Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись инициалы, фамилия
«__»_____2018 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 – «Информатика и вычислительная техника»
код и наименование специальности

Разработка программного обеспечения геофизического прибора для поиска минеральных ресурсов на основе извлечения информации из пассивных шумовых полей земли
тема

Пояснительная записка

Руководитель _____ _____ Д.А. Швец
подпись, дата должность, учёная степень инициалы, фамилия

Выпускник _____ Д.Н. Исаев
подпись, дата инициалы, фамилия

Нормоконтролер _____ В.И. Иванов
подпись, дата инициалы, фамилия

Красноярск 2018

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись инициалы, фамилия
«__»_____2018 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы
бакалаврской работы, дипломного проекта, дипломной работы, магистерской диссертации

Студенту Исаеву Дмитрию Николаевичу
фамилия, имя, отчество

Группа КИ12-10БГЭК Направление (специальность) 09.03.01
номер код

«Информатика и вычислительная техника»
наименование

Тема выпускной квалификационной работы Разработка программного обеспечения для геофизического прибора на основе извлечения информации из пассивных шумовых полей земли

Утверждена приказом по университету № 5758 от 04.05.2016

Руководитель ВКР Д.А. Швец, к.т.н, кафедра вычислительной техники
инициалы, фамилия, должность, учебное звание и место работы

Исходные данные для ВКР: задание на ВКР

Перечень разделов для ВКР: Анализ предметной области и постановка задачи; Выбор микроконтроллера для реализации задачи; Разработка программного обеспечения.

Перечень графического материала: Блок-схема алгоритма работы прибора, Блок-схема программы микроконтроллера

Руководитель ВКР _____ Д.А. Швец
подпись инициалы и фамилия

Задание принял к исполнению Д.Н. Исаев
подпись, инициалы и фамилия студента

«____» _____ 2018 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современная электроразведка решает широкий круг геологических задач и объединяет группу методов прикладной геофизики, основанных на наблюдении и изучении особенностей распределения характеристик электромагнитных полей естественного или искусственного происхождения, обусловленных дифференциацией горных пород и руд по физическим свойствам (удельной электропроводности, диэлектрической проницаемости, магнитной проницаемости).

При поиске полиметаллических руд наибольшую эффективность дает метод вызванной поляризации (ВП), заключающийся в возбуждении геологического разреза импульсным или гармоническим током с помощью заземленной питающей линии и регистрации переходной или фазовой характеристик электрического поля, получаемых с приемной линии [1].

Разработка программного обеспечения для геофизического прибора, решающего задачи электроразведки, требует комплексного подхода, так как включает в себя сразу несколько блоков, подлежащих реализации. Как правило основные блоки системы включают в себя аналоговую обработку и последующую цифровую обработку полученных данных.

Преобразование аналоговых сигналов, осуществляется при помощи аналого-цифрового преобразователя, в рамках прибора, реализованного на базе микроконтроллера.

Сегодня микроконтроллеры имеют широчайшее распространение в массе приборов различных сфер, начиная от бытовой и заканчивая военной.

Целью работы является разработка программного обеспечения для геофизического прибора, основанного на методе получения информации из пассивных шумовых полей земли.

1 Анализ предметной области и постановка задачи

1.1 Анализ предметной области

Физической основой метода ВП является регистрация параметров двойного электрического слоя на границе раздела электронно-проводящего рудного тела и ионопроводящей среды. Проходящий через среду поляризующий ток наблюдается на поверхности земли в виде электрического потенциала разряда указанной границы.

В практике электроразведочных работ метод вызванной поляризации реализуется в виде прокладки питающей линии, заземленной по концам и питаемой импульсным или гармоническим током, с мощностью источника от 100 до 10000 Вт. Прокладка такой линии через горно-таежный ландшафт и завоз на место работ достаточно мощного генератора, требуют использования тяжелой техники и занимает много времени и рабочего персонала, в составе не менее пяти человек.

В данной работе геофизический прибор использует в качестве источника возбуждения георазреза естественные электромагнитные поля Земли (ЕЭМПЗ), что исключает из схемы измерения мощные источники возбуждающего тока и длинную питающую линию. Это дает существенное удешевление ведения электроразведочных работ. Затраты на разработку программного обеспечения считаются несущественными, относительно общих затрат на проведение разведочных работ [2].

1.1.1 Физические основы метода вызванной поляризации

Метод вызванной поляризации основан на регистрации электрических полей, вызванных электрохимическими и электрокинетическими процессами, возникающими под действием электрического тока, возбуждаемого

сторонними источниками в двойных электрических слоях, на границе раздела электронный проводник-ионопроводящая среда [3]. Эффект наблюдается в постоянном или низкочастотном электрическом поле (0-100 Гц) только в неоднородных (гетерогенных) средах, состоящих из твердого, жидкого или газообразного вещества. Чем выше степень неоднородности горных пород, тем, как правило, выше уровень поляризации.

Для изучения вызванной поляризации используют четырехэлектродные установки AMNB, изображенные на рисунке 1, с помощью которых измеряют параметр кажущейся поляризуемости η_k . Параметр η_k это отношение напряженности поля вызванной поляризации к напряженности первичного поля и измеряется, как правило, в процентах. Так как в процессе измерений η_k регистрируется первичное электрическое поле, то по результатам наблюдений определяют кажущееся сопротивление ρ_k . Если измерения ведутся на переменном токе, то потенциал ВП можно измерять в градусах фазового сдвига измеренного гармонического сигнала $E_{min} \cdot \sin(\omega \cdot t + \phi_{ВП})$ относительно токового сигнала $I_{ген} \cdot \sin(\omega \cdot t)$, определяя, таким образом, задержку в реакции среды относительно процесса воздействия внешнего поля [2].

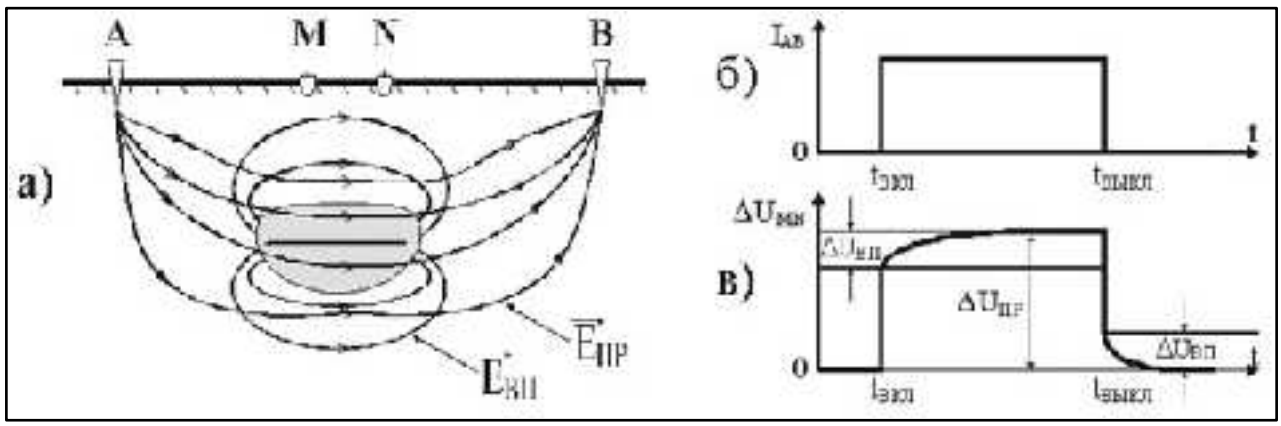


Рисунок 1 – Измерение вызванной поляризации

Наибольшими значениями поляризуемости обладают породы и руды с включениями электронопроводящих минералов пирита, халькопирита, галенита, графита, антрацита, самородных металлов (медь самородная, ртуть, серебро) и т.д. Поляризуемость таких пород бывает, как правило, очень высокой от 5 до 40% и более. Различают два типа поляризуемости электронных проводников: поверхностную и объемную. Поверхностная поляризуемость характерна для сплошных рудных тел большого объема, у которых на поверхности под действием протекающего тока накапливаются вторичные заряды. При выключении тока в течение нескольких секунд происходит релаксация накопленной энергии. Для вкрапленных руд, которые не обнаруживаются другими методами геофизики, характерна объемная поляризация. Для поиска таких объектов метод вызванной поляризации является ведущим методом исследований. Безрудные породы (осадочные и магматические) поляризуются значительно слабее: их поляризуемость меняется от 0.5 до 3%.

Кроме того, метод ВП также применяется в гидрогеологической геофизике для определения уровня грунтовых вод и выявления различных литологических комплексов пород в разрезе, благодаря заметной зависимости

 осадочных пород от влажности, глинистости, пористости.

Таким образом, основные области применения метода вызванной поляризации: поиски рудных залежей, гидрогеология и геологическое картирование. Следует отметить, что в последнее время метод находит все большее применение при поисках нефтегазовых месторождений. Как показали многочисленные полевые эксперименты, над углеводородными залежами образуются скопления вкрапленников пирита, что создает физические предпосылки для поисков и оконтуривания месторождений нефти и газа [2].

1.1.2 Языки программирования микроконтроллеров

Создание любой программы для микроконтроллера напрямую связано с выбором подходящего языка программирования. Языки различаются по своему составу и применению, а также подразделяются на языки высокого и низкого уровней.

Языки низкого уровня такие как Ассемблер близки к машинному коду, но не являются им. Каждому оператору соответствует не более одной машинной команды, что приводит к громоздкости и неудобству для понимания человеком. Однако, программа занимает малый объем памяти [5].

К языкам программирования микроконтроллеров высокого уровня относятся: PL/M, Pascal, C/C++. При работе с языками программирования микроконтроллеров высокого уровня происходит увеличение эффективности программирования за счет замены одного оператора несколькими машинными командами. Языки программирования микроконтроллеров высокого уровня требуют больших затрат памяти, так как объем такой программы достаточно большой. Преимуществом их использования можно назвать возможность работы программы на различных микропроцессорах, при использовании программ-трансляторов.

Наибольшее распространение получил язык C/C++. Его популярность обоснована рядом причин. Стандартный язык имеет возможность преобразования исходного кода для нужного в работе микроконтроллера. Для этого необходимо учитывать архитектуру микроконтроллера выбранного типа и использовать компилятор. Также к преимуществам над другими языками программирования микроконтроллеров можно отнести факт наличия в C/C++ множества программных средств и библиотек [6].

1.2 Постановка задачи

В рамках разработки программного обеспечения для геофизического прибора на базе микроконтроллера необходимо реализовать следующие пункты:

– выбор микроконтроллера, в соответствии заданным параметрам;
– выбор подходящей среды разработки;
– написание программного кода на языке C++.

Выбранный микроконтроллер должен отвечать следующим требованиям:
– ядро Cortex-M3 архитектуры ARM;
– иметь двенадцати разрядный аналого-цифровой преобразователь;
– серия микроконтроллера STM32;
– наличие интерфейсов микроконтроллеров USART, SPI;
– наличие таймеров прерывания;
– не менее десяти портов GPIO;
– flash память не менее 1024 килобайт;
– тактовая частота 72МГц;
– доступность и цена не выше чем 5$.

2 Выбор микроконтроллера для реализации задачи

2.1 Структура системы геофизического прибора

Общая структурная схема прибора изображена на рисунке 2.

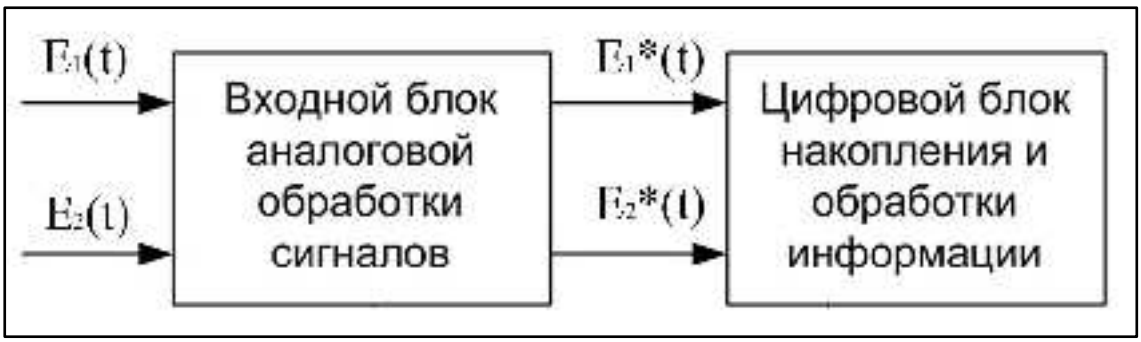


Рисунок 2 – Общая структурная схема прибора

Входное устройство должно выполнять следующие функции:

- усиление входного сигнала с чувствительностью не менее 1 мкВ;
- полосовую фильтрацию в диапазоне частот от 0.1 до 20 Гц;
- подавление импульсных и промышленных помех;
- максимальную автоматизацию вычислительных операций;
- входное сопротивление не менее 3 МОм.

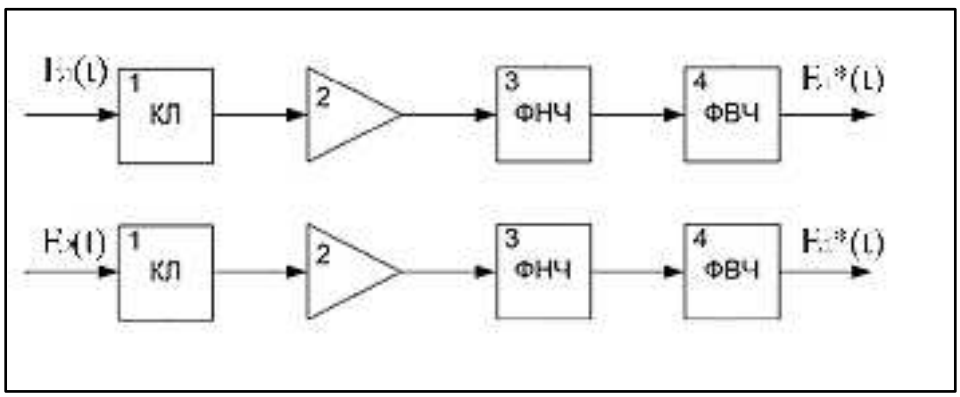
Для реализации вышеизложенных функций входной блок разделяется на три узла: малошумящий усилитель, полосовой фильтр (последовательно включённые ФНЧ и ФВЧ), узел подавления импульсных помех и цифровая система обработки.

Малошумящий усилитель разрабатывается на основе современных микросхем инструментальных усилителей. Они обладают следующими параметрами, такими как, низкий уровень собственных шумов, высокое входное сопротивление, малые токи питания и напряжение смещения, низкое выходное напряжение смещения с возможностью его корректировки, минимальное количество внешних элементов и другими.

Фильтры нижних частот проектируются на основе микросхем-фильтров на переключаемых конденсаторах. Современные микросхемы-фильтры, так же, как и инструментальные, обладают высокими техническими характеристиками и широким ассортиментом. При правильном выборе они вполне подходят для применения в данном устройстве. Большим преимуществом данных фильтров является тот факт, что имеется возможность программным образом управлять частотой среза.

Узел подавления импульсных помех представляет собой размыкающий ключ, срабатывающий по определённому пороговому напряжению сигнала. При необходимости перестройки порога срабатывания могут применяться микросхемы аналоговых ключей с цифровым управлением.

На основании вышеизложенных данных разработана следующая функциональная схема входного блока аналоговой обработки, изображенная на рисунке 3.



1 – Узел подавления импульсных помех;

2 – Узел усилителя;

3 – Фильтр нижних частот;

4 – Фильтр верхних частот.

Рисунок 3 – Функциональная схема входного блока аналоговой обработки сигнала

Блок цифровой обработки информации обеспечивает выполнение следующих функций:

– аналого-цифровое преобразование сигналов в двух каналах одновременно;

– фильтрацию сигналов;

– оперативное запоминание данных реализаций в памяти устройства;

– обработка реализаций на основе различных алгоритмов обнаружения;

– автоматическую компенсацию уровней сигналов в каналах вариаций коэффициента k;

– выдачу информации на цифровой дисплей.

Наличие огромного выбора различных устройств на базе микроконтроллеров позволяет использовать их в основном узле блока цифровой обработки информации.

Для передачи данных на персональный компьютер используются стандартные микросхемы интерфейсов RS232/USB.

Функциональная схема цифрового блока накопления и обработки информации изображена на рисунке 4.

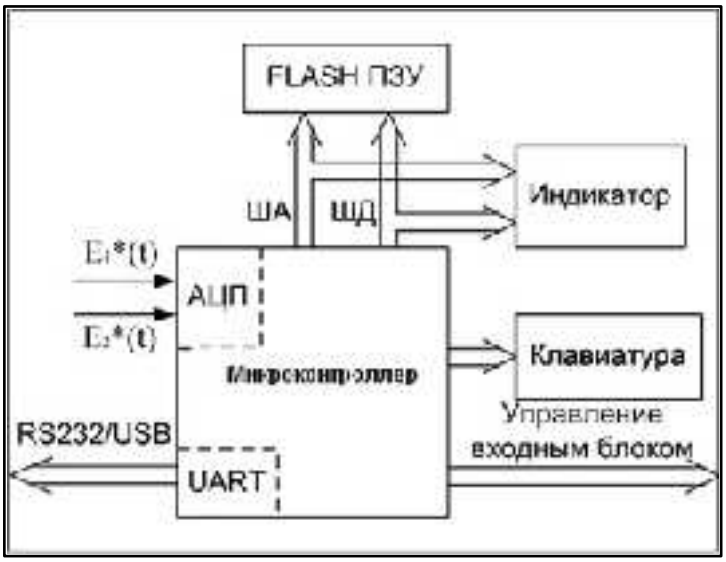


Рисунок 4 – Функциональная схема цифрового блока накопления и обработки информации

Окончательная структурная схема устройства изображена на рисунке 5.

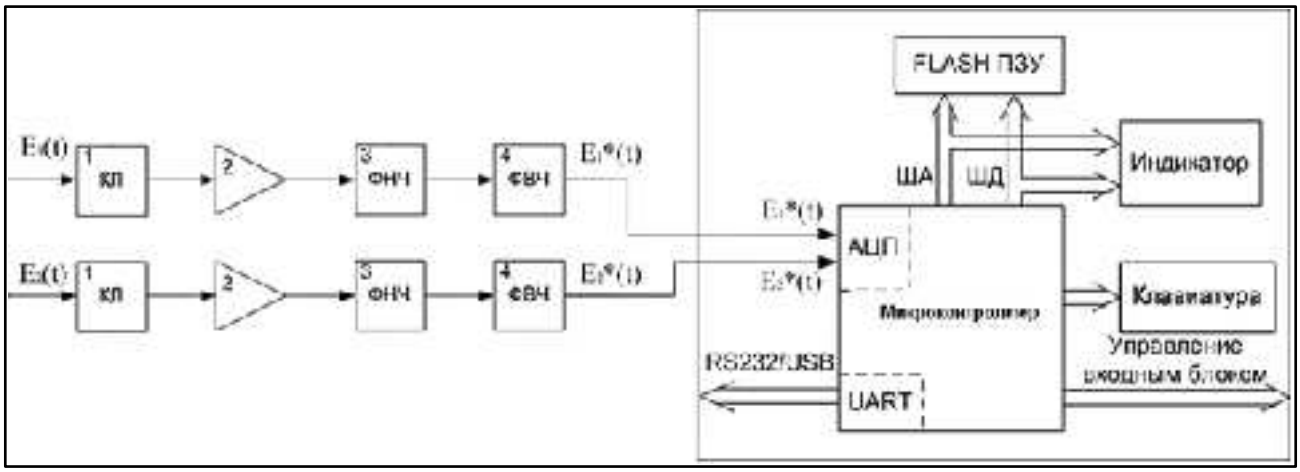


Рисунок 5 – Структурная схема устройства

Аналого-цифровое преобразование сигналов в двух каналах организуется путем мультиплексирования двух каналов на один АЦП [4]. При тактировании микроконтроллера частотой 1 МГц и частоте дискретизации 256 Гц задержка между выборками в разных каналах будет составлять не более 50 мкс. Следовательно, для частот от 1 Гц до 20 Гц разность в каналах будет составлять не более 0.1%.

Цифровой фильтр рассчитывается при помощи пакета программ Matlab. Используется фильтр с конечной импульсной характеристикой. Это обусловлено наличием в микроконтроллере периферийного 64 разрядного аппаратного умножителя с накоплением, что позволяет производить фильтрацию с меньшей затратой системных ресурсов, чем при использовании фильтров с бесконечной импульсной характеристикой.

Алгоритмическая схема работы прибора приводиться в приложении А. Данная программа работает следующим образом: после включения прибора микроконтроллер переходит в режим ожидания замера. Кнопкой начала замера запускается АЦП и происходит преобразование сигналов в цифровой код с частотой дискретизации 256 Гц. Далее, в режиме реального времени, выборки сигнала поступают в цифровой фильтр, а затем в ОЗУ. После этих операций

вычисляется параметр η, соответствующий параметру вызванной поляризации рудного тела. Подобным образом производиться оптимизация η по параметру κ.

Данный параметр вводиться для того, чтобы выровнять сигналы в каналах по амплитуде, поскольку кажущееся сопротивление для разных каналов может отличаться в несколько раз. Автоматическая оптимизация вычислительного алгоритма оценки параметра η производиться методом Пауэла [1].

После выполнения функции оптимизации, текущие значения параметров η и κ сохраняются во Flash память и выводятся на LCD дисплей.

2.2 Интерфейсы драйверов микроконтроллера

Интерфейсы микроконтроллера служат для связи микроконтроллера с другими устройствами. Они определяют набор характеристик, правил, и методов для успешного обмена данными между устройствами, например, между микроконтроллером и персональным компьютером. В зависимости от серии микроконтроллера существуют различные типы интерфейсов, подходящие под определенный вид задач. В данной работе, применяются последовательные интерфейсы USART\UART и SPI.

USART\UART – универсальный синхронный \ асинхронный приемопередатчик, удобный для создания канала информационного обмена между микроконтроллером и внешним миром. Как следует из названия, интерфейс поддерживает дуплексный режим работы, так как может использоваться для одновременного приема и передачи данных. Кроме этого, интерфейс поддерживает протокол RS-232, для связи с персональным с персональным компьютером.

При передаче UART преобразует параллельный код в последовательный и передает его побитно в линию, обрамляя исходную последовательность битами старта, останова и контроля. При приеме данных UART преобразует

последовательный код в параллельный. Непременным условием правильной передачи или приема является одинаковая скорость работы приемного и передающего UART, что обеспечивается стабильной частотой кварцевого резонатора [7].

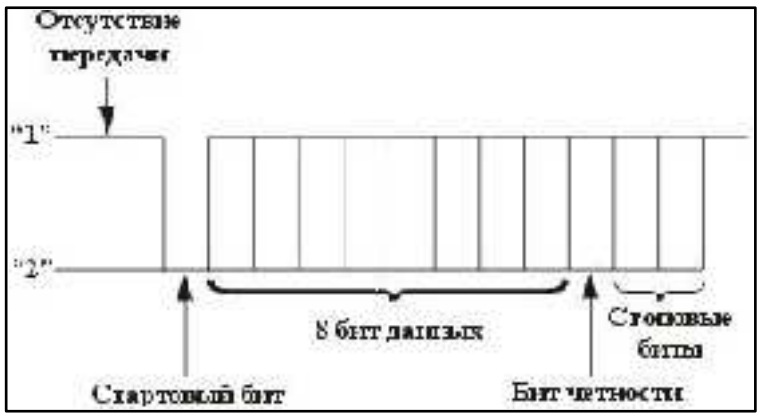


Рисунок 6 – Формат данных интерфейса UART

SPI – последовательный периферийный трехпроводный интерфейс, предназначенный для организации обмена данными между двумя устройствами. С его помощью может осуществляться обмен данными между микроконтроллером и различными устройствами, такими, как цифровые потенциометры, АЦП, FLASH-ПЗУ, или между несколькими микроконтроллерами. Кроме того, через интерфейс SPI может осуществляться программирование микроконтроллера. Шина SPI организована по принципу “ведущий-подчиненный”. В качестве ведущего шины обычно выступает микроконтроллер. Подключенные к ведущему шины внешние устройства образуют подчиненных шины.

Главным составным блоком интерфейса SPI является обычный сдвиговый регистр, сигналы синхронизации и ввода/вывода битового потока которого и образуют интерфейсные сигналы. SPI выступает в роли протокола обмена данными между двумя сдвиговыми регистрами, каждый из которых одновременно выполняет и функцию приемника, и функцию передатчика. Непременным условием передачи данных по шине SPI является генерация

сигнала синхронизации шины. Этот сигнал имеет право генерировать только ведущий шины и от этого сигнала полностью зависит работа подчиненного шины.

Протокол передачи по интерфейсу SPI заключается в выполнении операции сдвига и, соответственно, побитного ввода и вывода данных по определенным фронтам сигнала синхронизации. Установка данных при передаче, и выборка при приеме всегда выполняются по противоположным фронтам синхронизации. Это необходимо для гарантирования выборки данных после надежного их установления. Если к этому учесть, что в качестве первого фронта в цикле передачи может выступать нарастающий или падающий фронт, то всего возможно четыре варианта логики работы интерфейса SPI. Эти варианты получили название режимов SPI и описываются двумя параметрами: CPOL – исходный уровень сигнала синхронизации, его уровень определяет первый и последний фронт передачи, и CPHA – фаза синхронизации от этого параметра зависит, в какой последовательности выполняется установка и выборка данных [8].

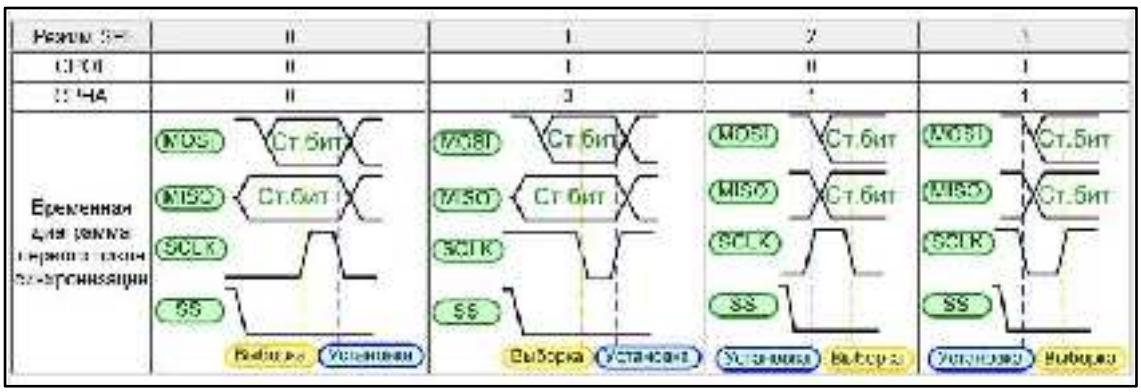


Рисунок 7 – Режимы работы интерфейса SPI

2.3 Аналого-цифровой преобразователь микроконтроллеров STM32

В основе данной работы лежит использование аналого-цифрового преобразователя (АЦП) для преобразования входной физической величины в ее числовое представление. Входная величина может быть любой, например, силой тока, напряжением, емкостью и другими величинами. Само преобразованием заключается в сравнении этой входной величины с некой опорной величиной, как правило, с опорным напряжением.

АЦП обладает рядом характеристик важнейшими из которых можно назвать частоту преобразования или частоту дискретизации и разрядность. Частота преобразования обычно измеряется в отсчетах в секунду, разрядность – в битах. Чем выше скорость преобразования и разрядность, тем труднее добиться требуемых характеристик, это влияет на цену и сложность изготовления аналого-цифрового преобразователя.

В микроконтроллерах семейства STM32 АЦП обладают различными параметрами в зависимости от серии микроконтроллера. В таблице 1 представлены основные параметры АЦП в сериях микроконтроллеров [9]. Параметры аналого-цифрового преобразователя необходимо учитывать при выборе микроконтроллера, а данные серии полностью удовлетворяют требованиям технического задания.

Серия МК	Количество АЦП в одном МК	Разрядность, бит	Частота дискретизации, Мвыб/сек	Количество входных каналов	Время преобразования, мкс	Погрешность преобразования, МЗР
STM32F100	1	12	1	16	1,17…21	2…5
STM32F101	1				1…18	
STM32F102	1				1,2…18	
STM32F103	2…3			16…21	1…18	
STM32F105	2			16	1…18	
STM32F2xx	3		2	24	0,5…16,4	
STM32L	1		1	24	1…25	2…4
STM32W	1		0,1	9	5	–

Таблица 1 – Параметры АЦП в различных семействах STM32

2.4 Выбор микроконтроллера

На сегодняшний день, компания STMicroelectronics занимает одну из лидирующих позиций среди производителей микроконтроллеров на ядре ARM Cortex-M3. Семейство STM32 включает в себя более десяти серий микроконтроллеров, предназначенных для различных целей и задач, включая микроконтроллеры с высокой производительностью, недорогие микроконтроллеры, микроконтроллеры с низким энергопотреблением и другие. База доступных микроконтроллеров, расположенная на официальном сайте компании производителя, насчитывает более тысячи различных моделей.

Как следует из названия, отличительной особенностью микроконтроллеров семейства STM32 является их 32-разрядность, что напрямую влияет на производительность микроконтроллера. Кроме этого, семейству свойственно низкое энергопотребление, а также возможность работы в режимах энергосбережения. Процессор Cortex-M3 работает с системой команд Thumb-2, которая в сочетании с функцией хранения невыровненных данных и побитовой обработки, обеспечивает 32-разрядную производительность при эквивалентной стоимости современных восьми и шестнадцатиразрядных микроконтроллеров [10].

Семейство STM32 включает в себя две крупные линейки микроконтроллеров Access и Performance. Серия Access (F101xx) разработана с целью внедрения 32-разрядной схемотехники в критичные к стоимости применения или шестнадцати разрядные проекты. Более современная линейка Performance (F103xx) обладает почти вдвое превышающими параметрами и ориентируется на повышенную производительность обработки.

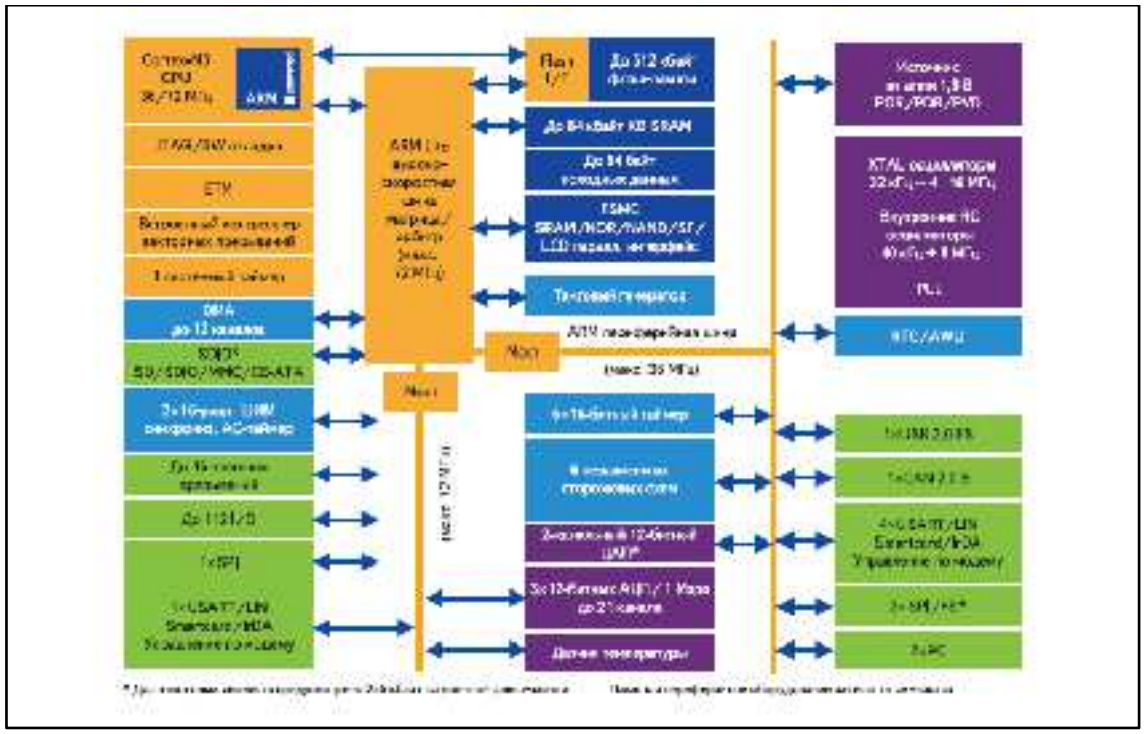


Рисунок 8 – Архитектура микроконтроллера семейства STM32

Исходя из изложенных ранее требований к микроконтроллеру, и обратившись к базе микроконтроллеров на сайте компании производителя, произведен выбор в пользу модели микроконтроллера STM32F103RGTx линейки Performance. Микроконтроллер обладает следующими параметрами, полностью удовлетворяющими поставленные требования [11]:

– ядро: ARM Cortex M3;
– тактовая частота: до 72МГц;
– FLASH-память: 1024 кб;
– RAM-память: 96 кб;
– интерфейсы: SPI, I2C, USART, USB 2.0, CAN;
– АЦП: до двух 12 бит/16 каналов;
– 14 таймеров прерывания;
– Напряжение питания: 2…3,6В.

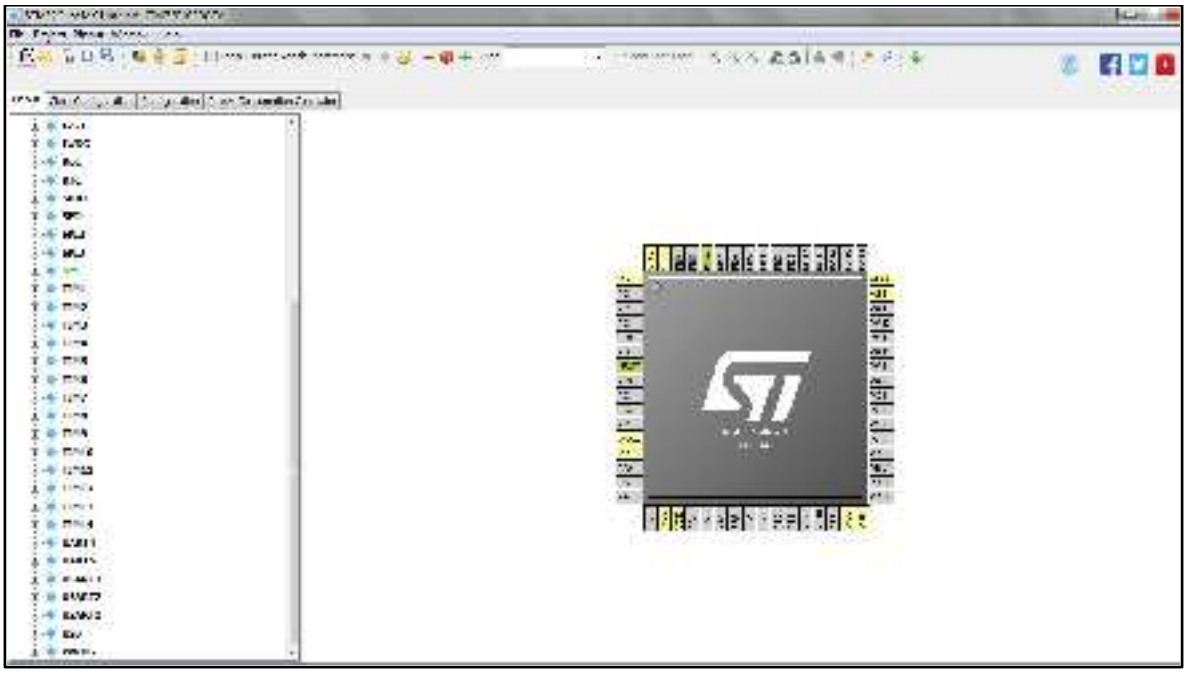


Рисунок 9 – Микроконтроллер STM32F103RGTx в среде STM32CubeMX

3 Разработка программного обеспечения

3.1 Среда разработки программного обеспечения

Для целей разработки программного обеспечения выбрана программная среда Keil uVision, обладающая набором утилит и средств для разработки программ, включающих C++ компиляторы, макроассемблеры, отладчики, симуляторы, линкеры, IDE-приложения и оценочные платы для различных семейств микроконтроллеров, таким образом полностью подходящая для написания программного обеспечения для микроконтроллера. Среда разработки представлена на английском языке и функционирует только на операционный системах Windows серий: 2000; Vista; 7 [12].

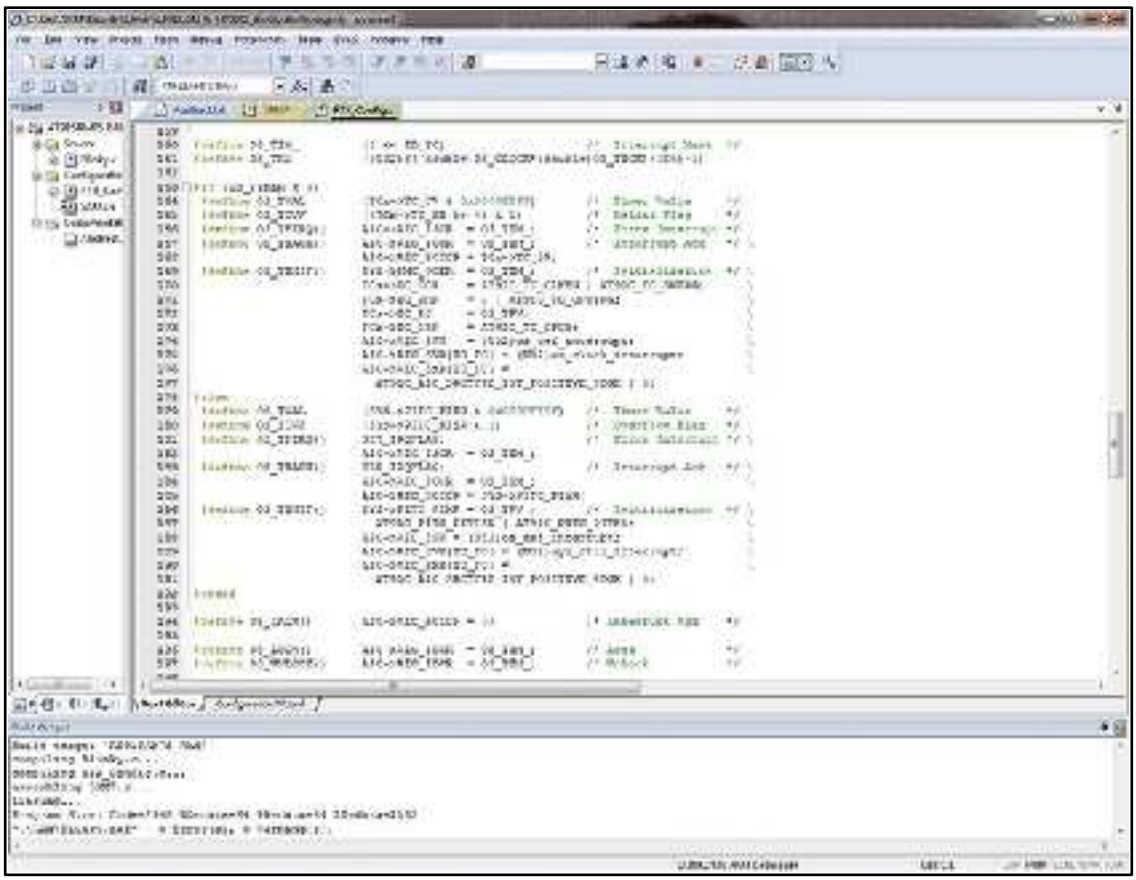


Рисунок 10 – программная среда Keil uVision

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. Ниже приведено описание основных программных средств Keil uVision.

База данных микроконтроллеров, содержит подробную информацию обо всех поддерживаемых устройствах. Здесь хранятся их конфигурационные данные и ссылки на источники информации с дополнительными техническими описаниями. При добавлении нового устройства в проект все его уникальные опции устанавливаются автоматически.

Менеджер проектов, служащий для объединения отдельных текстов программных модулей и файлов в группы, обрабатываемые по единым правилам. Подобная группировка позволяет намного лучше ориентироваться среди множества файлов.

Встроенный редактор, облегчающий работу с исходным текстом за счет использования многооконного интерфейса, выделения синтаксических элементов шрифтом и цветом. Существует опция настройки в соответствии со вкусами разработчика. Редактирование остается доступным и во время отладки программы, что позволяет сразу исправлять ошибки или отмечать проблемные участки кода.

Средства автоматической компиляции, ассемблирования и компоновки проекта, которые предназначены для создания исполняемого (загрузочного) модуля программы. При этом между файлами автоматически генерируются новые ассемблерные и компиляторные связи, которые в дальнейшем позволяют обрабатывать только те файлы, в которых произошли изменения или файлы, находящиеся в зависимости от изменённых. Функция глобальной оптимизации

проекта позволяет достичь наилучшего использования регистров микроконтроллера путем неоднократной компиляции исходного кода. Компиляторы uVision работают с текстами, написанными на C++ или ассемблере для контроллеров семейств ARM, MSC51, C166 и многих других. Кроме того возможно использование компиляторов других производителей.

Отладчик-симулятор, отлаживающий работу скомпилированной программы на виртуальной модели микропроцессора. Довольно достоверно моделируется работа ядра контроллера и его периферийного оборудования: портов ввода-вывода, таймеров, контроллеров прерываний. Для облегчения комплексной отладки разрабатываемого программного обеспечения возможно подключение программных моделей нестандартного оборудования.

Дополнительные утилиты, облегчающие выполнение наиболее распространенных задач. Число и набор меняется от версии к версии. Выделяют следующие из них [13]:

– Source Browser – содержит базу данных программных символов для быстрого поиска;

– Find in Files – предназначен для поиска заданного кода во всех файлах указанной папки или проекта;

– Tools Menu – позволяет использовать утилиты сторонних производителей;

– PC-Lint – анализирует исходный текст программы с выделением потенциально опасных мест;

– Flash tool – программирует FLASH-память микроконтроллеров.

3.2 Программное обеспечение STM32CubeMX

Программное обеспечение, для разработки программ для микроконтроллеров STM32, созданное фирмой STMicroelectronics, служит для

упрощения процесса разработки, за счет применения драйверов высокого уровня и удобной графической среды.

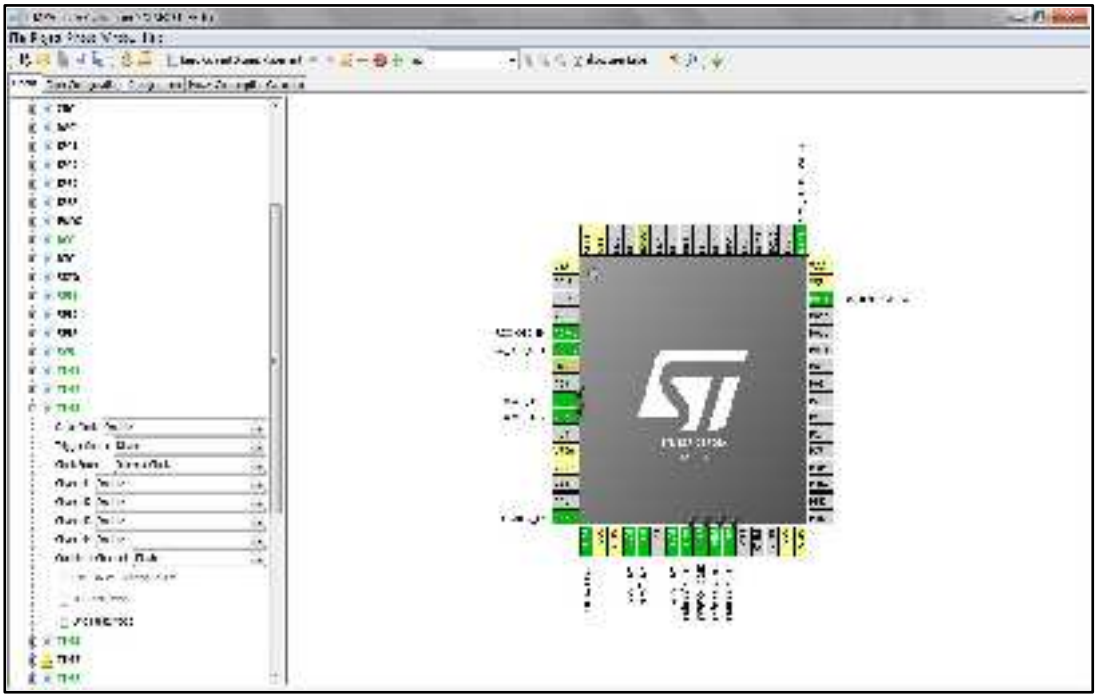


Рисунок 11 – программная среда STM32CubeMX

Данное программное обеспечения включает в себя:

– STM32CubeMX – утилита с графическим интерфейсом, предназначенная для генерации кода инициализации STM32 и встроенной в микроконтроллер периферии;

– Комплексное программное обеспечение, для встраиваемых платформ, сконфигурированное для конкретной серии микроконтроллеров;

– STM32Cube HAL, встраиваемое программное обеспечение уровня абстракции HAL, для STM32, обеспечивающее максимальную переносимость кода внутри семейства STM32;

– Набор встраиваемых компонентов ПО, таких как RTOS, USB, FatFS, TCP/IP, Graphics, настроенный для их совместной работы;

– Полный комплект примеров, для всех программых компонентов и утилит, находящихся в наборе ПО.

Драйверы HAL содержат полный набор готовых к использованию библиотек, которые делают проще реализацию пользовательского приложения. В их числе ряд готовых функций, констант, классов и процедур. Библиотеки драйверов обеспечивают общие для всех серий микроконтроллеров функции, а так же содержат в себе более специфические, индивидуальные функции для конкретных семейств. Исходный код библиотеки драйверов разработан в соответствии с рекомендациями Strict ANSIC, что делает код независимым от инструментов разработки. Весь исходный код проверен с помощью инструмента статистического анализа CodeSonarTM, и полностью документирован. Драйверы HAL реализуют обнаружение ошибок во время выполнения программы, так как HAL проверяет входные значения всех функций. Такая динамическая проверка способствует повышению надежности встроенного ПО. Обнаружение ошибок во время выполнения программы, также, способствует ускорению разработки пользовательских приложений и процесса отладки [14].

3.3 Описание программного кода

Программа обеспечивает работу микроконтроллера, преобразуя аналоговый сигнал в цифровой и передавая его на персональный компьютер. Общий принцип работы заключается в ожидании на прием 1 байта информации – сигнала к началу работы. После на портах микроконтроллера выставляются частоты цифровых фильтров и частота дискретизации. Происходит считывание данных с аналоговой части прибора, преобразование сигнала и отсылка преобразованного сигнала через Comport на персональный компьютер для дальнейшей обработки.

Драйверы HAL и заголовочные файлы библиотек, используемых в данном программном обеспечении подключаются при помощи раздела #include до описания основной части main.

```
/* USER CODE BEGIN Includes */
#include "Core.h"
#include <string.h>
#include "stm32f1xx_hal.h"
```

Рисунок 12 – Заголовочные файлы

<String.h> является стандартным заголовочным файлом для языков Си, “Core.h” и “stm32f1xx_hal.h” содержат в себе драйверы микроконтроллера и библиотеку HAL.

Раздел #define создает макросы для идентификации строки. Эти порты необходимы для организации передачи через последовательный периферийный интерфейс – SPI.

```
#define NSS_Pin GPIO_PIN_4
#define NSS_GPIO_Port GPIOA
#define SCK_Pin GPIO_PIN_5
#define SCK_GPIO_Port GPIOA
#define MOSI_Pin GPIO_PIN_7
#define MOSI_GPIO_Port GPIOA
```

Рисунок 13 – Макросы

В этом разделе описываются структуры для всех элементов системы: АЦП, SPI, UART и таймеры

```
/* Private variables ---------
ADC_HandleTypeDef hadc1;

SPI_HandleTypeDef hspi1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim7;
TIM_HandleTypeDef htim12;

UART_HandleTypeDef huart2;
```

Рисунок 14 – Структуры

```
/* ADC1 init function */
static void MX_ADC1_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Common config
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = ENABLE;
  hadc1.Init.NbrOfDiscConversion = 1;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 2;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Regular Channel
    */
  sConfig.Channel = ADC_CHANNEL_11;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  /**Configure Regular Channel
  */
sConfig.Channel = ADC_CHANNEL_12;
sConfig.Rank = 2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}
```

Рисунок 15 – Инициализация АЦП

Так как прибор принимает данные с двух каналов, то необходимо инициализировать АЦП, который мог бы обрабатывать два сигнала. Эти сигнала будут поступать на ADC_CHANNEL 11 и ADC_CHANNEL 12. Для АЦП используются порты PC1 и PC2.

Для передачи данных используется последовательный периферийный интерфейс – SPI. Интерфейс настраивается на двунаправленную связь с сигналом синхронизации по низкому фронту. При передаче первым отправляется старший бит.

```
/* SPI1 init function */
static void MX_SPI1_Init(void)
{

  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_HARD_INPUT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_4;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
```

Рисунок 16 – Инициализация SPI

Настройка портов PC4, PC5, PB0 и PB1 на выход, для этого устанавливается тактирование портов, режим и скорость работы. Режим OUTPUT_PP позволяет выдавать на порт как логический ноль так и логическую единицу.

```
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1, GPIO_PIN_RESET);

  /*Configure GPIO pins : PC4 PC5 */
  GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PB0 PB1 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}
```

Рисунок 17 – Настройка портов

Для приема и передачи данных на персональный компьютер используется универсальный асинхронный приемопередатчик – UART. Передатчик использует зарезервированные порты PA2 и PA3.

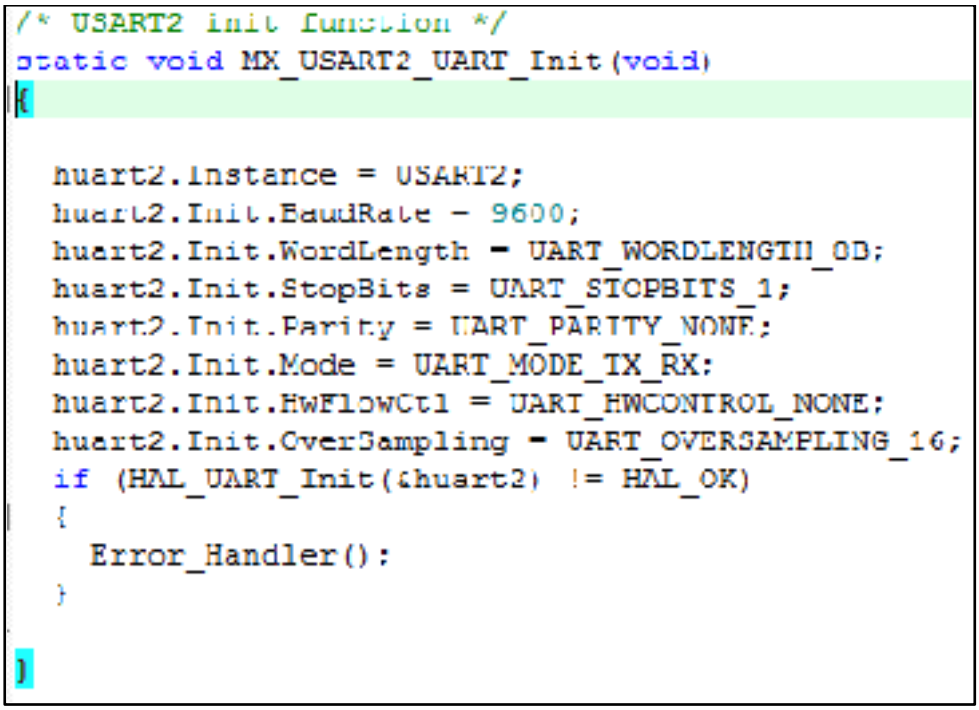


Рисунок 18 – Инициализация UART

В программе используется ряд таймеров, инициализация которых идентична, за исключением параметров пределителя и периода. Таймеры Tim1, Tim2, Tim3, Tim4, Tim5, Tim6, Tim7, Tim12. Tim 4,12 и 6 используются, для прерываний, при получении на вход портов PINB_0, PINB_1, PIC_5 частот срезов фильтров и частоты дискретизации.

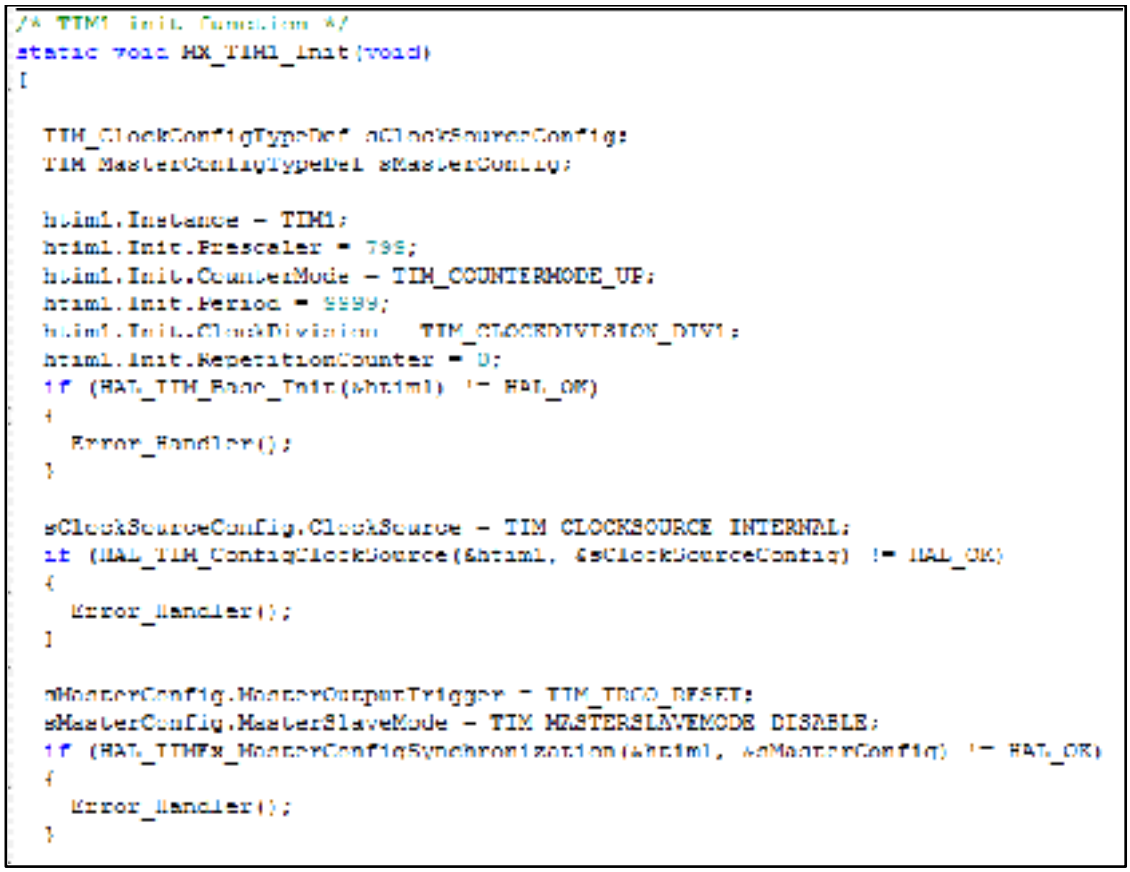


Рисунок 19 – Инициализация таймеров TIM

Для приема данных по UART используется функция прерывания. Данные записываются в переменную Rx_data[].

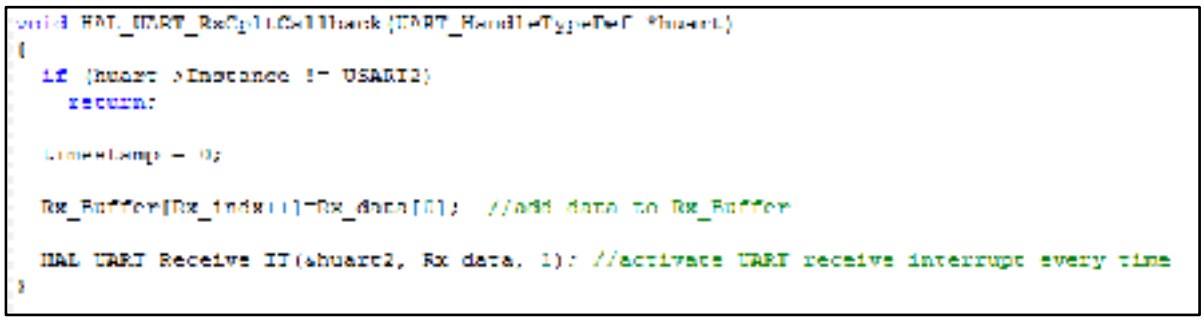


Рисунок 20 – Прерывание для UART

Функция выполняет следующий набор команд, с использованием SPI передачи. По SPI передается команда varCmd означающая готовность интерфейса к передаче данных. После передается один байт данных хранящихся в переменной data[]. Процедура повторяется снова, с отправки команды varCmd и последуюшей отправки одного байта данных.

```
void CoreSpiSend(uint8_t data[])
{
    uint8_t varCmd = 0x10;

    HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(NSS_GPIO_Port, NSS_Pin, GPIO_PIN_RESET);
    HAL_Delay(10);

    for(int i=1; i >=0; i--)
    {
      HAL_GPIO_WritePin(MOSI_GPIO_Port, MOSI_Pin, (varCmd >> (0+i)) & 1 ? GPIO_PIN_SET : GPIO_PIN_RESET);
      HAL_Delay(10);

      HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
      HAL_Delay(10);
      HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
      HAL_Delay(10);
    }

    for(int i = 7; i < 8; i--)
    {
      HAL_GPIO_WritePin(MOSI_GPIO_Port, MOSI_Pin, (data[0] >> (0 + i)) & 1 ? GPIO_PIN_SET : GPIO_PIN_RESET);
      HAL_Delay(10);

      HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
      HAL_Delay(10);
      HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
      HAL_Delay(10);

    }
```

```
    for(int i = 1; i < 8; i++)
    {
      HAL_GPIO_WritePin(MOSI_GPIO_Port, MOSI_Pin, (varCmd >> (0+i)) & 1 ? GPIO_PIN_SET : GPIO_PIN_RESET);
      HAL_Delay(10);

      HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
      HAL_Delay(10);
      HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
      HAL_Delay(10);
    }

    for(int i = 7; i < 8; i++)
    {
      HAL_GPIO_WritePin(MOSI_GPIO_Port, MOSI_Pin, (data[1] >> (0+i)) & 1 ? GPIO_PIN_SET : GPIO_PIN_RESET);
      HAL_Delay(10);

      HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
      HAL_Delay(10);
      HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
      HAL_Delay(10);

    }


    HAL_GPIO_WritePin(NSS_GPIO_Port, NSS_Pin, GPIO_PIN_SET);
}
```

Рисунок 21 – Устройство SPI

Tim1 и Tim2 таймеры служат для обработки прерывания, связанного с началом замеров и преобразования данных в АЦП. Функция CoreTimer1Setup устанавливает частоту работы первого таймера.

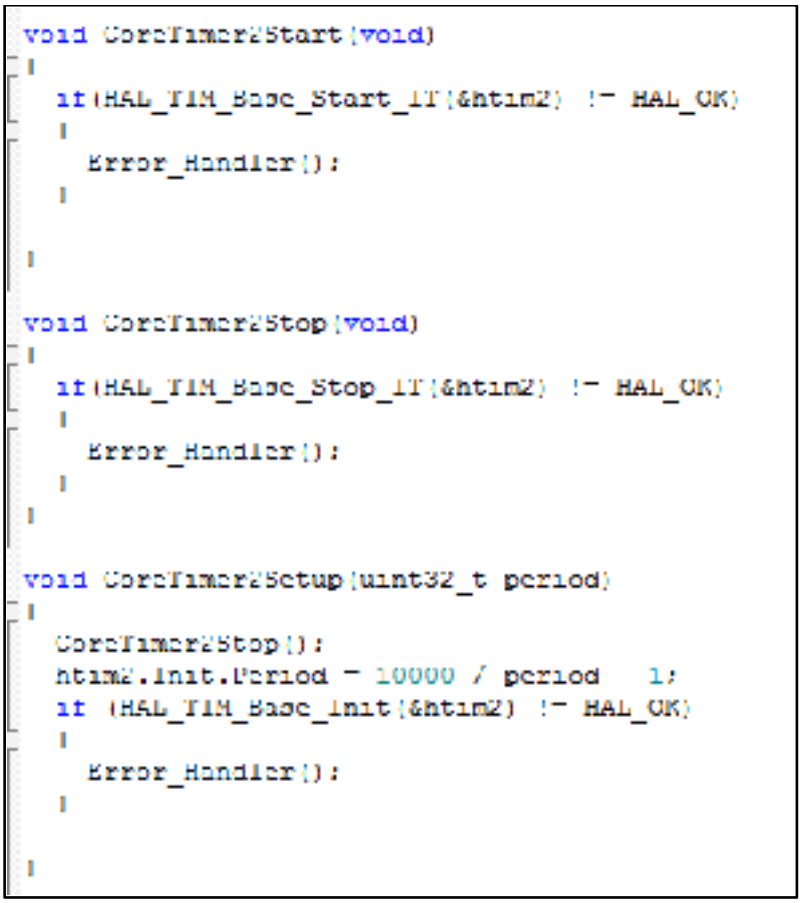
```
void CoreTimer1Start(void)
{
  if(HAL_TIM_Base_Start_IT(&htim1) != HAL_OK)
  {
    Error_Handler();
  }

}

void CoreTimer1Stop(void)
{
  if(HAL_TIM_Base_Stop_IT(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
}

void CoreTimer1Setup(uint32_t period)
{
  CoreTimer1Stop();
  htim1.Init.Period = 10000 / period - 1;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
```

Рисунок 22 – Работа таймера Tim1



```
void CoreTimer2Start(void)
{
  if(HAL_TIM_Base_Start_IT(&htim2) != HAL_OK)
  {
    Error_Handler();
  }

}

void CoreTimer2Stop(void)
{
  if(HAL_TIM_Base_Stop_IT(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
}

void CoreTimer2Setup(uint32_t period)
{
  CoreTimer2Stop();
  htim2.Init.Period = 10000 / period - 1;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }

}
```

Рисунок 23 – Работа таймера Tim2

Таймер Tim3 является таймером для основного цикла программы, и служит для прерывания связанного с началом работы измерений.

```
void CoreTimerWorkStart(void)
{
  if(HAL_TIM_Base_Start_IT(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
}

void CoreTimerWorkStop(void)
{
  if(HAL_TIM_Base_Stop_IT(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
}

void CoreTimerWorkSetup(uint32_t period)
{
  CoreTimerWorkStop();
  htim3.Init.Period = period - 1;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Рисунок 24 – Работа таймера Tim3

ЗАКЛЮЧЕНИЕ

Результатом выполнения выпускной квалификационной работы стало создание программного обеспечения для геофизического прибора. В соответствии с изложенными требованиями для реализации задачи был выбран и запрограммирован микроконтроллер серии STM32. Средой разработки программы аналогово-цифрового преобразователя выбран Keil μvision, поддерживающий язык программирования C++.

В ходе успешных опытных работ, проводимых в 2015 и 2016 годах, был применен данный двухканальный геофизический прибор, основанный на новом методе вызванной поляризации.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Потылицын, В. С. Дифференциальный фазовый метод извлечения геофизической информации из шумового электромагнитного поля земли в диапазоне частот: дис. … д-ра радиофизики: 01.04.03 / Потылицын Вадим Сергеевич. –Красноярск, 2016. -143 с.
2. Шайдуров, Г. Я. О возможности использования естественных электромагнитных полей для регистрации потенциалов вызванной поляризации. Новая аппаратура и методика её применения в народном хозяйстве / Шайдуров Георгий Яковлевич. –Красноярск, 1967, вып. 2, с.3 – 7.
3. Хмелевской, В. К. Геофизические методы исследования земной коры. Часть 1. — Международный университет природы, общества и человека «Дубна», 1997.
4. Аналогово-цифровое преобразование [Электронный ресурс]. – Режим доступа: https://geektimes.ru/post/253708.
5. Языки программирования [Электронный ресурс]. – Режим доступа: http://ec-skat.ru/service/Programmirovanie/YAziki_programmirovaniya.
6. Языки программирования для микроконтроллеров [Электронный ресурс]. – Режим доступа: http://digteh.ru/Progr/progr.php.
7. Интерфейсы микроконтроллеров [Электронный ресурс]. – Режим доступа: https://www.drive2.ru/b/2602560.
8. Последовательный интерфейс SPI [Электронный ресурс]. – Режим доступа: http://www.gaw.ru/html.cgi/txt/interface/spi/index.htm.
9. Аналого-цифровой преобразователь в микроконтроллерах STM32 [Электронный ресурс]. – Режим доступа: https://www.compel.ru/lib/ne/2011/2/7-atsp-v-mikrokontrollerah-stm32-periferiya-reshaet-mnogoe.
10. 32-разрядные микроконтроллеры серии STM32 [Электронный ресурс]. – Режим доступа: http://www.kit-e.ru/articles/micro/2008_11_82.php.
11. Официальный сайт STMicroelectronics [Электронный ресурс]. – Режим доступа: http://www.st.com/content/st_com/en.html.

12. Официальный сайт производителя Keil uVision [Электронный ресурс]. – Режим доступа: http://www.keil.com.

13. Характеристика Keil uVision [Электронный ресурс]. – Режим доступа: http://cxem.net/software/keil.php.

14. STM32CubeMX – конфигуратор для микроконтроллеров STMicroelectronics [Электронный ресурс]. – Режим доступа: https://www.compel.ru/lib/ne/2016/2/8-cubemx-i-workbench-sozdanie-proekta-na-baze-stm32-s-pomoshhyu-besplatnogo-po.

ПРИЛОЖЕНИЕ А

Блок-схемы

Рисунок А.1 – Блок-схема алгоритма работы прибора

Рисунок А.2 – Блок-схема программы микроконтроллера

ПРИЛОЖЕНИЕ Б

Листинг файлов исходного кода программы

```
/* USER CODE BEGIN Includes */
#include "Core.h"
#include <string.h>
#include "stm32f1xx_hal.h"

#define NSS_Pin GPIO_PIN_4
#define NSS_GPIO_Port GPIOA
#define SCK_Pin GPIO_PIN_5
#define SCK_GPIO_Port GPIOA
#define MOSI_Pin GPIO_PIN_7
#define MOSI_GPIO_Port GPIOA

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

SPI_HandleTypeDef hspi1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim7;
TIM_HandleTypeDef htim12;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM6_Init(void);
static void MX_TIM7_Init(void);
static void MX_TIM4_Init(void);
static void MX_TIM12_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/
uint8_t spiR[4];
uint8_t uartIn[16];
HAL_StatusTypeDef UARTStatus =  0;

uint8_t t[16];

uint8_t Rx_indx, Rx_data[2], Rx_Buffer[100], Transfer_cplt;

volatile uint32_t timestamp = 0;
```

```
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI1_Init();
  MX_USART2_UART_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  MX_ADC1_Init();
  MX_TIM6_Init();
  MX_TIM7_Init();
  MX_TIM4_Init();
  MX_TIM12_Init();

  /* USER CODE BEGIN 2 */
       spiR[0]=0x13/* 223*/; spiR[1]=127; spiR[2]=223; spiR[3]=127;
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);

       GPIO_InitTypeDef GPIO_InitStruct;
       GPIO_InitStruct.Pin = NSS_Pin|SCK_Pin|MOSI_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  GPIO_InitStruct.Pin = GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* USER CODE END 2 */

       int grcConst = 100;

  int grc1 = 20;
       htim6.Init.Period = 1000000 / (2 * grc1 * grcConst);
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }
       if(HAL_TIM_Base_Start_IT(&htim6) != HAL_OK)
       {
             Error_Handler();
       }

  int grc2 = 60;
       htim4.Init.Period = 1000000 / (2 * grc2 * grcConst);
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {
```

```
    Error_Handler();
  }
      if(HAL_TIM_Base_Start_IT(&htim4) != HAL_OK)
      {
            Error_Handler();
      }

  int grc3 = 200;
      htim12.Init.Period = 1000000 / (2 * grc3 * grcConst);
  if (HAL_TIM_Base_Init(&htim12) != HAL_OK)
  {
    Error_Handler();
  }
      if(HAL_TIM_Base_Start_IT(&htim12) != HAL_OK)
      {
            Error_Handler();
      }

      HAL_Delay(500);
      CoreStarted = 1;

      uint8_t period[2] = {128, 128};
      CoreSpiSetup(period);

      CoreTimerWorkSetup(5000);

      HAL_UART_Receive_IT(&huart2, Rx_data, 1);

  while (1)
  {
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
            HAL_Delay(1000);

            if(timestamp > 1 && Rx_indx > 0)
            {
                  CoreProcessInput(Rx_Buffer, Rx_indx);

                  Rx_indx = 0;
                  Transfer_cplt = 0;
                  memset(Rx_data, 0, 16);
                  memset(Rx_Buffer, 0, 100);
                  HAL_UART_Receive_IT(&huart2, Rx_data, 1);
            }
  }
       /* USER CODE END 3 */
 }

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL8;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
```

```
                          RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }

  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV8;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }

  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC1 init function */
static void MX_ADC1_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Common config
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = ENABLE;
  hadc1.Init.NbrOfDiscConversion = 1;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 2;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Regular Channel
    */
  sConfig.Channel = ADC_CHANNEL_11;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure Regular Channel
    */
  sConfig.Channel = ADC_CHANNEL_12;
  sConfig.Rank = 2;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* SPI1 init function */
static void MX_SPI1_Init(void)
{
```

```
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_HARD_INPUT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_4;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM1 init function */
static void MX_TIM1_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 799;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 9999;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM2 init function */
static void MX_TIM2_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 799;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 9999;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM3 init function */
static void MX_TIM3_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 7999;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 999;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM4 init function */
static void MX_TIM4_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 31;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 1;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM6 init function */
static void MX_TIM6_Init(void)
{

  TIM_MasterConfigTypeDef sMasterConfig;

  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 31;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 24;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM7 init function */
static void MX_TIM7_Init(void)
{

  TIM_MasterConfigTypeDef sMasterConfig;

  htim7.Instance = TIM7;
  htim7.Init.Prescaler = 0;
  htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim7.Init.Period = 1;
  if (HAL_TIM_Base_Init(&htim7) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM12 init function */
static void MX_TIM12_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;

  htim12.Instance = TIM12;
  htim12.Init.Prescaler = 31;
  htim12.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim12.Init.Period = 1;
  htim12.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_Base_Init(&htim12) != HAL_OK)
  {
```

```
    Error_Handler();
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim12, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* USART2 init function */
static void MX_USART2_UART_Init(void)
{

  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4|GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1, GPIO_PIN_RESET);

  /*Configure GPIO pins : PC4 PC5 */
  GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PB0 PB1 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
```

```
void CoreTimer1Start(void)
{
	if(HAL_TIM_Base_Start_IT(&htim1) != HAL_OK)
	{
		Error_Handler();
	}

}

void CoreTimer1Stop(void)
{
	if(HAL_TIM_Base_Stop_IT(&htim1) != HAL_OK)
	{
		Error_Handler();
	}
}

void CoreTimer1Setup(uint32_t period)
{
	CoreTimer1Stop();
	htim1.Init.Period = 10000 / period - 1;
	if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }

}

void CoreTimer2Start(void)
{
	if(HAL_TIM_Base_Start_IT(&htim2) != HAL_OK)
	{
		Error_Handler();
	}

}

void CoreTimer2Stop(void)
{
	if(HAL_TIM_Base_Stop_IT(&htim2) != HAL_OK)
	{
		Error_Handler();
	}
}

void CoreTimer2Setup(uint32_t period)
{
	CoreTimer2Stop();
	htim2.Init.Period = 10000 / period - 1;
	if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }

}

void CoreSpiSetup(uint8_t data[])
{
		uint8_t varCmd = 0x13;

		HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
		HAL_GPIO_WritePin(NSS_GPIO_Port, NSS_Pin, GPIO_PIN_RESET);
		HAL_Delay(10);

		for(int i=1; i <=8; i++)
		{
			HAL_GPIO_WritePin(MOSI_GPIO_Port, MOSI_Pin, (varCmd >> (8-i)) & 1 ?
GPIO_PIN_SET : GPIO_PIN_RESET);
			HAL_Delay(10);
```

```
                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
                        HAL_Delay(10);
                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
                        HAL_Delay(10);
                }

                for(int i=1; i <=8; i++)
                {
                        HAL_GPIO_WritePin(MOSI_GPIO_Port, MOSI_Pin, (data[0] >> (8-i)) & 1 ?
GPIO_PIN_SET : GPIO_PIN_RESET);
                        HAL_Delay(10);

                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
                        HAL_Delay(10);
                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
                        HAL_Delay(10);

                }

                for(int i=1; i <=8; i++)
                {
                        HAL_GPIO_WritePin(MOSI_GPIO_Port, MOSI_Pin, (varCmd >> (8-i)) & 1 ?
GPIO_PIN_SET : GPIO_PIN_RESET);
                        HAL_Delay(10);

                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
                        HAL_Delay(10);
                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
                        HAL_Delay(10);
                }

                for(int i=1; i <=8; i++)
                {
                        HAL_GPIO_WritePin(MOSI_GPIO_Port, MOSI_Pin, (data[1] >> (8-i)) & 1 ?
GPIO_PIN_SET : GPIO_PIN_RESET);
                        HAL_Delay(10);

                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_SET);
                        HAL_Delay(10);
                        HAL_GPIO_WritePin(SCK_GPIO_Port, SCK_Pin, GPIO_PIN_RESET);
                        HAL_Delay(10);

                }


                HAL_GPIO_WritePin(NSS_GPIO_Port, NSS_Pin, GPIO_PIN_SET);
}


void CoreTimerWorkStart(void)
{
        if(HAL_TIM_Base_Start_IT(&htim3) != HAL_OK)
        {
                Error_Handler();
        }
}

void CoreTimerWorkStop(void)
{
        if(HAL_TIM_Base_Stop_IT(&htim3) != HAL_OK)
        {
                Error_Handler();
        }
}

void CoreTimerWorkSetup(uint32_t period)
{
        CoreTimerWorkStop();
        htim3.Init.Period = period - 1;
        if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
```

```
  {
    Error_Handler();
  }
}


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
       if (huart->Instance != USART2)
              return;

       timestamp = 0;

       Rx_Buffer[Rx_indx++]=Rx_data[0]; //add data to Rx_Buffer

       HAL_UART_Receive_IT(&huart2, Rx_data, 1);     //activate  UART  receive  interrupt
every time
}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */

}

#endif
```

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В. Непомнящий
подпись инициалы, фамилия
«__»_____2018 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 – «Информатика и вычислительная техника»
код и наименование специальности

Разработка программного обеспечения геофизического прибора для поиска минеральных ресурсов на основе извлечения информации из пассивных шумовых полей земли
тема

Пояснительная записка

Руководитель _____ _____ Д.А. Швец
подпись, дата должность, учёная степень инициалы, фамилия

Выпускник _____ Д.Н. Исаев
подпись, дата инициалы, фамилия

Нормоконтролер _____ В.И. Иванов
подпись, дата инициалы, фамилия

Красноярск 2018