данных
- Переменная длина инструкции MOVX для доступа к медленному ОЗУ и периферийным устройствам

- Улучшенный режим X2 с независимыми настройками ЦПУ и каждого периферийного устройства
- Клавиатурный интерфейс на порте 1 с функциям прерывания
- SPI-интерфейс (режим ведущий/подчиненный)
- 8-разрядный предделитель тактовых импульсов
- 16-разрядный программируемый счетный массив
 - Быстродействующий выход
 - Функции сравнения и захвата фронтов
 - Широтно-импульсная модуляция
 - Совместимый сторожевой таймер
- Вывод асинхронного сброса
- Полнодуплексный улучшенный УАПП с встроенным генератором скорости передачи
- Малые электромагнитные излучения (запрещен ALE)
- Аппаратный сторожевой таймер (однократно разрешается после сброса), флаг выключения питания
- Режимы управления энергопотреблением: режим холостого хода (Idle), экономичный (Power-down) режим
- Диапазон напряжения питания: 2.7В...5.5В
- Промышленный температурный диапазон(-40 ...+85°C)[7]

На рисунке 9 можно увидеть блок-схема микроконтроллера, а на рисунке 10 расположение выводов микроконтроллера

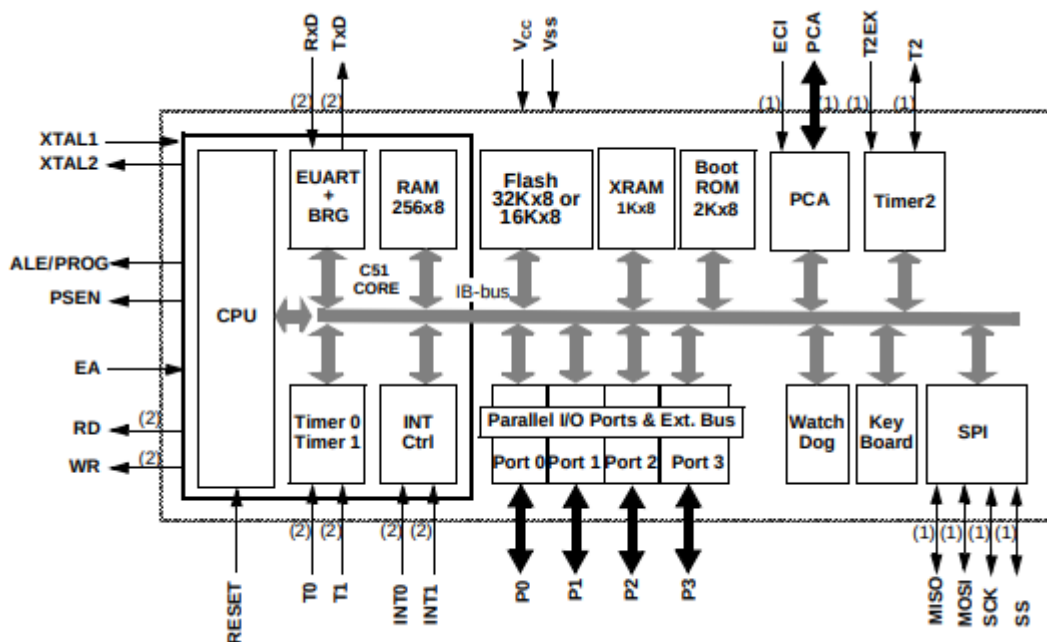


Рисунок 9 – Блок-схема микроконтроллера

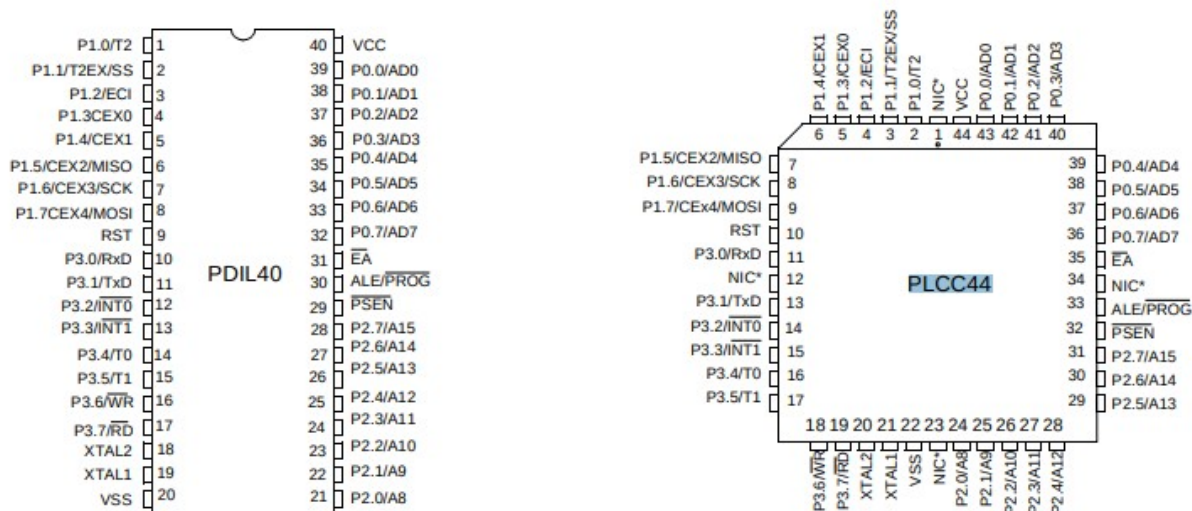


Рисунок 10 – Схема расположения выводов

Чтобы показать, что навыки для работы с микроконтроллером имеются – для данного микроконтроллера будет написана и смоделирована простейшая программа, мигания светодиодом. Для написания и компиляции программы будет использован Keil, для симуляции – Proteus. Рассмотрим эти среды

Keil uVision – Среда разработки, представляющая собой набор утилит для выполнения полного комплекса мероприятий по написанию программного обеспечения для микроконтроллеров.

Keil uVision позволяет работать с проектами любой степени сложности, начиная с введения и правки исходных текстов и заканчивая внутрисхемной отладкой кода и программированием ПЗУ микроконтроллера. От разработчика скрыта большая часть второстепенных функций, что сильно разгружает интерфейс и делает управление интуитивно понятным. Однако при возрастании сложности реализуемых задач, всегда можно задействовать весь потенциал модулей, функционирующих под управлением единой оболочки.[8]

Proteus – Мощнейшая система автоматизированного проектирования, позволяющая виртуально смоделировать работу огромного количества аналоговых и цифровых устройств.

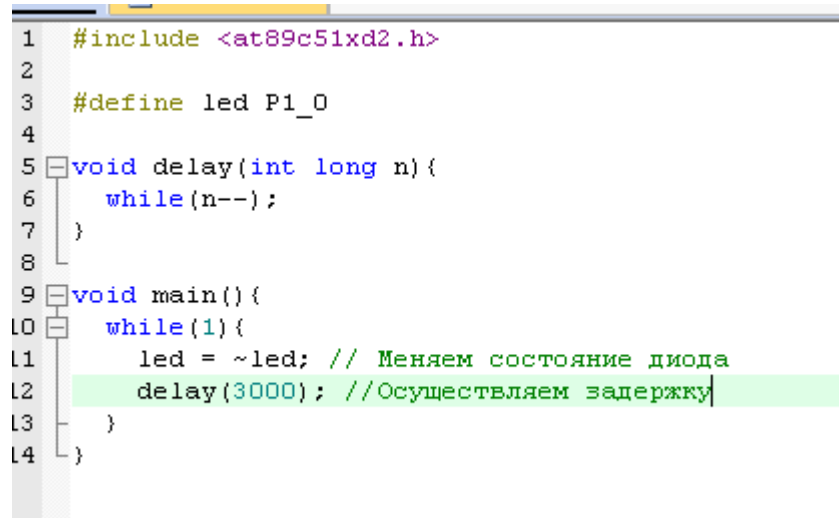
Программный пакет Proteus VSM позволяет собрать схему любого электронного устройства и симулировать его работу, выявляя ошибки, допущенные на стадии проектирования и трассировки. Программа состоит из двух модулей. ISIS – редактор электронных схем с последующей имитацией их работы. ARES – редактор печатных плат, оснащенный автотрассировщиком Electra, встроенным редактором библиотек и автоматической системой размещения компонентов на плате. Кроме этого ARES может создать трехмерную модель печатной платы.

Proteus VSM включает в себя более 6000 электронных компонентов со всеми справочными данными, а также демонстрационные ознакомительные проекты. Программа имеет инструменты USBCONN и COMPIM, которые позволяют подключить виртуальное устройство к портам USB и COM компьютера. При подсоединении к этим портам любого внешнего прибора виртуальная схема будет работать с ним, как если бы она существовала в реальности. Proteus VSM поддерживает следующие компиляторы: CodeVisionAVR и WinAVR (AVR), ICC (AVR, ARM7, Motorola), HiTECH

(8051, PIC Microchip) и Keil (8051, ARM). Существует возможность экспорта моделей электронных компонентов из программы Pspice.[9]

5.3 Написание и симуляция программы

Напишем в IDE Keil простейшую программу. Её содержание показано на рисунке 11



```
1  #include <at89c51xd2.h>
2
3  #define led P1_0
4
5  void delay(int long n){
6      while(n--);
7  }
8
9  void main(){
10     while(1){
11         led = ~led; // Меняем состояние диода
12         delay(3000); // Осуществляем задержку
13     }
14 }
```

Рисунок 11 – Код программы

Компилируем получившийся код, после чего создастся hex файл, который можно использовать в среде proteus

Создаём простую схему с диодом, микроконтроллером и осциллографом, она показана на рисунке 12, указываем микроконтроллеру путь до hex файла.

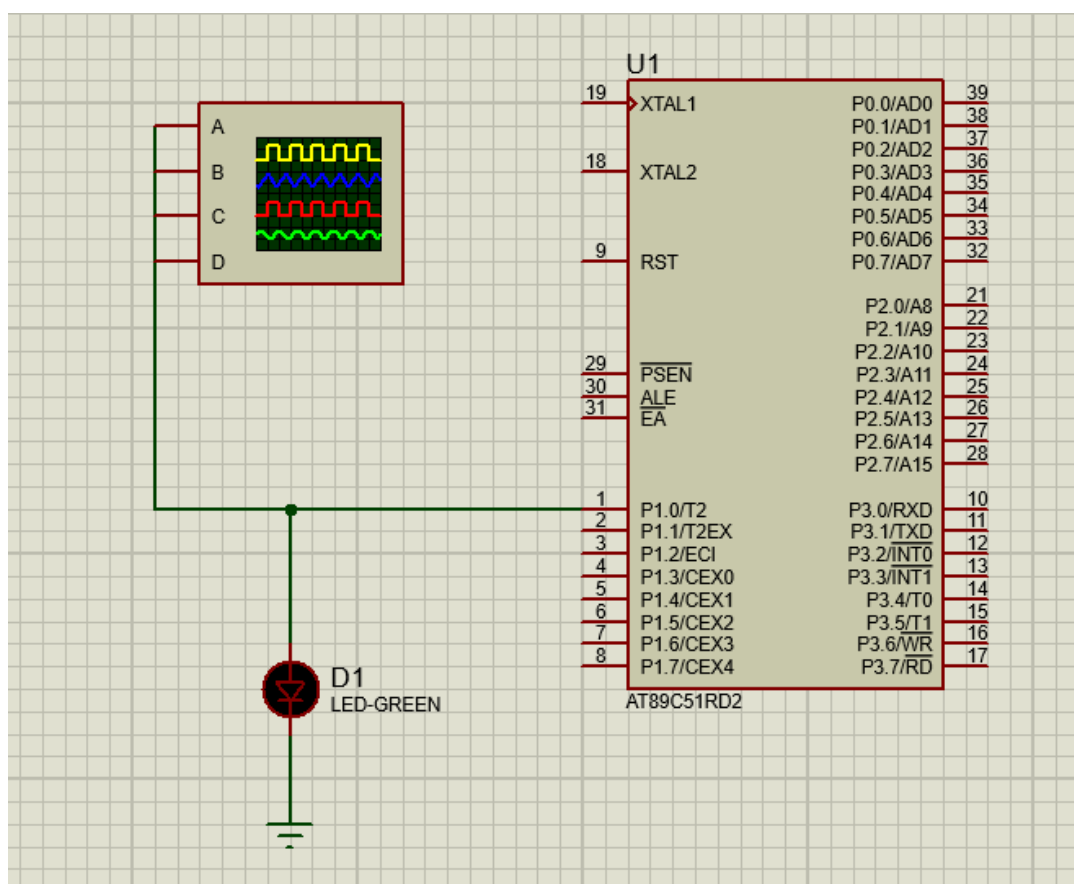


Рисунок 12 – Схема в среде proteus

Включаем симуляцию. На рисунке 13 показан скриншот симуляции на котором видно, что диод горит. На рисунке 14 показан скриншот окна осциллографа на котором видно, что сигнал подаваемый на диод меняет логическое состояние. Из увиденного следует, что диод то загорается, то гаснет. Это означает, что программа работает как ожидалось, а значит задание было выполнено верно.

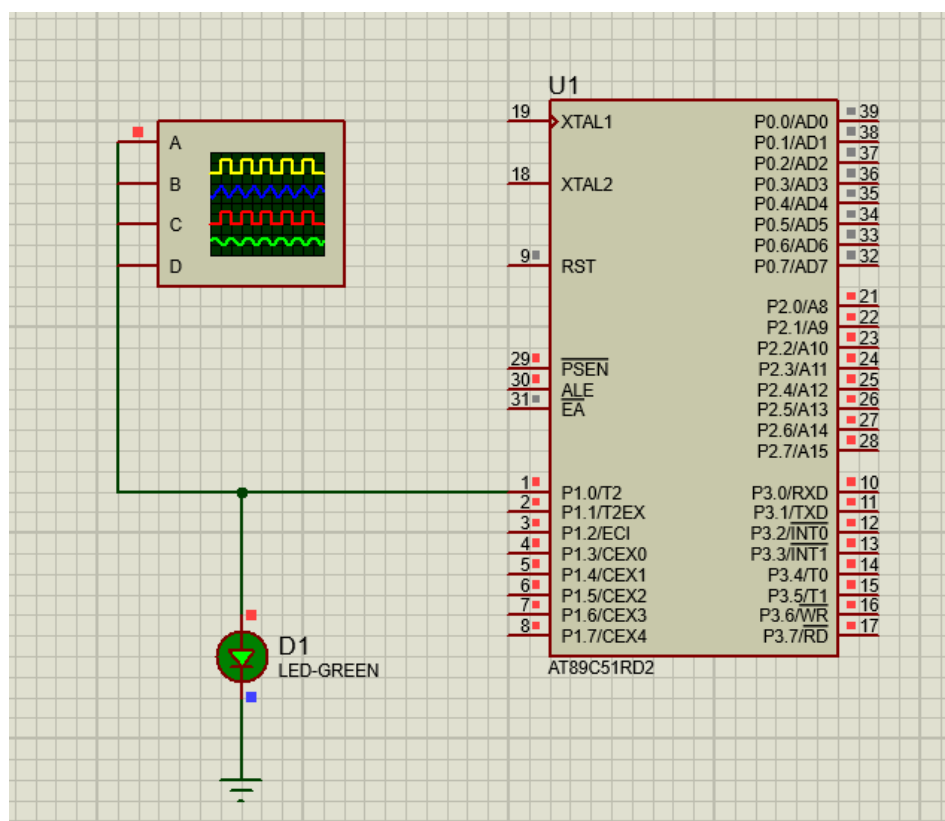


Рисунок 13 – Схема со включенной симуляцией

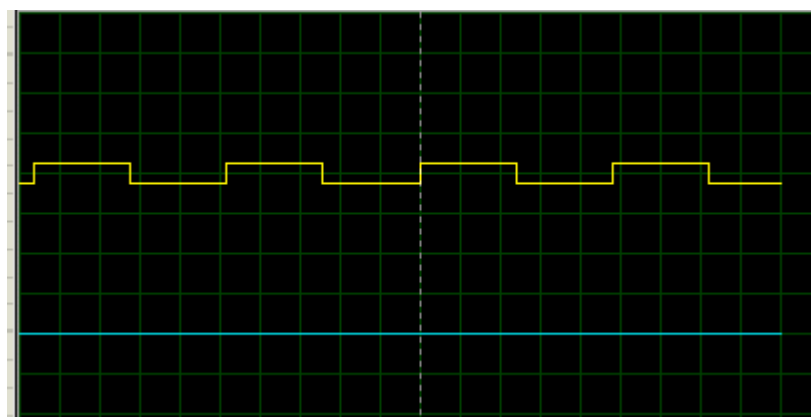


Рисунок 14 – Окно симуляции

6. Разработка схемы и программы в части сопряжения микропроцессора с элементами ввода информационного сигнала (ЦАП)

6.1 Сведения о ЦАП

Цифро-аналоговый преобразователь — устройство для перевода цифровых данных в аналоговый сигнал. В рамках данной работы будет использован AD5310

AD5310 – микросхема, содержащая один ЦАП, обеспечивающий формирование выходного напряжения 10-разрядным двоичным кодом,

питающийся однополярным напряжением от +2.5 В до +5.5 В и потреблением 115 мкА при 3В питания. В AD5310 встроен 3-проводный последовательный интерфейс, который работает на частоте до 30МГц и совместим со стандартными интерфейсами SPI, QSPI, MICROWIRE и DSP.

Характеристики:

- Один 10-разрядный ЦАП
- Поставка в 6-выводном SOT-23 и 8-выводном μ SOIC корпусах
- Малое энергопотребление: 140 мкА при 5 В питания
- Режим пониженного потребления: 200 нА при 5В, 50 нА при 3 В
- Питание от источника напряжением от +2.5В до +5.5В
- Гарантируется равномерность дифференциальной нелинейности для любых значений входного кода
- В качестве опорного напряжения используется напряжение питания
- Сброс в ноль выходного напряжения при подаче питания
- Три режима пониженного энергопотребления
- Экономичный последовательный интерфейс с триггерами Шмитта на входах
- Встроенный усилитель-повторитель выходного напряжения
- Вход SYNC для прерывания записи.

Структурная схема устройства изображена на рисунке 15, а на рисунке 16 показано расположение выводов ЦАПа.

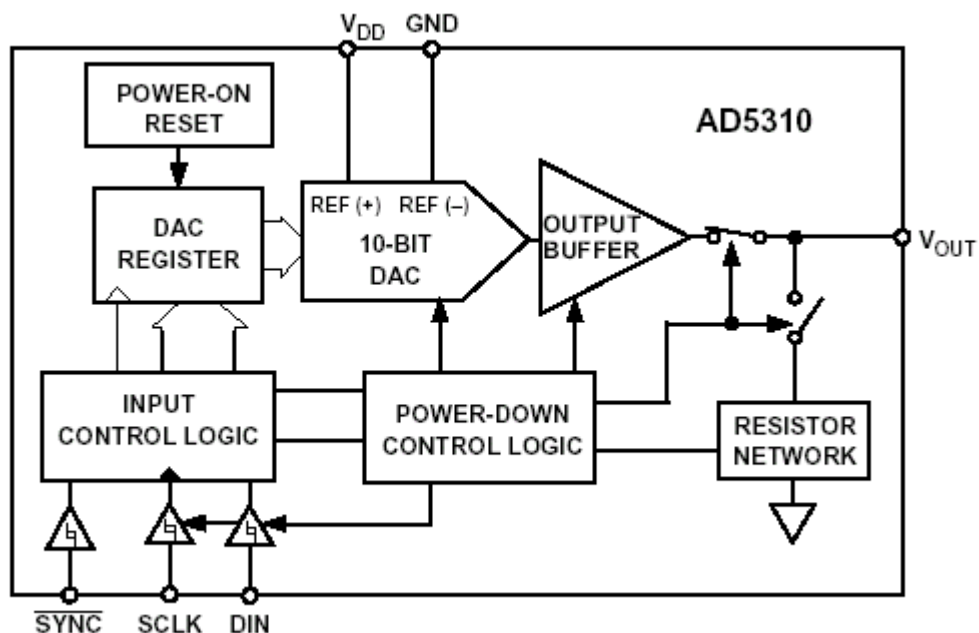


Рисунок 15 – Структурная схема устройства

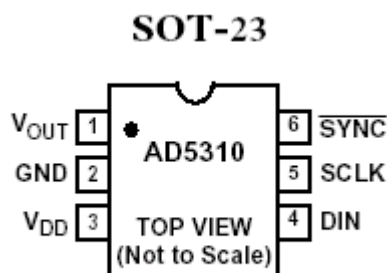


Рисунок 16– Расположение выводов ЦАП

Описание выводов:

- V_{DD} – Вход питания
- V_{OUT} – Выходное усиленное напряжение
- GND – Общий для всех частей микросхемы
- SYNC - Управляющий вход с активным низким уровнем (выбор микросхемы).
- SCLK – Вход синхронизации последовательной связи
- DIN – Ввод последовательных данных[10]

Временная диаграмма передачи данных на ЦАП показана на рисунке 17. Можно увидеть, что чтобы передать данные вывод SYNC должен быть притянут к земле во время передачи всех 16 бит.

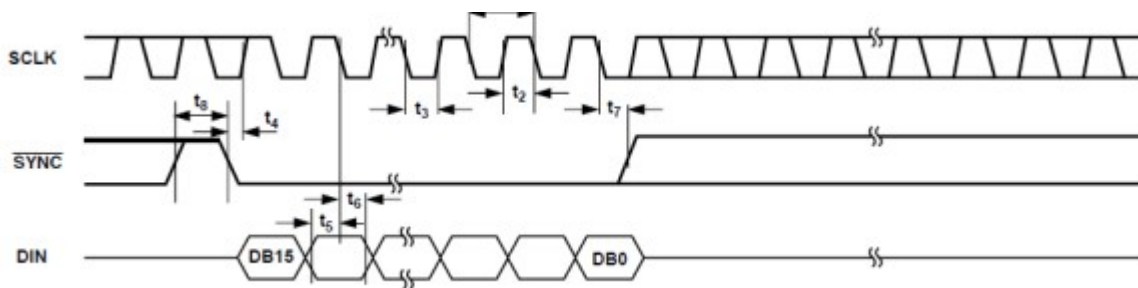


Рисунок 17 – Временная диаграмма передачи данных на ad5310

Конкретное содержание 16 бит передаваемых данных показано на рисунке 18. Биты PD1 и PD0 – задают режим работы ЦАП, мы будем использовать режим по умолчанию, так что там всегда будут нули. Биты с D9 до D0 – биты данных, через которые задаётся значение формируемое на выходе ЦАП. Обратите внимание, что нулевой бит находится справа.

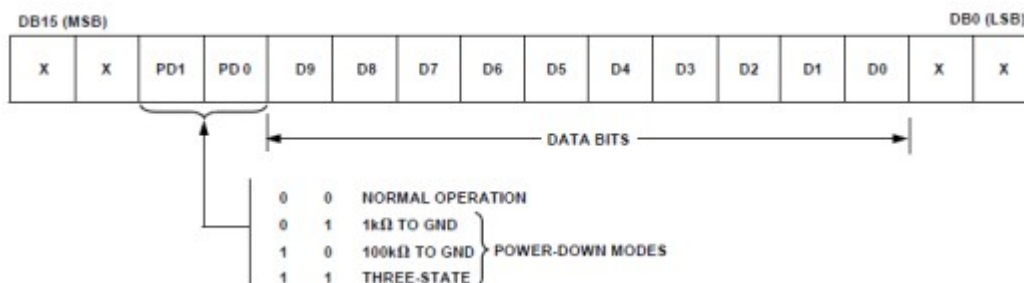


Рисунок 18 – Содержимое входного регистра данных

6.2 Сведения о передатчике значений

Для передачи данных ЦАП был выбран синхронный последовательный порт из-за его простоты и понятности. В AT89C51ED2 он реализован в виде нулевого режима работы последовательного порта. В рамках этого режима информация передаётся, и принимается через вывод входа приемника RxD. Принимаются или передаётся 8 бит данных. Для нас это значит, что передаваемые данные придётся дробить на два байта. Через вывод TxD выдаются импульсы синхронизации, которые сопровождают каждый бит. Скорость передачи фиксирована и составляет $1/6 F_{ген.}$ Для его настройки в выбранном микроконтроллере вообще не нужно ничего куда записывать.

Передача по последовательному порту начинается после записи байта в регистр данных SBUF. Временная диаграмма сигнала, вырабатываемого последовательным портом микроконтроллера при передаче восьми бит данных приведена на рисунке 19. Снова обращаю внимание на положение нулевого бита в передаваемом байте. В данном случае он находится слева. Это значит, что чтобы корректно передать на ЦАП значение придётся реверсировать порядок битов. Так же, на временной диаграмме можно заметить, что после окончания передачи устанавливается флаг TI, сигнализирующий об окончании передачи, для того, чтобы последовательный порт работал корректно его нужно сбрасывать по окончании передачи байта.

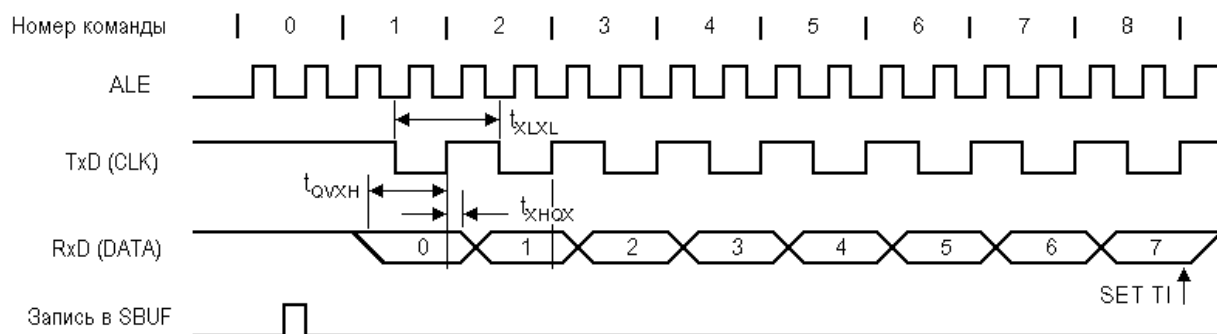


Рисунок 19 – Временная диаграмма работы последовательного порта в нулевом режиме после записи передаваемого байта в регистр данных SBUF.

6.3 Написание программы для работы с ЦАП

Итак, чтобы послать корректные данные на ЦАП нужно решить несколько задач:

- Разбить значение на байты
- Расположить значение внутри двух байтов так, чтобы оно соответствовало порядку битов данных в регистре данных ЦАП
- Реверсировать порядок битов значения

Для того, чтобы это сделать было создано несколько функций и макросов. Содержание кода этапа можно найти в приложении 3.

Рассмотрим происходящее внутри кода.

Чтобы передача данных не затягивалась было решено вынести разбивку и форматирование переселаемого значения в начало функции по пересылке значения `shift_transmit`. Следующая строчка реверсирует порядок битов значения, вызывая функцию `reverse`:

```
data_to_transmit = reverse(data_to_transmit);
```

Функция `reverse` довольно незамысловата. На входе она принимает число, а на выходе выдаёт его копию, но с зеркально расположенными битами. Например, если на вход поступит «0111101010», то на выходе будет уже «0101011110»

Далее, разобьём значение на два бита и расположим их в требуемых местах байтов данных с помощью вызовов макросов `extract_first_byte` и `extract_second_byte`, которые просто выполняют побитовый сдвиг влево или в право

```
first_byte = extract_first_byte(data_to_transmit);
second_byte = extract_second_byte(data_to_transmit);
```

Таким образом, наши биты готовы к отправке. Обнуляем вывод sync, чтобы сигнализировать ЦАП, что передача данных началась:

```
sync = 0;
```

Далее мы записываем первый бит данных в SBUF, ждём окончания передачи и обнуляем флаг окончания передачи:

```
sync = 0;  
SBUF = first_byte;  
while(TI==0);  
TI = 0; TI = 0;
```

Продельываем тоже самое с вторым байтом данных и устанавливаем выход sync в единицу, сигнализируя ЦАП, что передача значения окончена

```
SBUF = second_byte;  
while(TI==0);  
TI = 0; TI = 0;  
sync = 1;
```

Для того, чтобы продемонстрировать работоспособность написанного кода передадим на ЦАП пилообразный сигнал. Для этого соберём схему в proteus как показано на рисунке 20 и загрузим код из приложения 3. На выходе получится пилообразный сигнал изменяющийся в диапазоне от 0 до 5 В. Выходной сигнал показан на рисунке 21.

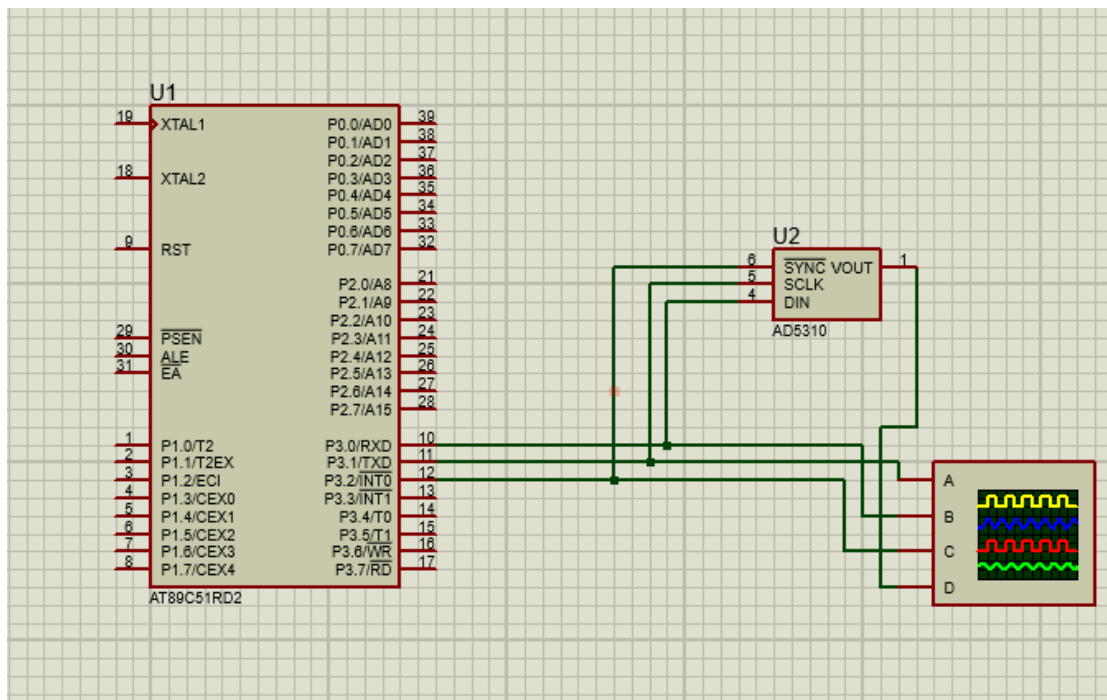


Рисунок 20 – Схема подключения ЦАП к микроконтроллеру

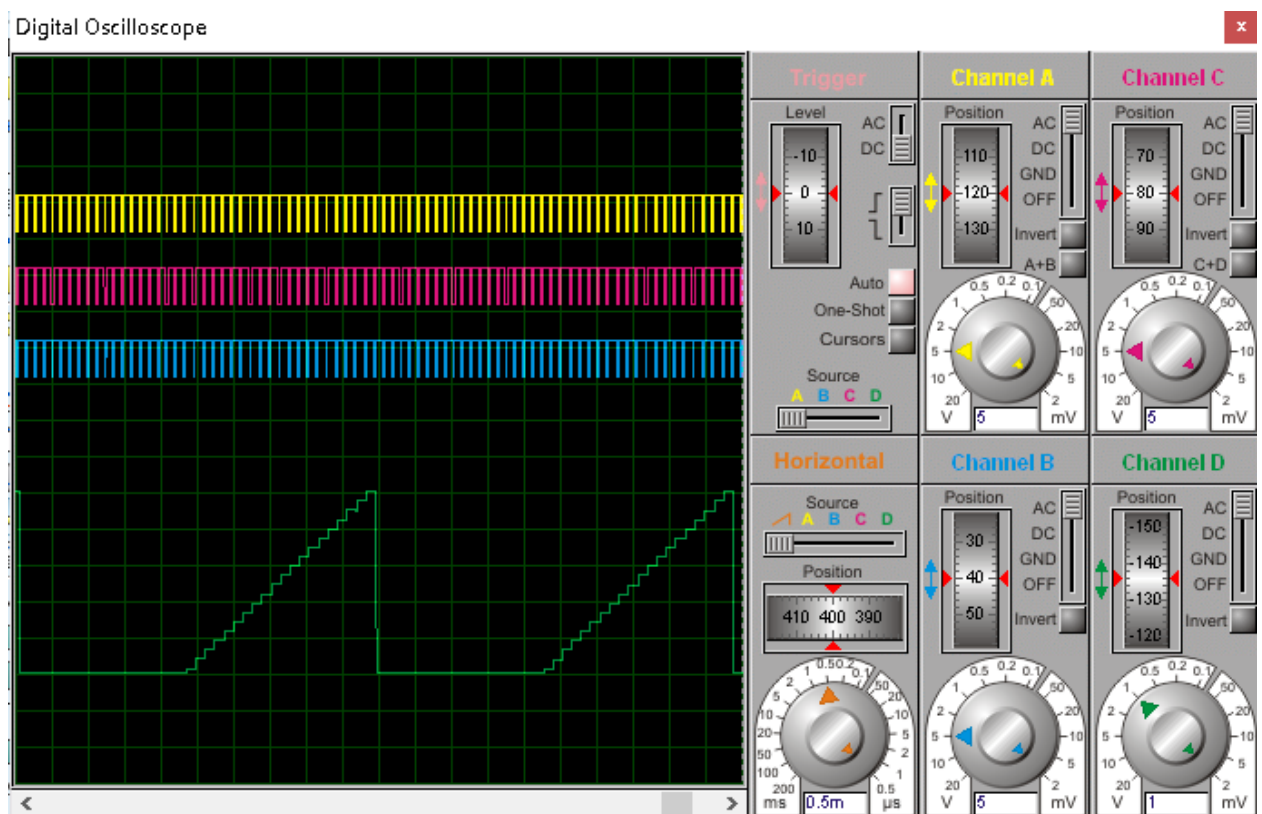


Рисунок 21 – Осциллограмма выходного сигнала ЦАП(зелёным)

7. Разработка схемы и программы в части сопряжения микропроцессора с элементами ввода информационного сигнала (АЦП)

7.1 Сведения об АЦП

АЦП – устройство, преобразующее входной аналоговый сигнал в дискретный код. В цифровой фильтрации сигналов АЦП необходимы. Согласно ТЗ нужно использовать ЦАП AD1674

AD1674 – многоцелевой 12 битный аналого-цифровой преобразователь, содержащий встроенные устройство выборки-хранения (УВХ), 10 В источник опорного напряжения (ИОН), буфер тактовых импульсов и выходной буфер с тремя состояниями для связи с микроконтроллером.

Устройство имеет следующие характеристики:

- Монолитный 12 битный 10 мс АЦП со схемой выборки
- Встроенное устройство выборки-хранения
- Цоколевка, соответствующая промышленному стандарту
- 8 и 16 битный микропроцессорный интерфейс
- Определенные и проверенные статические и динамические характеристики
- Униполярный и биполярный входы
- Диапазоны входного сигнала ± 5 В, ± 10 В, 0 В - 10 В и 0 В - 20 В
- Коммерческий, промышленный и военный температурные диапазоны
- MIL-STD-883 и SMD корпусные исполнения[12]

Функциональная схема показана на рисунке 22. Расположение выводов показано на рисунке 23. Цифровые выводы устройства:

- A_0 – При преобразовании отвечает за его битность. При чтении отвечает за зануление младших битов
- CE – Включение устройства
- CS – Выбор устройства. Активное состояние – 0
- DB* – Параллельные выводы результата преобразования. Один вывод – один бит

-

Pin	Signal	Pin	Signal
1	V_{LOGIC}	28	STS
2	12/8	27	DB11(MSB)
3	$\overline{\text{CS}}$	26	DB10
4	A_0	25	DB9
5	R/C	24	DB8
6	CE	23	DB7
7	V_{CC}	22	DB6
8	REF OUT	21	DB5
9	AGND	20	DB4
10	REF IN	19	DB3
11	V_{EE}	18	DB2
12	BIP OFF	17	DB1
13	10V _{IN}	16	DB0(LSB)
14	20V _{IN}	15	DGND

Рисунок 23 – Расположение выводов AD1674

Схема подключения АЦП в униполярном режиме показана на рисунке 24. Она понадобится нам при моделировании схемы.

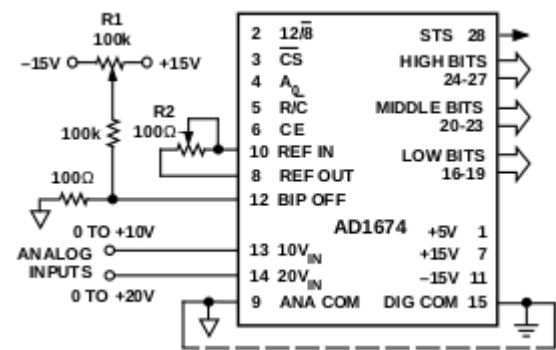


Рисунок 24 – Схема подключения АЦП в униполярном режиме

Таблица истинности для входов АЦП показана в таблице 4. Она нужна, чтобы лучше ориентироваться в режимах работы АЦП

Таблица 4 – Таблица истинности входов АЦП

CE	CS	R/C	12/8	A ₀	Операция
0	x	x	x	x	Ничего
x	1	x	x	x	Ничего
1	0	0	x	0	Начало 12-битного преобразования
1	0	0	x	1	Начало 8-битного преобразования
1	0	1	1	x	Считывание в 12-битном режиме
1	0	1	0	0	Считывание 8 старших битов, младшие в нуле
1	0	1	0	1	Считывание 4 битов по бокам, 4 бита по середине = 0

Так же, для того, чтобы понимать как происходит работа с АЦП, нужно иметь ввиду две временные диаграммы. Первая – состояние битов АЦП при преобразовании, показано на рисунке 25. Вторая – состояние битов при процессе считывания, показана на рисунке 26

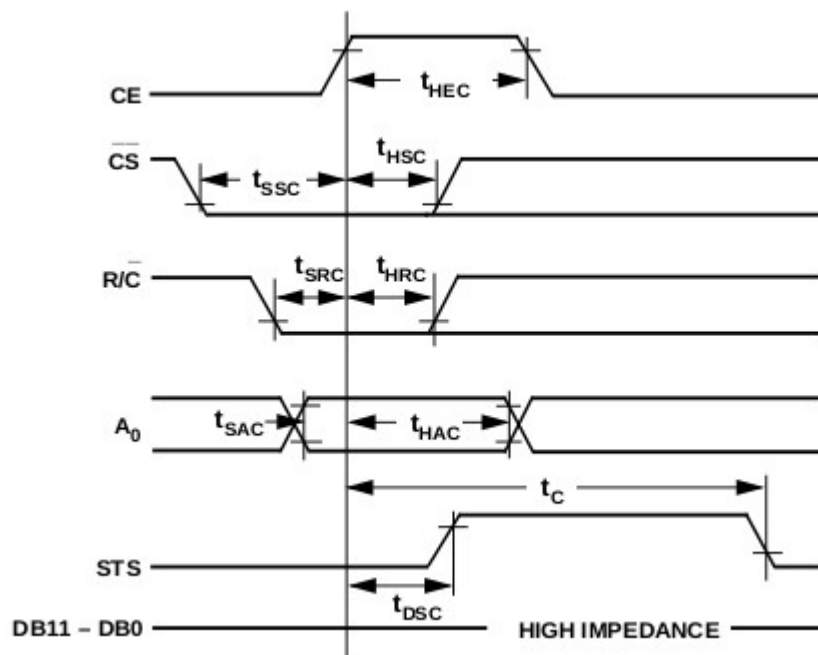


Рисунок 25 – Временная диаграмма преобразования

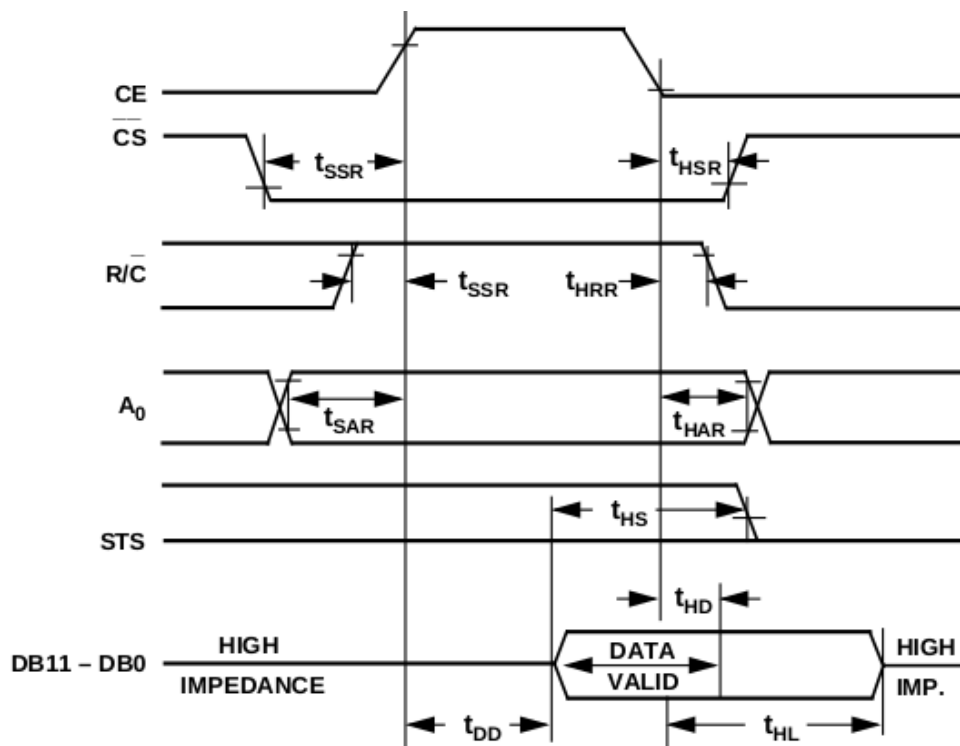


Рисунок 26 – Временная диаграмма считывания результатов преобразования.

7.2 Написание программы и моделирование работы АЦП

Начать стоит с построения схемы. Имея ввиду рисунок 24 результат сборки схемы в proteus показан на рисунке 27.

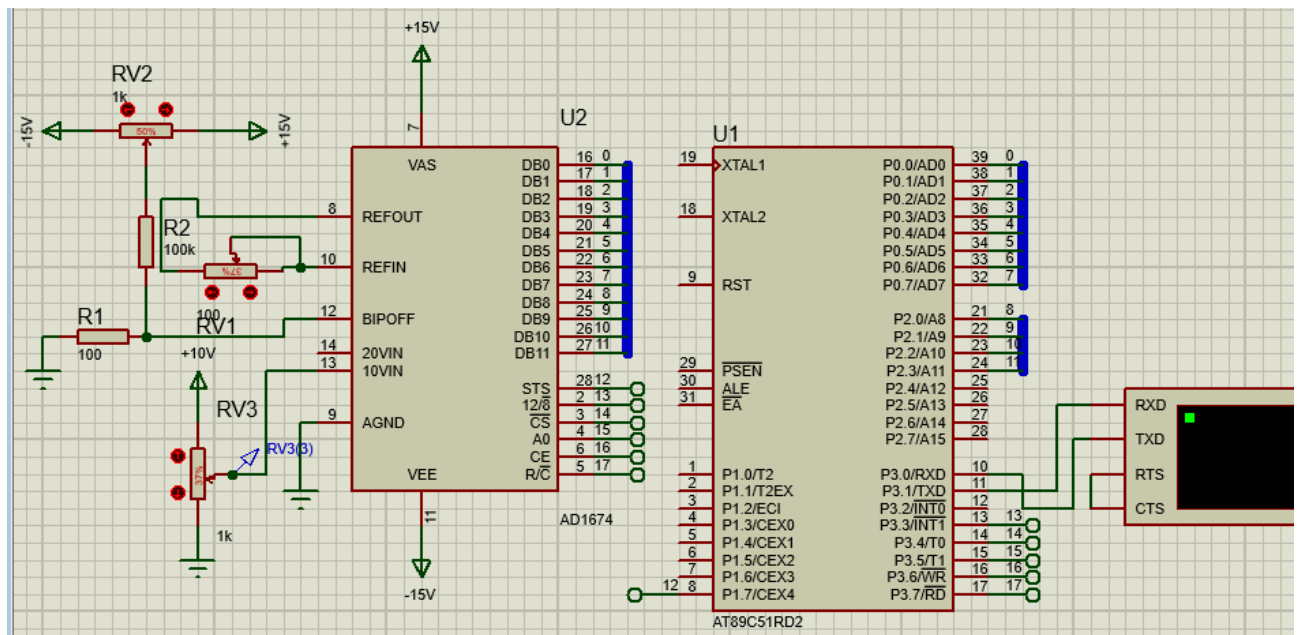


Рисунок 27 – Схема подключения AD1674 к AT89C51ED2

Теперь разберём программу. Будут рассмотрены только части кода связанные непосредственно с AD1674. Полный код программы приведён в приложении 4

Первым делом, определим режим работы АЦП. Для наибольшей точности будет использован 12 битный режим преобразования. При считывании будут передаваться все биты. Это задаётся следующей строчкой:
`bit_12_8 = 1; CS = 0; A0 = 0; CE = 1;`

Далее, нужно преобразовать сигнал и считать его. Это делается с помощью функции `ad1674_read`. Она показана ниже. Сначала мы обнуляем R/C бит, чтобы начать преобразование. Затем, ждём окончания преобразования, обнуления бита STS. Ну и наконец устанавливаем бит R/C и ждём один такт, чтобы считать результат преобразования. Наконец возвращаем результат преобразования из функции.

```
float ad1674_read(void){  
    R_C = 0;
```

```

while(STS==1);
R_C = 1; R_C = 1;
return P0 | ((P2 & 0xF) << 8);
}

```

Всё, на этом непосредственно работа с АЦП закончена. Далее мы преобразуем полученное значение в вольты и выводим с помощью uart. Результат показан на рисунках 28 и 29.

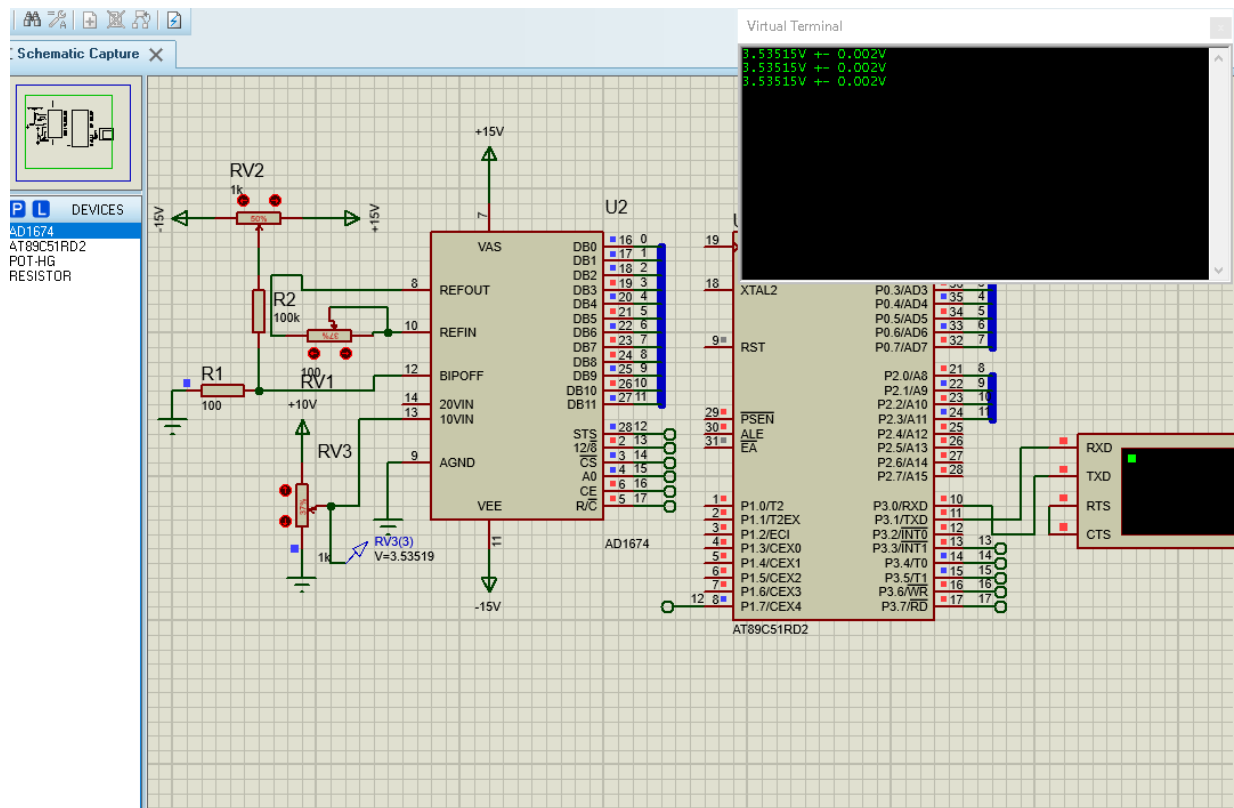


Рисунок 28 – Результат преобразования при напряжении на входе равном 3.53519 В

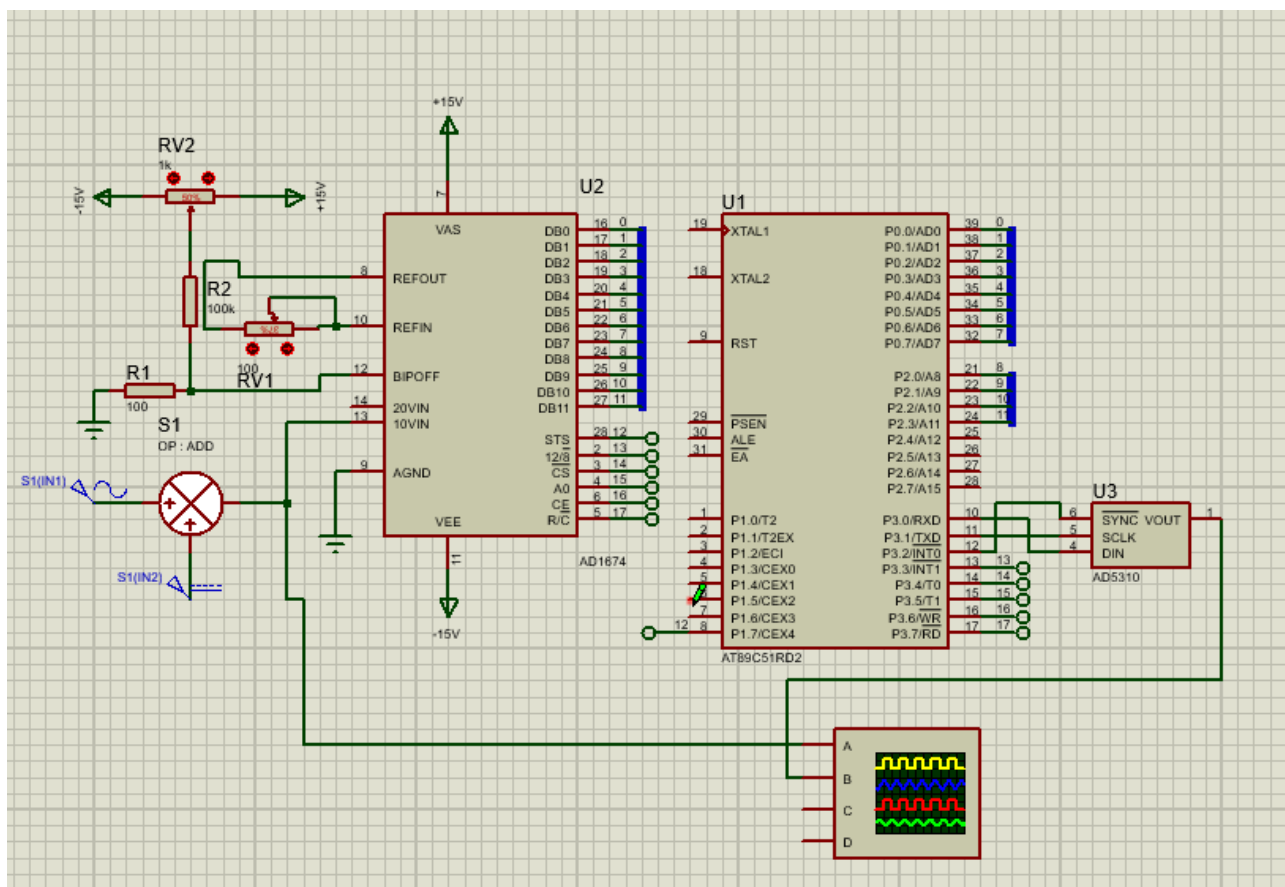


Рисунок 30 – Схема сквозного тракта АЦП – ЦАП

Код составлен из частей 6-ого и 7-ого этапов, а значит мы не будем заострять на нём внимание. Его можно посмотреть в приложении 5.

Чтобы протестировать схему были поданы синусоидальный сигнал, объединённый с постоянным сигналом, чтобы приподнять их над нулём (наш АЦП работает в униполярном режиме). Синусоидальный сигнал имеет период 300 Гц и изменяется в диапазоне от 0 до 10В. На выходе получается слегка сдвинутый сигнал той же частоты изменяющийся в диапазоне от 0 до 5В. Разница обусловлена диапазонами работы АЦП и ЦАП. Результат работы схемы с синусоидальным сигналом показан на рисунке 31

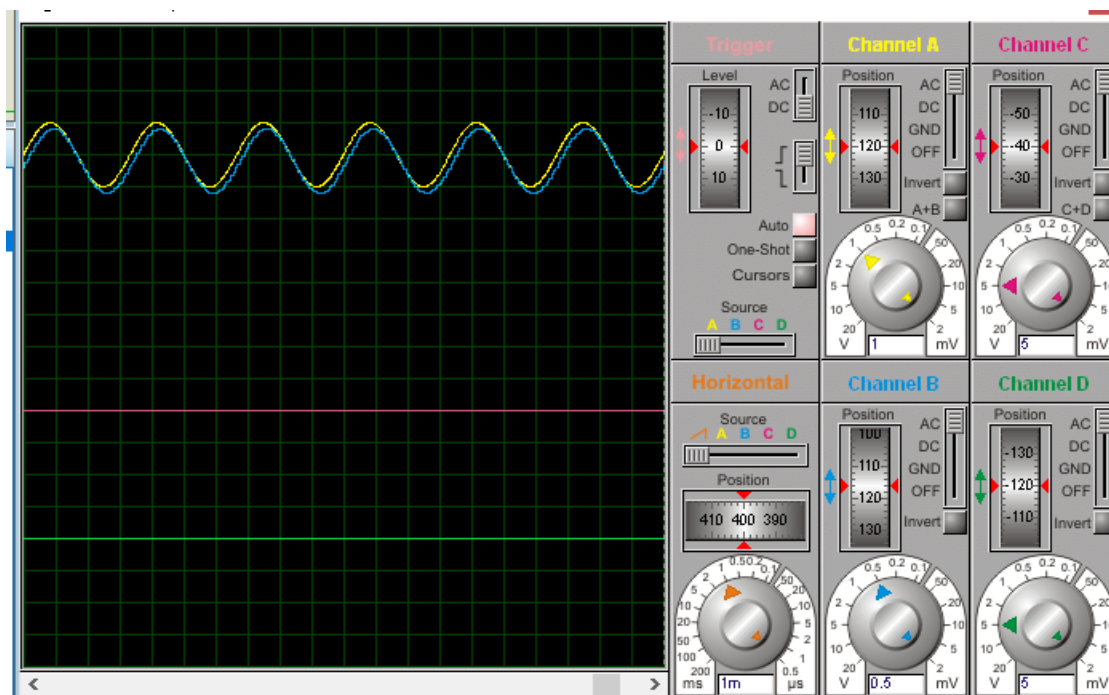


Рисунок 31 – Результат работы схемы с синусоидальным сигналом.

Жёлтый – входной, синий – выходной.

Можно заметить, смотря на значения смещения сигналов осциллографа, что сигнал на выходе устройства слегка приподнят. Чтобы устранить Несоответствия входного и выходного сигналов на выход был поставлен ОУ, как показано на рисунке 32. В результате мы имеем оцилограмму показанную на рисунке 33. Видно, что она полностью совпадает со входной и в амплитуде и по смещению относительно нуля.

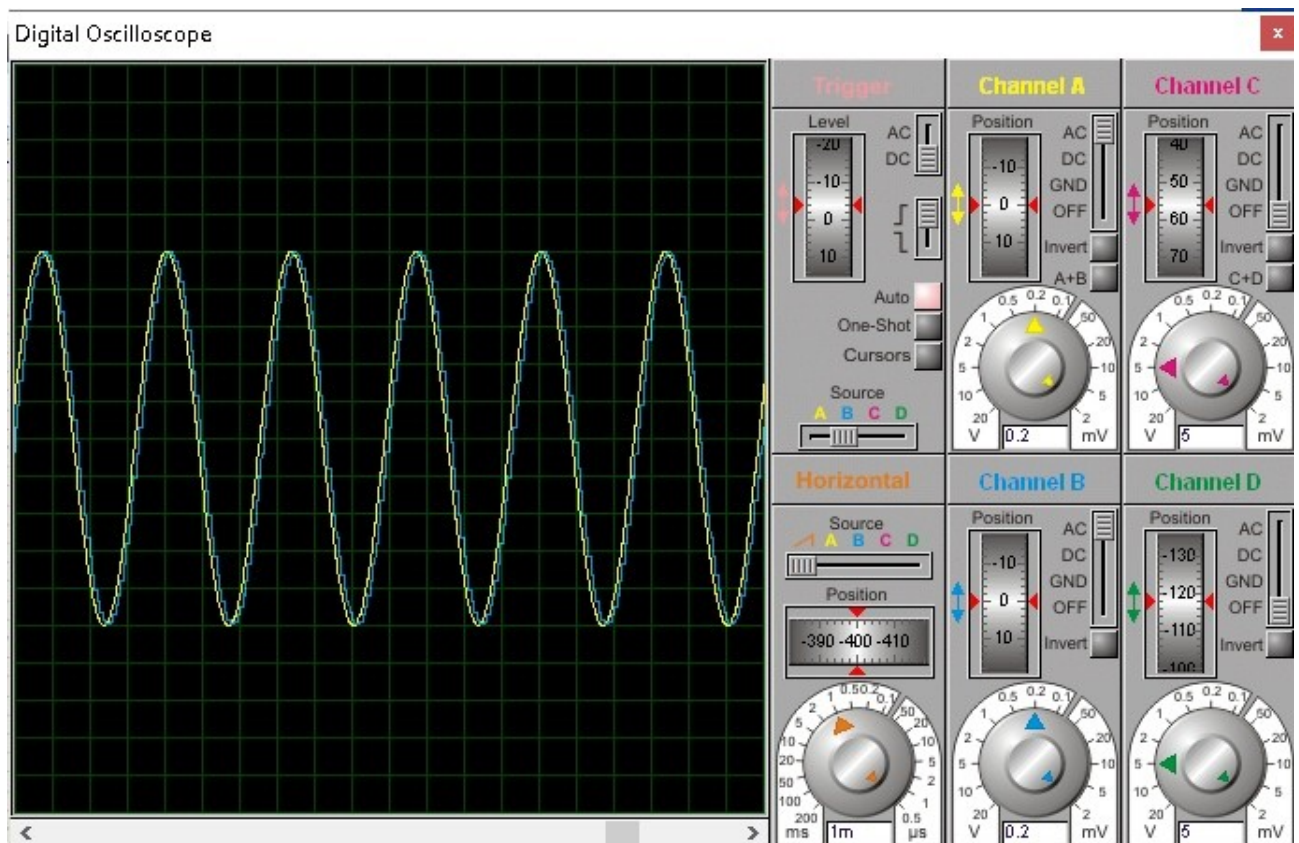


Рисунок 33 – Результат работы схемы с ОУ с синусоидальным сигналом.
Жёлтый – входной, синий – выходной.

ЗАКЛЮЧЕНИЕ

В результате был разработан и промоделирован цифровой узкополосный полосовой фильтр с фиксированными параметрами, с заданными характеристиками на базе микроконтроллера 8051, при использовании внутреннего АЦП и внешнего ЦАП.

Получены навыки разработки, моделирования и программирования цифрового фильтра, подключения и программирования внешнего ЦАП. В ходе работы сперва были проведены необходимые расчёты цифрового фильтра на языке программирования python 3.7, потом была написана программа для микроконтроллера 8051 в среде программирования keil и в конце произведено моделирование в программе Proteus 8 Proffesional. Устройство удовлетворяет всем требованиям технического задания.

Список литературы:

1. Лем Г. Аналоговые и цифровые фильтры //Москва: Изд-во «Мир». – 1982.
2. Цифровые фильтры [Электронный ресурс] // Сборник технических статей : [сайт]. [2010]. URL: <http://www.teh-lib.ru/cimpu/cifrovye-filtry.html> (Дата обращения: 26.09.2020)
3. Карташев В. Г. Основы теории дискретных сигналов и цифровых фильтров: Учеб. пособие для вузов. — М.: Высш. школа, 1982. — 109 с.
4. Расчёт и программная реализация равнополочного банка цифровых фильтров на DSP TMS320VC5510 [Электронный ресурс] // studwood.ru : [сайт]. URL: https://studwood.ru/1729974/tehnika/tipy_tsifrovyh_filtrov_osobennosti (Дата обращения: 26.09.2020)
5. Типы цифровых фильтров выбор между ними [Электронный ресурс] // Студопедия.Нет - Информационный студенческий ресурс : [сайт]. URL: https://studopedia.net/3_48300_tipi-tsifrovih-filtrov-vibor-mezhdu-nimi.html (Дата обращения: 26.09.2020)
6. Цифровые фильтры частотной селекции: учебное пособие / О.О. Жаринов, И.О. Жаринов.СПб: Изд-во ГУАП, 2019. – 77 с. [библиотечный шифр 621.372 Ж 34].
7. AT89C51RB2/RC2 - Complete Datasheet. [Электронный документ] URL:<https://ww1.microchip.com/downloads/en/DeviceDoc/doc4180.pdf> (Дата обращения 9.02.2021)
8. Keil uVision – среда разработки программного обеспечения. [Электронный ресурс] // Сборник технических статей <https://cxem.net> : [сайт]. [2010]. URL: <https://cxem.net/software/keil.php> (Дата обращения 9.02.2021)
9. Программа Proteus - рисование электронных схем. [Электронный ресурс] // Сборник технических статей <https://cxem.net> : [сайт]. URL:<https://cxem.net/software/proteus.php> (Дата обращения 9.02.2021)

10. AD5310 Datasheet. [Электронный документ]
URL:<https://www.analog.com/media/en/technical-documentation/data-sheets/ad5310.pdf> (Дата обращения 11.02.2021)
11. Последовательный порт микроконтроллера 8051. [Электронный ресурс] // Сборник технических статей digteh.ru : [сайт].
URL:<https://digteh.ru/MCS51/PoslPort.php> (Дата обращения 11.02.2021)
12. AD1674 Datasheet. [Электронный документ]
URL:<https://www.analog.com/media/en/technical-documentation/data-sheets/AD1674.pdf> (Дата обращения 13.02.2021)

ПРИЛОЖЕНИЕ 1

```
# Const
delta_T = 1/9000
q = 2048
```

```
def generate_recursive_filter(z, fn=0, fv=1):
```

```
    """
```

Генерирует рекурсивный фильтр с заданными параметрами и его коэффициенты

```
    :z: оператор передаточной функции
    :fn: Нижняя граница частоты среза фильтра
    :fv: Верхняя граница частоты среза фильтра
    :returns: Теоритический фильтр по аналоговой передаточной функции
    :returns: Alpha коэффициенты фильтра
    :returns: Beta коэффициенты фильтра
    """
```

```
    import numpy as np
```

```
    # Преобразовываем частоты для правильной работы
```

```
    fn = fn * 2 * np.pi
```

```
    fv = fv * 2 * np.pi
```

```
    gamma = 1/np.tan((delta_T/2) * (fv - fn))
```

```
    zeta = np.cos((delta_T/2) * (fv + fn))/np.cos((delta_T/2) * (fv - fn))
```

```
    # переменные используемые по формуле
```

```
    filter_order = 2
```

```
    order_number = 1
```

```
    # Непосредственно генерируем фильтр
```

```
    alphas = []
```

```
    betas = []
```

```
    our_filter = 1
```

```
    a_constants = [1, -2 * np.cos(((2 * order_number + filter_order - 1)/(2 *
filter_order)) * np.pi), 1]
```

```
    current_betta = [1, 0, -2, 0, 1]
```

```
    current_alpha = [gamma**2 * a_constants[0] + gamma * a_constants[1] +
a_constants[2],
```

```

-4 * gamma**2 * zeta * a_constants[0] - 2 * gamma * zeta * a_constants[1],
  4 * gamma**2 * zeta**2 * a_constants[0] + 2 * gamma**2 * a_constants[0]
- 2 * a_constants[2],
  -4 * gamma**2 * zeta * a_constants[0] + 2 * gamma * zeta * a_constants[1],
  gamma**2 * a_constants[0] - gamma * a_constants[1] + a_constants[2]]

```

```

our_filter = our_filter * ((current_beta[0] * z**4 + current_beta[1] * z**3 +
current_beta[2] * z**2 + current_beta[3] * z + current_beta[4])/ \
  (current_alpha[0] * z**4 + current_alpha[1] * z**3 + current_alpha[2] * z**2 +
current_alpha[3] * z + current_alpha[4]))

```

```

alphas = np.array(current_alpha)/current_alpha[0]

```

```

betas = np.array(current_beta)/current_alpha[0]

```

```

return np.array(our_filter), np.array(alphas), np.array(betas)

```

```

def recursive_transfer_function(z, fn=0, fv=1):

```

```

    """

```

Генерирует теоритически заданный аналоговый прототип ПФ фильтра Баттерворта

соответственно входным данным

:z: оператор передаточной функции

:fn: Нижняя граница частоты среза фильтра

:fv: Верхняя граница частоты среза фильтра

:returns: рассчитанную передаточную функцию

```

    """

```

```

import numpy as np

```

```

fn = fn * 2 * np.pi

```

```

fv = fv * 2 * np.pi

```

```

# Вводим константы

```

```

gamma = 1/np.tan((delta_T/2) * (fv - fn))

```

```

zeta = np.cos((delta_T/2) * (fv + fn))/np.cos((delta_T/2) * (fv - fn))

```

```

# Осуществляем замену переменной p из ФНЧ в ПФ

```

```

changed_p = gamma * (((z**2) - 2 * zeta * z + 1)/((z**2) - 1))

```

```

# Записываем передаточную функцию фильтра Баттерворта 2-ого порядка

```

```

calculated_transfer_function = 1/((changed_p**2) + 1.41421 * changed_p + 1)

```

```

return calculated_transfer_function

```

```

def main():
    import numpy as np
    import matplotlib.pyplot as plt

    # Вычисляем частоту
    k = np.linspace(0, q, q)
    frequency = k/(q * delta_T)

    # Формируем массив коэффициентов
    coefficient = []
    for frequency_number in frequency:
        coefficient.append(complex(0, 2 * np.pi * frequency_number * delta_T))

    # Расчёт теоритического рекурсивного фильтра по передаточной функции
    filter_transfer_function =
recursive_transfer_function(np.exp(np.array(coefficient)), fv=410, fn=390)

    # Расчёт рекурсивного фильтра
    recursive_filter, A, B = generate_recursive_filter(np.exp(np.array(coefficient)),
fv=410, fn=390)
    fn_div_10 = 20 * np.log10(abs(generate_recursive_filter(np.exp(complex(0, 2 *
np.pi * 39 * delta_T)), fv=410, fn=390)[0]))
    fv_mul_10 = 20 * np.log10(abs(generate_recursive_filter(np.exp(complex(0, 2 *
np.pi * 4100 * delta_T)), fv=410, fn=390)[0]))

    # Вывод коэффициентов рекурсивного фильтра
    print("Альфа коэффициенты")
    for coef_number in range(0, len(A)):
        print(f'{coef_number + 1}-ый: {A[coef_number]:.64}')
    print('Бета коэффициенты')
    for coef_number in range(0, len(B)):
        print(f'{coef_number + 1}-ый: {B[coef_number]:.64}')

    # Сравниваем теоритическое АЧХ рекурсивного фильтра и АЧХ
рассчитаного фльтра
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.plot(frequency[:int(q/2)], 20 * np.log10(abs(filter_transfer_function))[:int(q/2)],
label='АЧХ фильтра рассчитанного по аналоговому прототипу')
    ax.plot(frequency[:int(q/2)], 20 * np.log10(abs(recursive_filter))[:int(q/2)],
label='АЧХ фильтра рассчитаного по таблице')

```

```

    ax.plot(frequency[:int(q/2)], -60 * np.ones(int(q/2)), '--', label='Граница
коэффициента подавления -60 дБ')
    ax.text(0, 0.5, f'Глубина затухания при 39 Гц = {round(fn_div_10)}',
transform=ax.transAxes, fontsize=14)
    ax.text(0, 0.4, f'Глубина затухания при 4100 Гц = {round(fv_mul_10)}',
transform=ax.transAxes, fontsize=14)
    ax.set_xlabel('Частота, Гц')
    ax.set_ylabel('Коэффициент подавления, дБ')
    ax.set_xscale('log')
    ax.legend()

plt.show()

if __name__ == "__main__":
    main()

```

ПРИЛОЖЕНИЕ 2

```
# Const
delta_T = 1/9000
N = 2000
q = 2048

def recursive_transfer_function(z, fn=0, fv=1):
    """
        Генерирует теоритически заданный аналоговый прототип ПФ фильтра
        Баттерворта
        соответственно входным данным

        :z: оператор передаточной функции
        :fn: Нижняя граница частоты среза фильтра
        :fv: Верхняя граница частоты среза фильтра
        :returns: рассчитанную передаточную функцию
    """
    import numpy as np

    fn = fn * 2 * np.pi
    fv = fv * 2 * np.pi

    # Вводим константы
    gamma = 1/np.tan((delta_T/2) * (fv - fn))
    zeta = np.cos((delta_T/2) * (fv + fn))/np.cos((delta_T/2) * (fv - fn))

    # Осуществляем замену переменной p из ФНЧ в ПФ
    changed_p = gamma * (((z**2) - 2 * zeta * z + 1)/((z**2) - 1))

    # Записываем передаточную функцию фильтра Баттерворта 2-ого порядка
    calculated_transfer_function = 1/((changed_p**2) + 1.41421 * changed_p + 1)

    return calculated_transfer_function

def generate_recursive_filter(z, fn=0, fv=1):
    """
        Генерирует рекурсивный фильтр с заданными параметрами и его
        коэффициенты

        :z: оператор передаточной функции
```

```

:fn: Нижняя граница частоты среза фильтра
:fv: Верхняя граница частоты среза фильтра
:returns: Теоритический фильтр по аналоговой передаточной функции
:returns: Alpha коэффициенты фильтра
:returns: Beta коэффициенты фильтра
"""

import numpy as np

# Преобразовываем частоты для правильной работы
fn = fn * 2 * np.pi
fv = fv * 2 * np.pi

gamma = 1/np.tan((delta_T/2) * (fv - fn))
zeta = np.cos((delta_T/2) * (fv + fn))/np.cos((delta_T/2) * (fv - fn))

# переменные используемые по формуле
filter_order = 2
order_number = 1

# Непосредственно генерируем фильтр
alphas = []
betas = []
our_filter = 1
a_constants = [1, -2 * np.cos(((2 * order_number + filter_order - 1)/(2 *
filter_order)) * np.pi), 1]

current_beta = [1, 0, -2, 0, 1]

current_alpha = [gamma**2 * a_constants[0] + gamma * a_constants[1] +
a_constants[2],
-4 * gamma**2 * zeta * a_constants[0] - 2 * gamma * zeta * a_constants[1],
4 * gamma**2 * zeta**2 * a_constants[0] + 2 * gamma**2 * a_constants[0]
- 2 * a_constants[2],
-4 * gamma**2 * zeta * a_constants[0] + 2 * gamma * zeta * a_constants[1],
gamma**2 * a_constants[0] - gamma * a_constants[1] + a_constants[2]]

our_filter = our_filter * ((current_beta[0] * z**4 + current_beta[1] * z**3 +
current_beta[2] * z**2 + current_beta[3] * z + current_beta[4])/ \
(current_alpha[0] * z**4 + current_alpha[1] * z**3 + current_alpha[2] * z**2 +
current_alpha[3] * z + current_alpha[4]))
alphas.append(np.array(current_alpha)/current_alpha[0])
betas.append(np.array(current_beta)/current_alpha[0])

```



```

return np.array(our_filter), np.array(alphas), np.array(betas)

def signal_generate(time, harmonics):
    """
    Генерирует сигнал с заданными параметрами

    :harmonics: гармоники сигнала в виде словаря {амплитуда:частота} для
    каждой гармоники
    :time: отсчёты по x-оси
    :returns: сигнал с заданными параметрами
    """
    from numpy import sin, pi, zeros

    generated_signal = zeros(len(time))

    for ind, harmonic in enumerate(harmonics):
        generated_signal += harmonic * sin(2 * pi * harmonics[harmonic] * time)

    return generated_signal

def compare_signals(regular_signal, filtered_signal, plot_range, shift_point):
    """
    Строит график сигнала и его АЧХ. Как бы показывает разницу между
    сигналами на графике

    :regular_signal: обычный сигнал
    :filter_signal: фильтрованный сигнал
    :shift_point: Точка сдвига фильтрованного сигнала. Нужна чтобы убрать
    задержку и было легче сравнить
    сигналы
    plot_range: отрезок на котором будет построен график
    """
    import matplotlib.pyplot as plt
    import numpy as np

    plt.figure()
    plt.plot(plot_range, regular_signal[plot_range], label='Нефильтрованный')
    plt.plot(plot_range, filtered_signal[np.array(plot_range) + shift_point],
    label='Фильтрованный')
    plt.xlabel('Отсчёты')
    plt.ylabel('Амплитуда')

```

```
plt.legend()
```

```
def signal_plot(input_signal):
```

```
    """
```

```
    Строит график сигнала и амплитудного спектра
```

```
    :input_signal: входной сигнал
```

```
    """
```

```
    import matplotlib.pyplot as plt
```

```
    import numpy as np
```

```
    # Вычисляем частоту сигнала
```

```
    k = np.arange(0, int(len(input_signal)/2), 1)
```

```
    frequency = k/(int(len(input_signal)) * delta_T)
```

```
    # Строим сигнал
```

```
    plt.figure()
```

```
    plt.xlabel('Отсчёты')
```

```
    plt.ylabel('Амплитуда')
```

```
    plt.plot(frequency, input_signal[:int(N/2)])
```

```
def signal_filtration_with_recursive_filter(signal_to_filter, A_coef, B_coef):
```

```
    """
```

```
    Фильтрация сигнала с помощью рекурсивного фильтра
```

```
    :signal_to_filter: сигнал, который нужно отфильтровать
```

```
    :A_coef: Alpha коэффициенты фильтра
```

```
    :B_coef: Beta коэффициенты фильтра
```

```
    :return: Фильтрованный сигнал
```

```
    """
```

```
    import numpy as np
```

```
    filtered_signal = []
```

```
    # Итерация по отсчётам сигнала
```

```
    for current_count in range(0, len(signal_to_filter)):
```

```
        filtered_signal.append(0)
```

```
        # Итерация по коэффициентам beta конкретного каскада
```

```
        for current_beta_coefficient in range(0, len(B_coef[0])):
```

```
            if current_count - current_beta_coefficient >= 0:
```

```
                filtered_signal[current_count] += B_coef[0][current_beta_coefficient] * \
                    signal_to_filter[current_count - current_beta_coefficient]
```

```

# Итерация по коэффициентам alpha конкретного каскада
for current_alpha_coefficient in range(1, len(A_coef[0])):
    if current_count - current_alpha_coefficient >= 0:
        filtered_signal[current_count] -= A_coef[0][current_alpha_coefficient] * \
            filtered_signal[current_count - current_alpha_coefficient]

return np.array(filtered_signal)

def main():
    import numpy as np
    import matplotlib.pyplot as plt

    # Задаём характеристики сигнала, который будем генерировать
    A1 = 3
    A2 = 2

    f1 = 200
    f2 = 400.33

    signal_discritisation = np.linspace(0, N-1, N) * delta_T

    # Генерируем сигналы
    lambd = 1
    signal_in_bandwith = signal_generate(signal_discritisation, {A2:f2})
    signal_out_of_bandwith = signal_generate(signal_discritisation, {A1:f1})
    gaussian_noise_signal = signal_in_bandwith + lambd * np.random.normal(0, 1, N)

    # Вычисляем частоту
    k = np.linspace(0, q, q)
    frequency = k/(q * delta_T)

    # Формируем массив коэффициентов
    coefficient = []
    for frequency_number in frequency:
        coefficient.append(complex(0, 2 * np.pi * frequency_number * delta_T))

    # Расчёт теоритического рекурсивного фильтра по передаточной функции
    filter_transfer_function =
recursive_transfer_function(np.exp(np.array(coefficient)), fv=410, fn=390)

    # Генерируем фильтр
    recursive_filter, A, B = generate_recursive_filter(np.exp(np.array(coefficient)),

```

```
fv=410, fn=390)
```

```
plt.figure()  
plt.plot(signal_in_bandwidth)
```

```
# Сравниваем теоритическое АЧХ рекурсивного фильтра и АЧХ  
рассчитаного фльтра
```

```
plt.figure()  
plt.plot(frequency[:int(q/2)], 20 * np.log10(abs(filter_transfer_function))[:int(q/2)],  
label='Фильтр, полученный из аналогового прототипа')  
plt.plot(frequency[:int(q/2)], 20 * np.log10(abs(recursive_filter))[:int(q/2)],  
label='Рассчитанный рекурсивный фильтр')  
plt.title('Сравнение АЧХ фильтра')  
plt.xlabel('Частота, Гц')  
plt.ylabel('Коэффициент подавления, дБ')  
plt.xscale('log')  
plt.legend()
```

```
# Выставляем точку смещения фильтрованного сигнала и длину выборки,  
которая будет выводиться
```

```
shift_point = 220  
range_start = 0  
range_stop = 1000  
plot_range = range(range_start, range_stop)
```

```
# Фильтруем сигнал без помех и строим графики
```

```
filtered_usuall_signal =  
signal_filtration_with_recursive_filter(signal_in_bandwidth, A, B)  
compare_signals(signal_in_bandwidth, filtered_usuall_signal, plot_range,  
shift_point)
```

```
# Фильтруем сигнал с гармонической помехой и строим графики
```

```
filtered_harmonic_noise_signal =  
signal_filtration_with_recursive_filter(signal_out_of_bandwidth, A, B)  
compare_signals(signal_out_of_bandwidth, filtered_harmonic_noise_signal,  
plot_range, shift_point)
```

```
# Фильтруем сигнал с гауссовским шумом и строим графики
```

```
filtered_gaussian_noise_signal =  
signal_filtration_with_recursive_filter(gaussian_noise_signal, A, B)  
compare_signals(gaussian_noise_signal, filtered_gaussian_noise_signal,  
plot_range, shift_point)
```

```
plt.show()
```

```
if __name__ == "__main__":  
    main()
```

ПРИЛОЖЕНИЕ 3

```
#include <at89c51xd2.h>

#define sync P3_2
#define extract_first_byte(value) (value & 0xF) << 4
#define extract_second_byte(value) value >> 4
#define SAWTOOTH_LENGTH 39

unsigned int reverse(unsigned int value)
{
    unsigned int result = 0;
    if ( value & 0x200 ) result |= 0x001;
    if ( value & 0x100 ) result |= 0x002;
    if ( value & 0x080 ) result |= 0x004;
    if ( value & 0x040 ) result |= 0x008;
    if ( value & 0x020 ) result |= 0x010;
    if ( value & 0x010 ) result |= 0x020;
    if ( value & 0x008 ) result |= 0x040;
    if ( value & 0x004 ) result |= 0x080;
    if ( value & 0x002 ) result |= 0x100;
    if ( value & 0x001 ) result |= 0x200;
    return result;
}

void shift_transmit(unsigned int data_to_transmit){
    char first_byte;
    char second_byte;

    data_to_transmit = reverse(data_to_transmit);

    first_byte = extract_first_byte(data_to_transmit);
    second_byte = extract_second_byte(data_to_transmit);

    sync = 0;
    SBUF = first_byte;
    while(TI==0);
    TI = 0; TI = 0;

    SBUF = second_byte;
    while(TI==0);
    TI = 0; TI = 0;
    sync = 1;
}
```

```

}

void main (void){
    unsigned int sawtooth [SAWTOOTH_LENGTH] = {0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,
                                                    0,  0,  0,  0,  0,  0, 26,
78, 131, 183, 236, 288, 341,
                                                    393, 446, 498, 551, 603, 656, 708,
761, 813, 866, 918, 971,1023};
    int sawtooth_index;

    sync = 0;

    while(1){
        for(sawtooth_index = 0; sawtooth_index < SAWTOOTH_LENGTH;
sawtooth_index++){
            shift_transmit(sawtooth[sawtooth_index]);
        }
    }
}

```

ПРИЛОЖЕНИЕ 4

```
#include <at89c51xd2.h>
#include <stdio.h>

#define bit_12_8      P3_3
#define STS           P1_7
#define CS            P3_4
#define A0            P3_5
#define CE            P3_6
#define R_C           P3_7

void Delay(int nCount){
    while(nCount--);
}

void serial_init(void){
    SCON = 0x50;

    T2CON &= 0xF0;
    T2CON |= 0x30;
    TH2=0xFF;
    TL2=0xFD;
    RCAP2H=0xFF;
    RCAP2L=0xFD;
    TR2 = 1;
}

void serial_IT(char uart_data){
    SBUF = uart_data;
    while(TI==0);
    TI = 0;
}

float ad1674_read(void){
    R_C = 0;
    while(STS==1);
    R_C = 1; R_C = 1;
    return P0 | ((P2 & 0xF) << 8);
}

void String(char *str)
{
```



```

        int i;
        for(i = 0; str[i] != 0; i++){
            serial_IT(str[i]);
        }
    }

void main (void)
{
    float adc_data;
    char adc_char_result[7];

    serial_init();
    bit_12_8 = 1; CS = 0; A0 = 0; CE = 1;

    while(1){
        adc_data = 10*(ad1674_read()/4096);
        sprintf(adc_char_result, "%f", adc_data);
        String(adc_char_result);
        String("V +- 0.002V\n\r");
        Delay(1000000);
    }
}

```

ПРИЛОЖЕНИЕ 5

```
#include <at89c51xd2.h>

#define bit_12_8 P3_3
#define STS      P1_7
#define CS       P3_4
#define A0       P3_5
#define CE       P3_6
#define R_C      P3_7
#define sync     P3_2

#define extract_first_byte(value) (value & 0xF) << 4
#define extract_second_byte(value) value >> 4

unsigned int reverse(unsigned int value)
{
    unsigned int result = 0;
    if ( value & 0x200 ) result |= 0x001;
    if ( value & 0x100 ) result |= 0x002;
    if ( value & 0x080 ) result |= 0x004;
    if ( value & 0x040 ) result |= 0x008;
    if ( value & 0x020 ) result |= 0x010;
    if ( value & 0x010 ) result |= 0x020;
    if ( value & 0x008 ) result |= 0x040;
    if ( value & 0x004 ) result |= 0x080;
    if ( value & 0x002 ) result |= 0x100;
    if ( value & 0x001 ) result |= 0x200;
    return result;
}

void shift_transmit(unsigned int data_to_transmit){
    char first_byte;
    char second_byte;

    data_to_transmit = reverse(data_to_transmit);

    first_byte = extract_first_byte(data_to_transmit);
    second_byte = extract_second_byte(data_to_transmit);

    sync = 0;
    SBUF = first_byte;
    while(TI==0);
}
```

```

    TI = 0; TI = 0;

    SBUF = second_byte;
    while(TI==0);
    TI = 0; TI = 0;
    sync = 1;
}

void Delay(int nCount){
    while(nCount--);
}

unsigned int ad1674_read(void){
    R_C = 0;
    while(STS==1);
    R_C = 1; R_C = 1;
    return P0 | ((P2 & 0xF) << 8);
}

void main (void)
{
    unsigned int adc_data;

    bit_12_8 = 1; CS = 0; A0 = 0; CE = 1;

    while(1){
        adc_data = ad1674_read() >> 2;
        shift_transmit(adc_data);
    }
}

```