СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последнее время стали актуальны проблемы автоматизации всех сфер жизнедеятельности человека. Многие из них подвергаются автоматизации уже десятки лет. Однако существенных успехов удалось добиться в данном направлении с момента внедрения электронных устройств автоматизации промышленности. В общем случае, системы управления позволяли ускорить выполнения отдельных видов работ, снизить риски, связанные с человеческим фактором и др.

Но во времена развития автоматизированных систем управления промышленностью мало уделялось внимания системам автоматизации, которые сейчас называются «умный дом».

Сегодня с помощью локальных вычислительных сетей и интернету система умного дома может управлять всеми инженерными системами и устройствами с помощью современных мобильных телефонов, планшетов и ноутбуков. Сочетание беспроводных и проводных систем позволяет управлять безопасностью, видеонаблюдением, светом, температурой и многим другим. Основой данной системы является маленькая микро-ЭВМ, управляющая приборами, которые нужно выключить или включить, изменить параметры освещенности или выставить необходимую температуру в кондиционере.

Целью данного дипломного проекта является разработка модуля автоматизации управления хозяйством, которая обеспечит сбор данных с датчиков, отправку их на сервер и управление исполнительными устройства, которые подключены к данного модулю.

Также целью данного дипломного проекта является разработка алгоритма взаимодействия датчиков с модулем автоматизации, а также разработка системы команд для управления исполнительными устройствами, подключенными к системе.

В соответствии с поставленной целью были определены следующие задачи:

– разработка архитектуры устройства;

– реализация протокола MODBUS для приема и передачи данных между сервером и модулем автоматизации;

– реализация протокола приема данных от исполнительных устройств и датчиков и отправка команд для исполнительных устройств;

– реализация беспроводного протокола для взаимодействия модуля автоматизации и других устройств.

1 ОБЗОР ЛИТЕРАТУРЫ

Для достижения поставленных целей по реализации системы автоматизации управления хозяйством необходимо рассмотреть аналоги продукта, которые присутствуют уже на рынке.

1.1 Умный дом от «Белтелеком»

Одним из аналогов на белорусском рынке является умный дом от телекоммуникационной компании Белтелеком, основным направлением которой является предоставление технологии электросвязи для своих абонентов. Однако в последние время данная компания предоставляет возможность удаленно контролировать состояние своего дома или квартиры, повышать комфорт и более рационально использовать ресурсы [1].

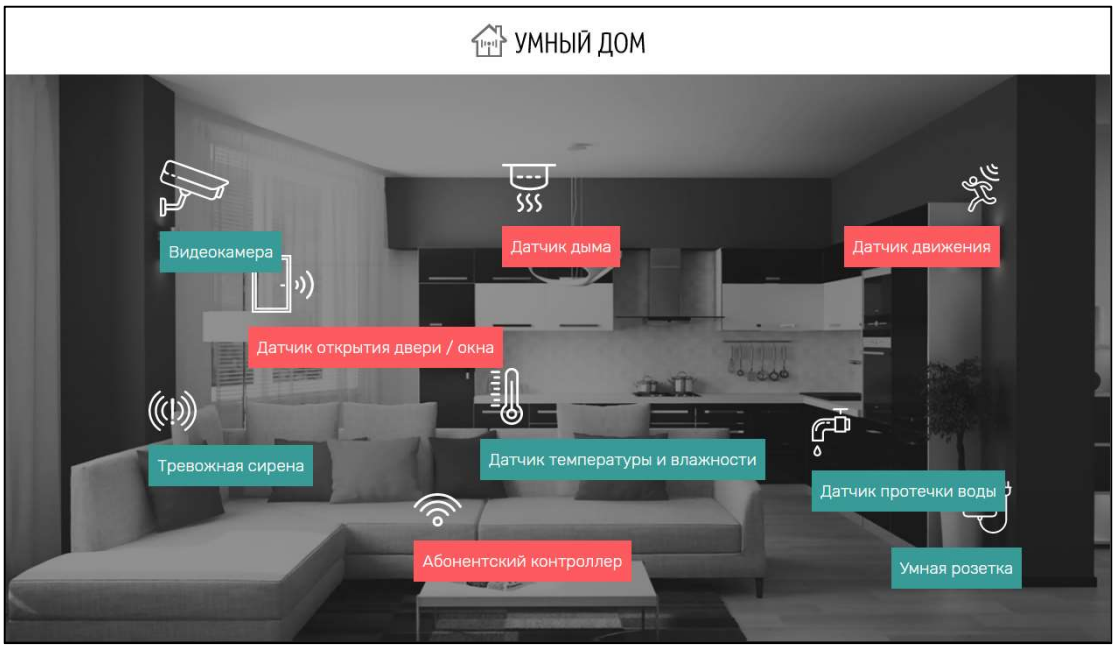

Рисунок 1.1 – Система умного дома от «Белтелеком» [1]

Имеется возможность воспользоваться базовым и дополнительным оборудованием, которое комплектуется по заказу абонента. Базовая комплектация включает в себя три датчика и контроллер управления. Дополнительное оборудование добавляет еще несколько датчиков, сирену, видеокамеру и умную розетку. К плюсам данного умного дома можно отнести следующие:

– малая абонентская плата;
– удобное приложение для управления системой;
– возможность настройки системы сотрудниками компании.

К минусам данной системы можно отнести:

– малое количество подключаемых устройств от «Белтелеком»;

– отсутствие возможности подключить устройства и датчики других производителей.

1.2 Умный дом Z-Wave

Умный дом Z-Wave основывается на запатентованном беспроводном протоколе связи Z-Wave, разработанном для автоматизации в частном и промышленном секторе. Данный производитель является основным конкурентов разрабатываемой системы автоматизации хозяйства.

Имеются готовые решения под сценарии использования умного дома, показанные на рисунке 1.2 [2] с сайта белорусского дистрибьютора Z-Wave. Помимо готовых решений компания предоставляет услуги проектирования, подбора и поставки оборудования и его установки у конечного пользователя, а также сервисное обслуживание предоставляемого оборудования.

Рисунок 1.2 – Сценарии использования умного дома [2]

К основным плюсам данной системы можно отнести:

– простота подключения устройств к Z-Wave модулю;

– расширяемость по мере необходимости и возникновения новых задач и идей;

– совместимость с беспроводным стандартом связи;

К основным минусам системы можно отнести:

– высокая стоимость системы;

– ограничения масштаба и радиуса действия модулей;

1.3 Протокол Modbus

Modbus – это протокол, который широко используется на международном уровне, открыт для всех пользователей и поддерживается многими производителями [3]. Для использования в современных сетях из него был разработан протокол Modbus TCP. Данный протокол в настоящее время является открытым для изменения стандартом, представленным IETF, организацией по стандартизации Интернета. Эта открытость означает, что каждый производитель и пользователь могут использовать протокол – возможность, которой уже воспользовались многие ведущие производители. Ускоренное развитие Ethernet соединения в промышленных зонах и офисной среде расширило сферу применения Modbus TCP во всех направлениях. Неоднородные системные среды являются типичными областями использования данного протокола.

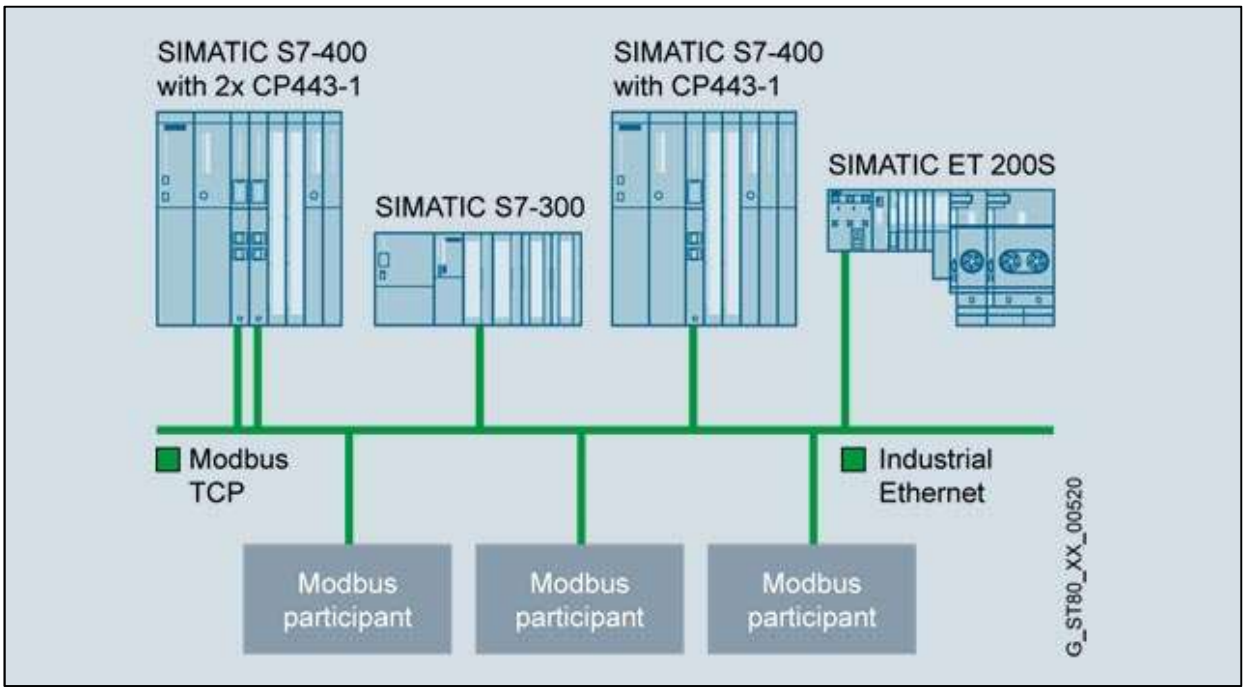

Рисунок 1.3 – Пример использования Modbus в промышленной системе автоматизации [3]

1.4 Радиопередатчик для беспроводной связи

Помимо проводных приема и передачи данных рассматривается вариант использования датчиков и исполнительных устройств, которые осуществляют обмен по беспроводному интерфейсу. Для организации приема и передачи данных по беспроводной связи рассматривается вариант использования радиопередатчика nRF24L01, представленный на рисунке 1.4. Данный модуль осуществляет обмен на частоте 2.4 ГГц, может использовать до 125 каналов, поддерживает передачу данных на скорости до 2 мбит/с и на низких скоростях

может обмениваться данными до 100 метров [4]. Рабочим напряжением модуля является напряжение от 1.9 до 3.6 вольт. В обычном режиме модуль потребляет всего лишь 12.3 мА. Данная характеристика важна при выборе, так как питание модулей беспроводной связи планируется осуществлять от батареек.

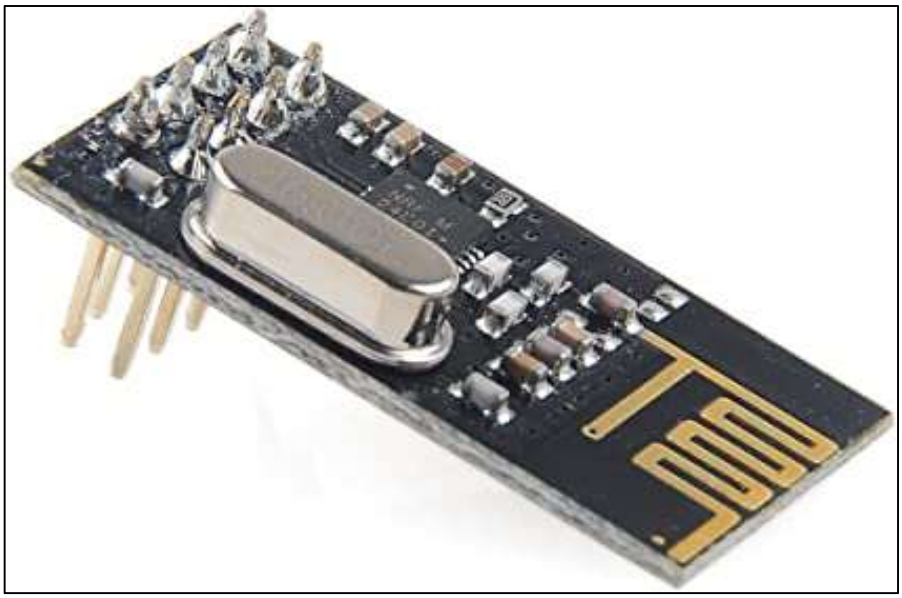

Рисунок 1.4 - Радиопередатчик nRF24L01 [5]

1.5 Датчики температуры и влажности

При рассмотрении датчиков температуры и влажности необходимо было также учитывать габариты, стоимость и совместимость с платами STM32FXX. Так как разрабатываемый модуль предполагается сделать максимально маленьким и удобным в использовании. Датчик влажности и температуры DHT22, представленный на рисунке 1.5, имеет малые габариты.

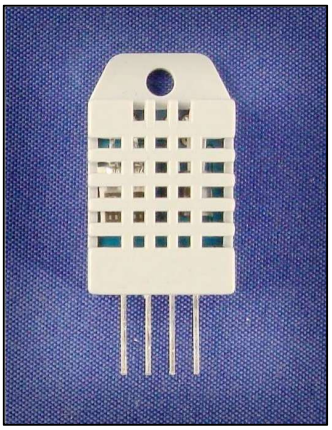

Рисунок 1.5 – Датчик относительной влажности и температуры [6]

Рабочее напряжение данного модуля 3.3 – 6 вольт. Измеряемая влажность имеет полный диапазон значений от 0 до 100%, а датчик температуры измеряет значения в диапазоне от -40 до +80 °C [6].

1.6 Датчик звука

Распознавание шума в комнате является также важно при проектировании системы автоматизации. Для определения шума был выбран датчик звука KY-038 представленный на рисунке 1.6.

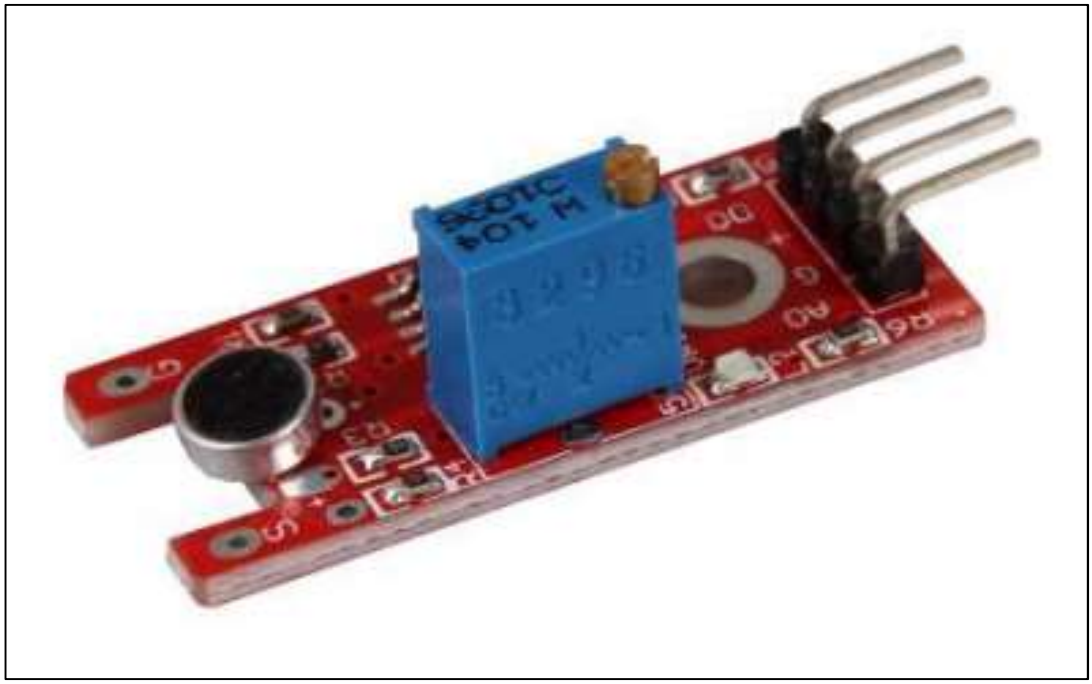

Рисунок 1.6 – Датчик звука KY-038 [7]

Датчик имеет три основных компонента, которые и определяют происходящее в комнате. Сенсорный блок, который снимает значение и отправляет аналоговый сигнал на второй блок – усилитель. Усилитель усиливает сигнал в соответствии с сопротивлением потенциометра и отправляет сигнал на аналоговый выход модуля. Третий компонент – компаратор, который переключает цифровой выход и светодиод, если сигнал падает ниже определенного значения. Контролировать чувствительность датчика можно регулируя потенциометр усилителя [7].

1.7 Датчик присутствия

Обнаружение человека в помещении можно применять для того, чтобы уменьшить расход электрической энергии в помещении. Для этого обычно используют датчик присутствия. Для реализации был рассмотрен датчик присутствия HC-SR501, представленный на рисунке 1.7.

Данный датчик также имеет рабочее напряжение от 5 до 20 вольт, потребляемая мощность 65 мА, время обнаружения присутствия 0.2 секунды. Чувствительность: до семи метров и 120 градусов. Выходное напряжение датчика составляет 3.3 В, что совместимо с микроконтроллерами STM32 [8].

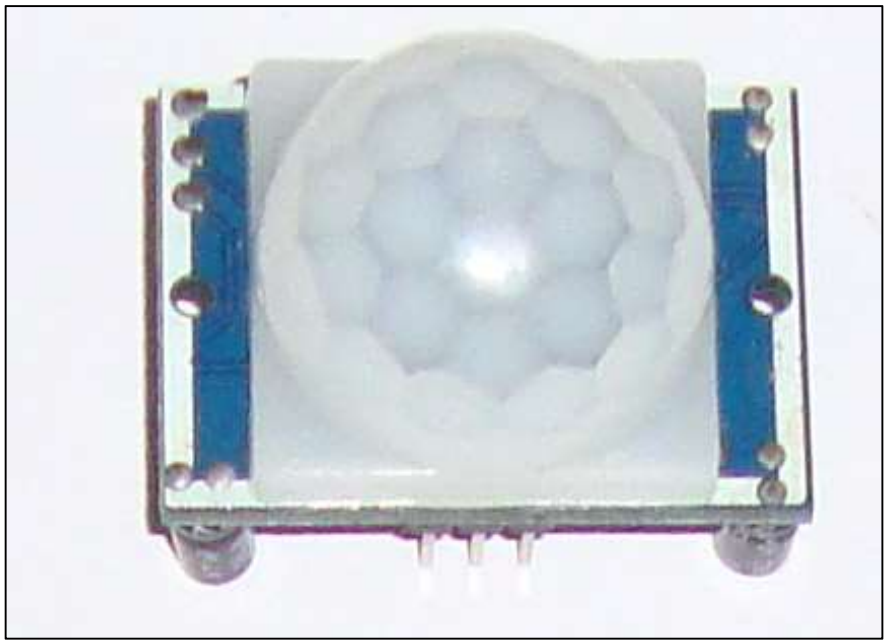

Рисунок 1.7 – Датчик присутствия [9]

1.8 Датчик освещенности TCRT5000

Для определения освещенности помещения был рассмотрен датчик TCRT5000, представленный на рисунке 1.8, основанный на отражении света, он включает в себя инфракрасный излучатель и фототранзистор, который блокирует видимый свет [10].

Рисунок 1.8 – Датчик TCRT5000 [10]

1.9 Датчик CO_2

Углекислый газ невозможно увидеть не вооруженным взглядом, однако существую датчики, которые могут позволить его обнаружить и определить его концентрацию в воздухе. Понимание того, какая концентрация углекислого газа в комнате позволяет определить момент, когда стоит включить вентиляцию в помещении или же определить утечку газа в местах, где это возможно и оповестить владельца помещения о том, что необходимо устранить проблему.

Датчик MH-Z19, представленный на рисунке 1.9, это инфракрасный газовый модуль, использующий технологию не дисперсионного инфракрасного метода для обнаружения присутствия CO_2 в воздухе. В модуль встроен датчик температуры, чтобы можно было измерять с учетом данной погрешности [11]. Датчик имеет универсальный асинхронный приемопередатчик для обмена с подключаемым устройством.

Рисунок 1.9 – Датчик MH-Z19 [11]

1.10 Шаговый двигатель

Шаговый двигатель – это универсальное устройство для решения многих задач, связанных с 3D-принтерами, робототехникой и другим. Шаговый двигатель 17HS440IS, представленный на рисунке 1.10, четырех проводной биполярный шаговый двигатель с шагом в 1,8° для плавного движения и хорошего удерживающего момента [12]. Максимальный потребляемый ток 1,7А/фаза.

Рисунок 1.10 – Шаговый двигатель 17HS440IS [12]

1.11 Микроконтроллер STM32F4XX

После рассмотрения аналогов, необходимо определиться с моделью микроконтроллера, который будет использоваться в системе автоматизации хозяйства. Необходимо предусмотреть возможность приема и передачи данных по проводному и беспроводному интерфейсу. Возможность считывания данных с датчиков и отправления команд исполнительным устройствам.

Контроллеры семейства STM32F4, структурная схема которых представлена на рисунке 1.11, основываются на ARM Cortex-M4 ядре. Данные микроконтроллеры изготавливаются по 90 нм технологии с использованием ST Microelectronics Accelerator для достижения наилучших показателей производительности. Снизить потребление тока удалось при помощи динамического потребления питания при выполнении кода из Flash-памяти.

Все микроконтроллеры данной серии поддерживают DSP-инструкции. Имеют высокопроизводительную AHB-матрицу шин. Имеют до 1 Мбайта Flash-памяти и до 192 кбайт SRAM-памяти. Напряжение питания от 1.8 до 3.6 В. Также у этих микроконтроллеров имеются внутренние RC-генераторы на 16МГц и 32кГц для RTC. Подключаются внешние источники тактирования от 4 до 26 МГц.

Имеется три 12-бит аналога цифровых преобразователя на 24 входных канала. В наличии микроконтроллера также есть два 12-бат цифро-аналоговых преобразователя. Для того чтобы процессор не простаивал можно использовать DMA-контроллер на 16 потоков с поддержкой пакетной передачи данных.

Важно при проектировании системы автоматизации уделить внимание алгоритмам шифрования данных. У микроконтроллеров серии STM32FXX может присутствовать модуль шифрования: AES 128, 192, 256 и другие [13].

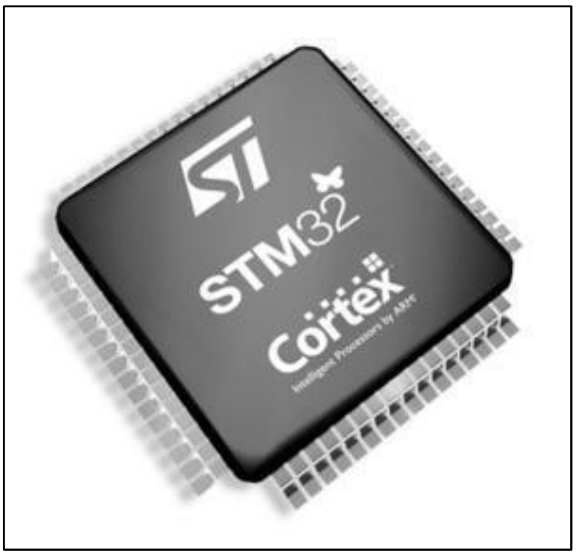

Рисунок 1.11 – Структурная схема микроконтроллера STM32FXX [13]

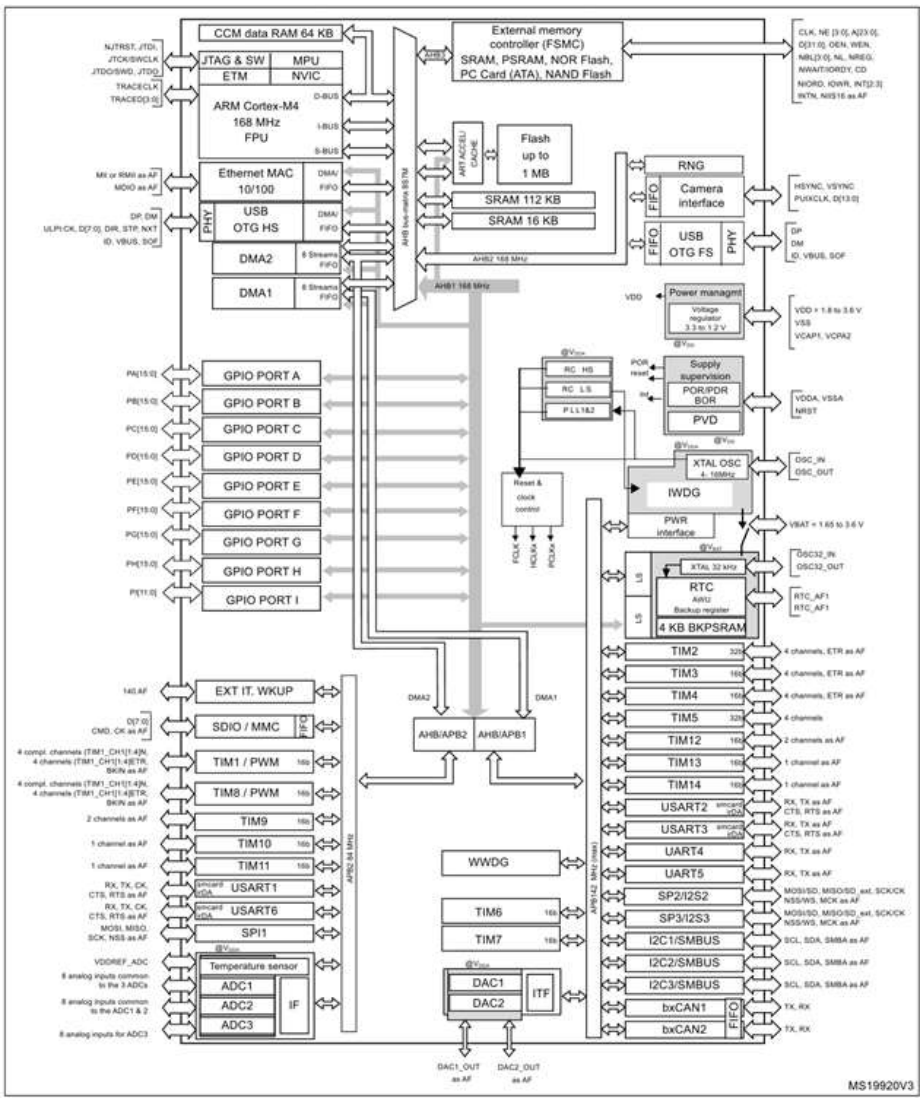

Рисунок 1.12 – Структурная схема микроконтроллера STM32FXX [13]

1.12 Преобразователи напряжения DC/DC

Преобразователи – это электронные устройства, которые представляют из себя устройства в герметичных защищенных корпусах с выводами на для монтажа на печатную плату. Данные устройства преобразуют постоянное напряжение в постоянное. Одними из высокопроизводительных преобразователей являются преобразователи компании TRACO ELECTRONIC. TCR 1-2450, представленный на рисунке 1.13, является преобразователем питания с возможным входным напряжением от 4.6 до 36 вольт и выходным напряжением от 1.5 до 15 вольт [14].

Рисунок 1.13 – Преобразователь TCR 1-2450 [14]

1.13 Интерфейс RS-485

Интерфейс RS-485 является стандартом физического уровня для асинхронного интерфейса. В основе своей используется в промышленных сетях, особенно в сфере автоматизации производства. В стандарте интерфейса оговариваются лишь его электрические и временные характеристики. В частности, данный интерфейс поддерживает до 32 приемопередатчиков в одном сегменте сети. Максимальная длина одного сегмента сети составляет до 1200 метров. В один момент времени может быть активным лишь один передатчик. Максимальное количество узлов в сети – 256 с учетом усилителей. Тип приемопередатчиков должен быть дифференциальный и потенциальный. На расстоянии до 100 метров максимально допустимая скорость передачи данных до 2400 кбит/с.

Стандарт интерфейса не оговаривает тип гальванической развязки, типы соединений и используемые кабели для передачи сигналов. Протокол обмена также выбирается в зависимости от потребностей данной сети, как и параметры качества сигнала.

1.14 Гальваническая развязка

При рассмотрении RS-485 интерфейса была упомянута гальваническая развязка, которая представляет из себя передачу информационного сигнала между электрическими цепями, которые не имеют непосредственного электрического контакта между собой [15]. При использовании гальванической развязки электрические потенциалы могут сильно отличаться. Существует большое количество видов развязок, которые делятся на:

– трансформаторные;
– оптоэлектронные;
– акустические;
– радиоканальные;
– звуковые;
– емкостные;
– с преобразователями.

Для того чтобы гальванически развязать RS-485 интерфейс можно использовать микросхему ISO35DW, представленную на рисунке 1.14.

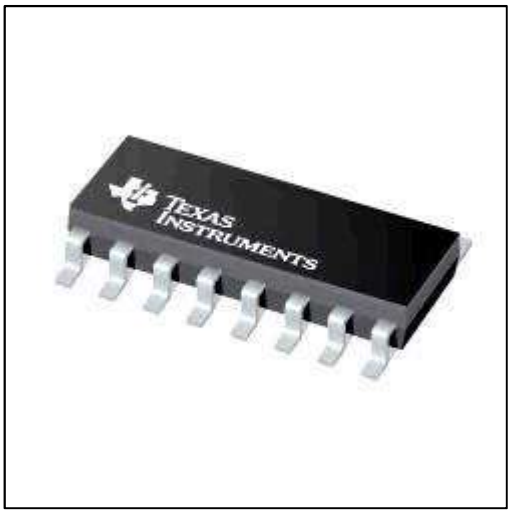

Рисунок 1.14 – Микросхема ISO35DW [16]

Данная микросхема является полнодуплексным дифференциальным драйвером для приемопередатчиков, использующих 485/422 интерфейс. Это устройство предназначено для передачи данных на большие расстояния.

2 РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ

Изучив предметную область разрабатываемой системы автоматизации, были разработаны основные требования, которые должны быть выполнены при реализации дипломного проекта. Для упрощения разработки системы разобьем ее на структурные блоки.

2.1 Описание основных блоков устройства

В разрабатываемой аппаратной системе автоматизации управления хозяйством выделены следующие блоки:

– блок преобразователя питания;
– блок управления;
– блок интерфейса взаимодействия с ПК;
– блок сервера;
– блок беспроводной связи;
– блок исполнительного устройства;
– блок датчиков;
– блок беспроводных датчиков.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.019 С1.

Блоки были выделены таким образом, чтобы они выполняли определенную задачу, чтобы система работала корректно. Рассмотрим более детально работу и задачи каждого блока отдельно, а также их взаимодействие и обмен данных между блоками.

2.2 Блок преобразователя питания

Блок преобразователя питания является полностью аппаратным компонентом. Данный блок представляет из себя электрическое устройство, преобразующее электрическую энергию с более высокими значениями напряжения в электрическую энергию с более низкими значениями напряжения. Преобразование электрической энергии необходимо ввиду того, что блок датчиков в основном использует напряжение от 3.3 до 5В.

Микроконтроллеры от компании STMicroelectronics требуют питание от источника с напряжение в диапазоне от 1.8 до 3.6В. Блок беспроводной связи имеет диапазон поддерживаемых напряжений от 1.9 до 3.6В.

Исходя из потребностей выше перечисленных модулей, необходимо будет сделать преобразование входного напряжения до двух значений:

– 3.6В для питания блока беспроводной связи, микроконтроллера и некоторых датчиков.

– 5В для питания датчиков у которых поддерживаемый диапазон напряжений от данного значения.

2.3 Блок датчиков

Блок датчиков – это набор аппаратных устройств, которые являются средствами измерений и по конструктивным особенностям могут представлять из себя составные устройства из нескольких измерительных преобразователей и микросхем для взаимодействия с другими устройствами или же иметь лишь аналоговые измерительные средства.

Основная задача данного блока – это измерение физических величин, которые будут переданы посредством различных протоколов обмена блоку управления или же будут переданы в аналоговом формате данных.

Все датчики, рассмотренные в обзоре литературы требуют внешнее питание от 3.3 до 5В, поэтому блок датчиков должен будет получать требуемые значения напряжения от блока преобразователя питания.

2.4 Блок беспроводных датчиков

Блок беспроводных датчиков представляет из себя набор аппаратных устройств, представляющих из себя измерительные приборы, которые передают данные посредством беспроводной связи и имеющие автономное питание на борту.

Размещение датчика с набортным беспроводным интерфейсом будет регламентироваться его физическими возможностями приема и передачи данных, а также способом использования датчика. Для питания датчиков с беспроводным интерфейсом достаточным является питание от литий ионной батареи.

Данные, которые будут получены с датчика, могут иметь как аналоговое представление, так и цифровое, так как микроконтроллер, который будет передавать данные, может и выполнять конвертацию аналоговых данных в цифровые. Цифровые данные получать по различным проводным интерфейсам.

Беспроводные датчики будут передавать данные в цифровом формате на основной микроконтроллер в виду отсутствия целесообразности увеличивать количество проводов между ними. Планируется также передача данных о состоянии датчика микроконтроллеру, чтобы можно было отслеживать поломки или отказ датчиков.

2.5 Блок беспроводной связи

Блок беспроводной связи представляет из себя микросхему, которая обеспечивает обмен информацией между основным микроконтроллером и микроконтроллерами, которые подключаются к удаленным датчикам или исполнительным устройствам.

Данный блок отвечает за прием и передачу данных о состоянии датчиков, исполнительных устройств, а также за прием данных датчиков.

Блок беспроводной связи создает соединение, по которому блок управления может получать данные от датчиков с беспроводным интерфейсом, а также получать информацию о их состоянии. Блок беспроводной связи позволяет контролировать устройства, подключенные к подсети, в которой находится микроконтроллер. Так как в радиусе действия датчика беспроводной связи может находиться несколько микроконтроллеров и им необходимо различать устройства подчиненные, которые относятся непосредственно к ним и те, которыми управляет или же значения которых считывает другое устройство мастер. В данном случае мастер устройства – микроконтроллер, а подчиненное устройство – исполнительные устройства и датчики.

Блок беспроводной связи требует 3.6В напряжения, которое будет приходить в данный блок от блока преобразователя питания.

2.6 Блок исполнительных устройств

Блок исполнительных устройств представляет из себя устройства, которые находятся удаленно от микроконтроллера, который ими управляет. Под исполнительными устройствами понимаются устройства, способные управлять какими-либо предметами, устройствами или частями помещения. Некоторые исполнительные устройства – это модули управления или контроллеры, которые будут иметь свое питание, так как потребление тока у них слишком велико для автономного питания. Необходимо будет выбрать исполнительные устройства, которые осуществляют обмен данными по беспроводному интерфейсу.

Основное взаимодействие блока исполнительных устройств будет заключаться в обмене данными с блоком беспроводной связи, исполнительные устройства не будут зависеть от того, какое устройство отдает команды данному исполнительному устройству, если между устройствами настроена беспроводная связь и устройство знает протокольные команды для взаимодействия с исполнительным устройством, то исполнительное устройство будет выполнять его команды.

2.7 Блок управления

Блок управления – это основной блок разрабатываемого модуля, основой которого будет являться микроконтроллер.

Питание необходимое для включения и стабильной работы блока управления будет получаться от блока преобразователя питания. Блок управления выполняет ряд задач, связанных с передачей данных и сбором информации, поэтому рассмотрим их по порядку.

Блок управления будет получать данные с блока датчиков непосредственно по проводному интерфейсу и будет обрабатывать

полученные данные. Также, если данные приходят в виде аналогового сигнала, блок управления должен будет иметь возможность преобразовывать данные сигналы в цифровое представление. После того как блок управления преобразует и получит данные с блока датчиков. Он должен будет их упаковывать с учетом протокола обмена, который будет разработан далее.

Блок беспроводной связи будет конфигурироваться и получать команды от блока управления. Блок управления будет переходить в режим определения окружения и определять устройства, которые будут находиться в его распоряжении. После того как блок управления установит связь со всеми устройствами, которые будут ему подчиняться, он должен будет перевести блок беспроводной связи в режим обмена данными с другими устройствами и выполнять свою основную работу.

Посредством блока беспроводной связи блок управления будет получать данные, которые приходят с блока беспроводных датчиков и также, как и с данными полученными из блока датчиков будет их отправлять в блок интерфейса взаимодействия с ПК. Данные, которые будут отправляться в данный блок должны будут иметь уже структурированный вид, чтобы эту информацию можно было использовать для отображения их конечному пользователю.

Помимо получения данных с блока беспроводных датчиков, блок управления будет осуществлять взаимодействие с исполнительными устройствами, которые подключены будут к беспроводной связи. Исполнительные устройства будут сообщать о своем состоянии микроконтроллеру, а тот в свою очередь передавать команды для данных исполнительных устройств, полученные из блока интерфейса взаимодействия с ПК.

Предполагается, что некоторые датчики из блока датчиков, которые будут располагаться непосредственно рядом с микроконтроллером будут получать питание от самого блока управления.

2.8 Блок интерфейса взаимодействия с ПК

Для того чтобы система имела смысл необходимо передавать данные пользователю, поэтому требуется разработка блока интерфейса взаимодействия с ПК.

По данному интерфейсу планируется передавать данные собранные и обработанные на какой-либо персональный компьютер, например, сервер, который будет заниматься преобразованием полученной информации в более понятный для человека вид.

Блок интерфейса взаимодействия с ПК должен будет иметь хорошую пропускную способность, чтобы своевременно передавать и получать необходимую информацию. Также этот блок будет передавать команды блоку управления, а тот в свою очередь передавать их устройствам, к которым относится данная команда.

2.9 Блок сервера

Блок сервера – это блок, который занимается преобразованием данных, полученных и обработанных микроконтроллером.

Посредством блока взаимодействия с ПК блок сервера будет осуществлять прием данных, после того как блок сервера получит эти данные, он должен будет хранить их и отправлять на приложения, которые будут подключены к серверу. В случае обнаружения проблем, полученных посредством датчиков, сервер должен иметь возможность отправки команд микроконтроллеру по устранению данных проблем с помощью исполнительных устройств. А также отправлять оповещения пользователю о произошедшем, чтобы пользователь мог решить проблему либо самостоятельно, либо отправив команду одному из исполнительных устройств.

В штатном режиме, конечный пользователь должен будет иметь возможность получать от сервера всю информацию от системы автоматизации и иметь возможность управлять исполнительными устройствами, отправляя команду с приложения в блок сервера, который в свою очередь отправит команду далее.

3 РАЗРАБОТКА ФУНКЦИОНАЛЬНОЙ СХЕМЫ

В предыдущем разделе была спроектирована структура проекта и описаны функции основных блоков проекта. В данном разделе блоки будут описаны с точки зрения разработки функций, которые реализуются в разрабатываемом дипломном проекте.

Проект состоит из следующих блоков:

– блок преобразователя питания;
– блок управления;
– блок интерфейса взаимодействия с ПК;
– блок сервера;
– блок беспроводной связи;
– блок исполнительного устройства;
– блок датчиков;
– блок беспроводных датчиков.

3.1 Блок сервера

Рассмотрим функциональные особенности данного блока. Данный блок обеспечивает связь между клиентским устройством и аппаратной частью системы, поэтому он должен иметь возможность считывать информацию с микроконтроллеров, которые находятся в сети. Ввиду того, что расстояние между сервером, который должен будет являться ведущим в обмене данными, и микроконтроллерами, которые находятся в других помещениях и которые являются ведомыми устройствами, может быть до 40-50 метров, необходимо было подобрать интерфейс, который сможет обеспечить прием и передачу данных на такие расстояния.

Для приема и передачи данных был выбран протокол обмена Modbus, описанный в разделе 1.3, ввиду надежности передачи данных, простоты и удобства. Важным фактором было также поддержка данного протокола микроконтроллерами STM32F4XX. На физическом уровне было принято решение использовать RS-485 интерфейс. Данный интерфейс используется при проектировании промышленных сетей и его пропускная способность на расстояниях, которые предполагаются в разрабатываемой системе, является допустимой и достаточной. Так как от блока сервера необходимо будет иметь физическую связь с подчиненными микроконтроллерами, вместе с RS-485 интерфейсом планируется в обжиме передавать и питание, а у каждого контроллера использовать свое преобразование питания до необходимых ему уровней, тем самым упростив проектирование электрической проводки, которая необходима для работы системы в целом.

Далее рассмотрим блок преобразователя питания, который является важной частью разрабатываемого устройства с аппаратной составляющей данного диплома.

3.2 Блок преобразователя питания

Данный функциональный блок должен обеспечивать питанием все датчики и устройства, подключенные к микроконтроллеру напрямую. Как было описано в разделе выше, питание которое необходимо будет преобразовывать, будет приходить вместе с RS-485 интерфейсом. При рассмотрении литературы было установлено, что практически все датчики и драйверы, необходимые для работы исполнительных устройств могут работать от 3.3 и 5 вольт. Следовательно, питание, которое будет приходить извне должно быть преобразовано до 5 и 3.3 вольт. Микроконтроллеры STM32F4XX имеют диапазон допустимых значений напряжения от 1.9 до 3.6 вольт, что было рассмотрено в разделе 1.11.

3.3 Блок датчиков

Блок датчиков представляет из себя набор датчиков, которые будут подключены к микроконтроллеру по проводному интерфейсу. Необходимо обозначить какие датчики будут подключаться к микроконтроллеру. Также необходимо чтобы датчик подключался по интерфейсу, определенному в микроконтроллере, так как у микроконтроллеров серии STM32F4XX каждый вывод подключается к определенной внутренней периферии. Рассмотрим датчики, которые планируется подключать в разрабатываемой системе:

– датчик освещенности;
– датчик присутствия;
– датчик температуры и влажности;
– датчик CO2.

3.3.1 Датчик освещенности

Датчик освещенности должен будет иметь возможность определять уровень освещенности помещения и передавать аналоговый сигнал на микроконтроллер, который в свою очередь будет обрабатывать данный сигнал при помощи внутренних АЦП и передавать данную информацию серверу. Допустимое питание датчика освещенности должно входить в диапазон от 1.8 до 5 вольт.

Датчик освещенности в основе своей использует фоторезистивный эффект, который показывает изменение удельного сопротивления полупроводника, обусловленное действием на него электромагнитного излучения и не связанное с нагреванием полупроводника.

Суть данного явления заключается в том, что при поглощении световой энергии. Достаточной для ионизации собственных атомов полупроводника, происходит увеличение концентрации носителей заряда. Из-за увеличения

концентрации носителей уменьшается удельное сопротивление полупроводника.

Для того чтобы фоторезистивный эффект существовал необходимо, чтобы в полупроводнике происходило поглощение света с образование новых пар носителей заряда, или примесное поглощение с образованием носителей одного заряда.

Вольт-амперные характеристики показывают зависимости светового тока $I_{СВ}$ при неизменной величине светового потока, а также тока в темноте $I_{тем}$ от приложенного к полупроводнику напряжения.

Сопротивление полупроводника определяется в основном сопротивление контактов, если приложено малое напряжение. Напряжение, которое будет приложено к полупроводнику, будет падать на контактах между зернами полупроводника.

В связи с этим, напряженность электрического поля на контактах получается большой даже при малых напряжениях. Поэтому при увеличении напряжения напряжение сопротивления контактов уменьшается либо из-за эффектов сильного поля, либо из-за разогрева контактных областей полупроводника.

График вольт-амперной характеристики представлен на рисунке 3.1.

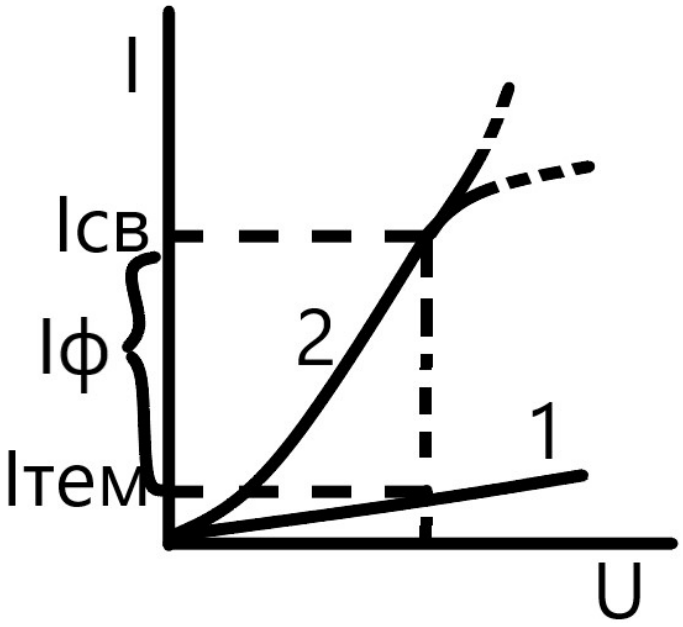

Рисунок 3.1 – Вольт-амперная характеристика полупроводника

На рисунке график 1 показывает вольт-амперную характеристику полупроводника в темноте, а график 2 при освещении.

3.3.2 Датчик присутствия

Датчик присутствия должен будет иметь возможность определять присутствие кого-либо в помещении и передавать цифровой сигнал на микроконтроллер, который в свою очередь будет передавать данную информацию серверу. Допустимое питание датчика освещенности должно входить в диапазон от 3.3 до 5 вольт.

Датчики присутствия, который предполагается использовать, должен использовать пассивный инфракрасный датчик. Данный датчик представляет из себя электронный датчик, который измеряет инфракрасное излучение от объектов, которые попадают в его область видимости.

Принцип работы данного датчика заключается в том, что датчик получает разность потоков инфракрасного излучения, падающего на две соседние площадки. В данном варианте работы важно, что излучение от объекта фокусируется на одной из площадок и при этом вторая фиксирует изменение. Наиболее надежно датчик работает, если изображение объекта попадает сначала на одну площадку, сигнал от этой площадки станет больше, чем на второй, а затем объект передвинется, так что его изображение попадет теперь на вторую площадку и сигнал у второй вырастет, а у первой он станет меньше. На рисунке 3.2 представлен принцип работы данного датчика.

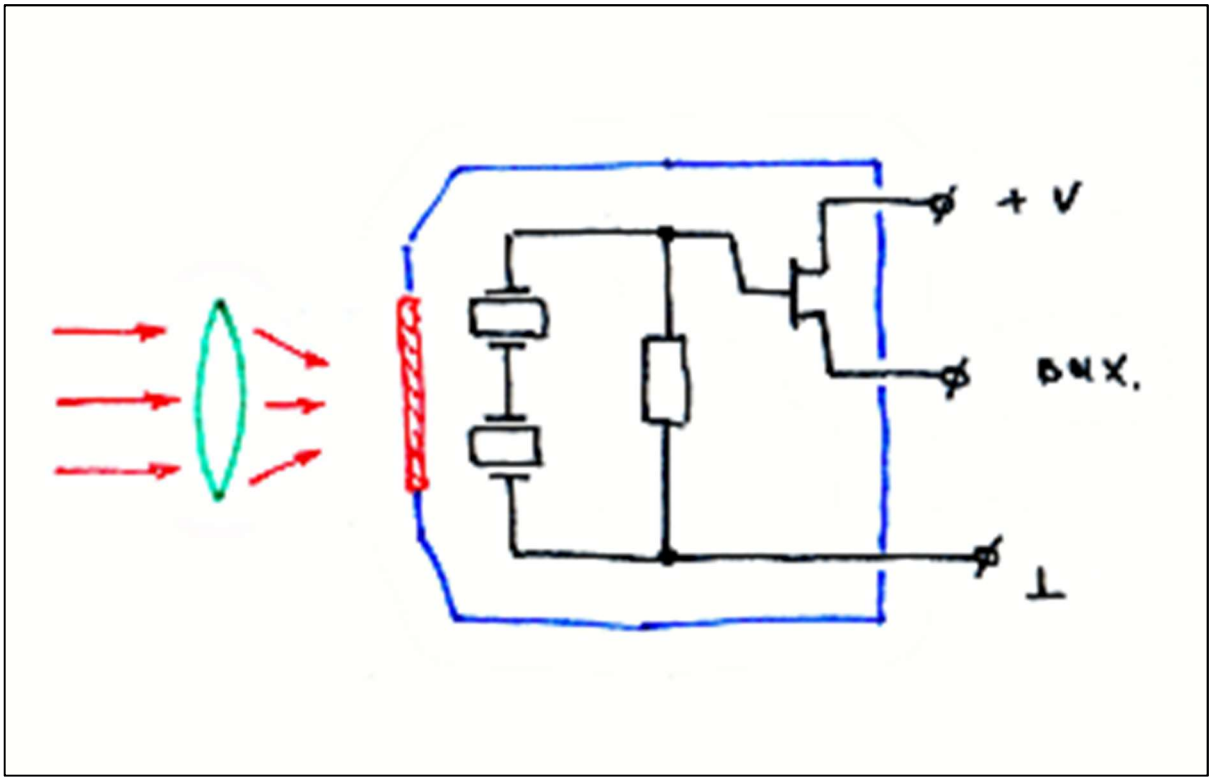

Рисунок 3.2 – Принцип работы датчика обнаружения присутствия

3.3.3 Датчик температуры и влажности

Многие датчики, которые были рассмотрены, имели конструктивную сборку, в которой присутствует датчик температуры и влажности вместе. Поэтому необходимо подобрать датчик, имеющий сразу оба датчика. Интерфейс взаимодействия датчика и микроконтроллера цифровой. Возможно использование SPI интерфейса для взаимодействия с датчиком. Данное взаимодействие будет обрабатываться при помощи настройки модуля STM32 посредством сервера и клиентского приложения. Предполагаемое питание датчика должно входить в диапазон от 1.8 до 5 вольт.

Датчик температуры должен использовать NTC-термистор, который является в свою очередь переменным резистором, который изменяет свое сопротивление при изменении температуры. Термистор представлен на рисунке 3.3.

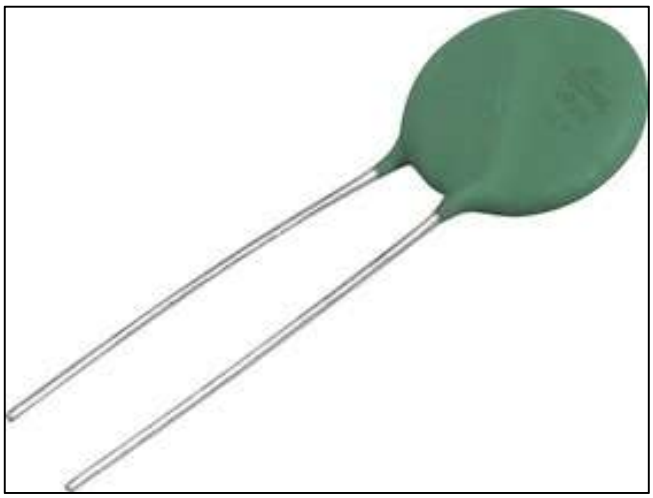

Рисунок 3.3 – Термистор [17]

Датчик влажности представляет из себя чувствительный элемент влажности, который должен использовать полимерную пленку, которая является чувствительной к влажности.

3.3.4 Датчик CO2

Датчики CO2 являются важным датчиком, основное отличие различных датчиков этого типа в том, каким образом они измеряют концентрацию CO2 в помещении. Важно чтобы датчик CO2 мог определять наличие газа минимум в диапазоне от 0 до 2000 ppm. А также, данный датчик должен иметь возможность передавать данные по последовательному интерфейсу, для упрощения работы с датчиком.

Допустимые нормы ppm представлены на рисунке 3.4.

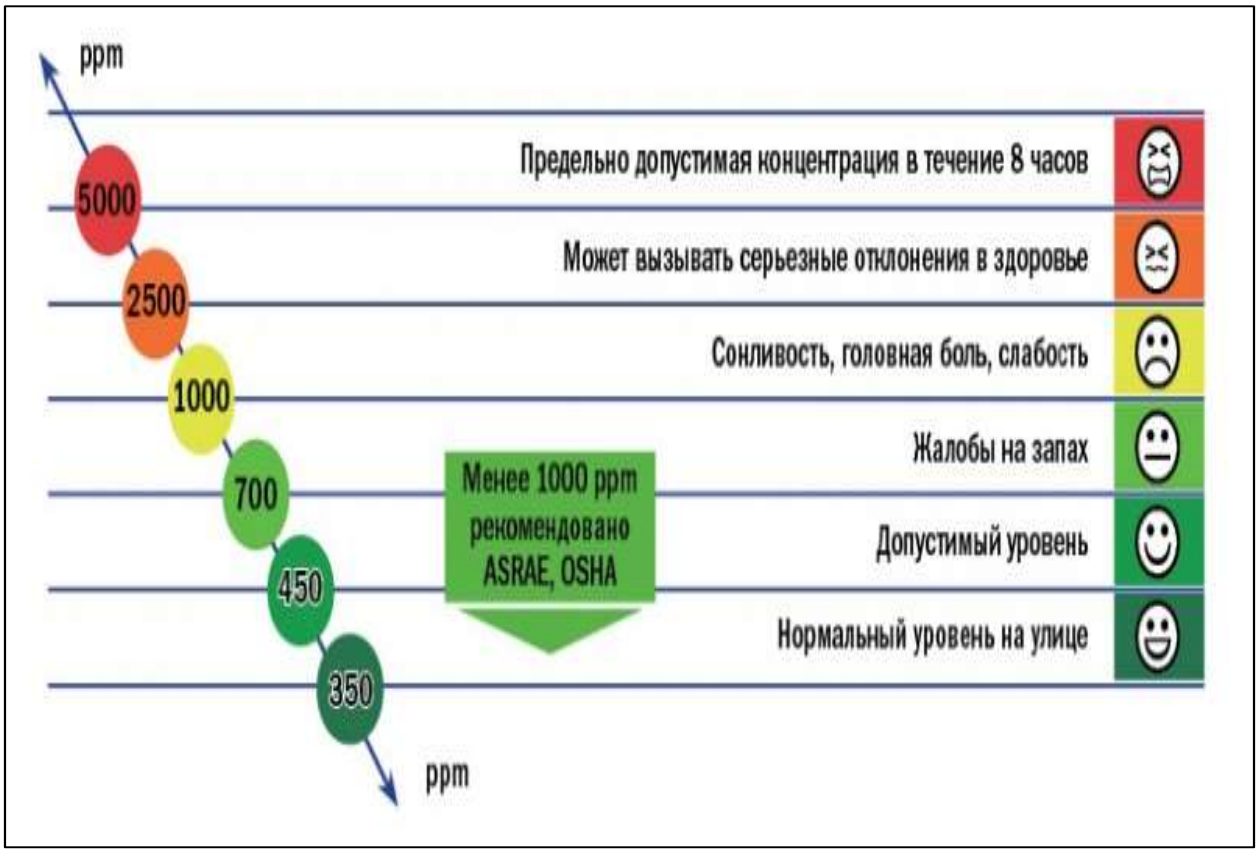

Рисунок 3.4 – Графики допустимых значений ppm [18]

Исходя из этих данных, датчик, который необходимо будет установить, как и планировалось, должен попадать в диапазон до 2000 ppm, так как большая концентрация уже является не нормальной для обычного пребывания в помещении.

3.4 Блок беспроводного интерфейса

Блок беспроводного интерфейса представляет из себя устройство, которое сможет обеспечить передачу данных между микроконтроллером, с которым он будет коммутирован, и датчиками, и исполнительными устройствами, которые будут находиться удаленно от основного устройства. Предполагаемый интерфейс взаимодействия микроконтроллера и устройства передачи данных по беспроводному соединению – SPI интерфейс. Данный интерфейс простой в реализации и поддерживается многими контроллерами. Питание для беспроводного интерфейса также должно быть в диапазоне питания микроконтроллера от 1.8 до 3.6 вольт.

3.5 Блок беспроводных датчиков

Блок беспроводных датчиков представляет из себя набор датчиков, которые будут подключены к микроконтроллеру по беспроводному

интерфейсу. Необходимо обозначить какие датчики будут находиться удаленно от микроконтроллера ввиду своих особенностей использования. Также необходимо определиться каким образом данные с датчиков будут передаваться по беспроводному интерфейсу.

Для того чтобы передавать данные от датчиков, подключенных по беспроводному интерфейсу предполагается использование датчика, который будет подключен к основному микроконтроллеру. Из этого следует, что необходимо наличие микроконтроллера при датчике, чтобы считывать и обрабатывать значение датчиков, а затем передавать их основному микроконтроллеру. Микроконтроллер должен иметь возможность передавать данные по SPI интерфейсу и иметь АЦП преобразователь, чтобы иметь возможность преобразовать значение, получаемое от аналоговых датчиков. Рассмотрим датчики, которые планируется подключать в разрабатываемой системе:

– датчик влажности;

– датчик присутствия.

Функциональные особенности датчика присутствия была рассмотрены в подразделе 3.3.2. Поэтому рассмотрим лишь особенности датчика влажности. Датчик влажности должен будет иметь возможность определять уровень влажности в точке помещения. Питание датчика должно быть согласовано с питанием микроконтроллера, к которому он будет подключен. Все беспроводные датчики и их обвязка должны будут иметь автономное питание, чтобы их можно было установить в любой точке комнаты, в зоне досягаемости беспроводного интерфейса.

При проектировании беспроводных датчиков необходимо также обеспечить данные датчики достаточным и необходимым аккумулятором для питания всего устройства.

3.6 Блок управления

Блок управления – это устройство, которое выполняет ряд задач, в результате выполнения которых осуществляется функционирование всех блоков в системе:

– сбор цифровых данных датчиков;

– сбор аналоговых данных датчиков;

– сбор цифровых данных датчиков, подключенных по беспроводному интерфейсу;

– передача данных по интерфейсу взаимодействия с ПК;

– передача команд исполнительным устройствам;

Исходя из того, какие связи есть у микроконтроллера необходимо рассмотреть данные подсистемы со стороны STM32F4XX.

Далее рассмотрим каждую из задач с функциональной точки зрения.

3.6.1 Подсистема RCC

Подсистема RCC отвечает за сброс состояния системы, сброс питания, и настройку источников синхронизации в устройстве и периферийных системах:

– тактирование устройств шины AHB1;
– тактирование устройств шины AHB2;
– тактирование устройств шины APB1;
– тактирование устройств шины APB2.

Так как система тактирования в микроконтроллерах STM32 имеет древовидную структуру, которая представлена на рисунке 3.5, необходимо отдельно включать тактирование той или иной системы, чтобы иметь возможность с ней работать. Исходя из рисунка, необходимо также правильно выставить делители и значения регистров, которые отвечают за то, какая частота будет получена на выходе той или иной ветки тактирования.

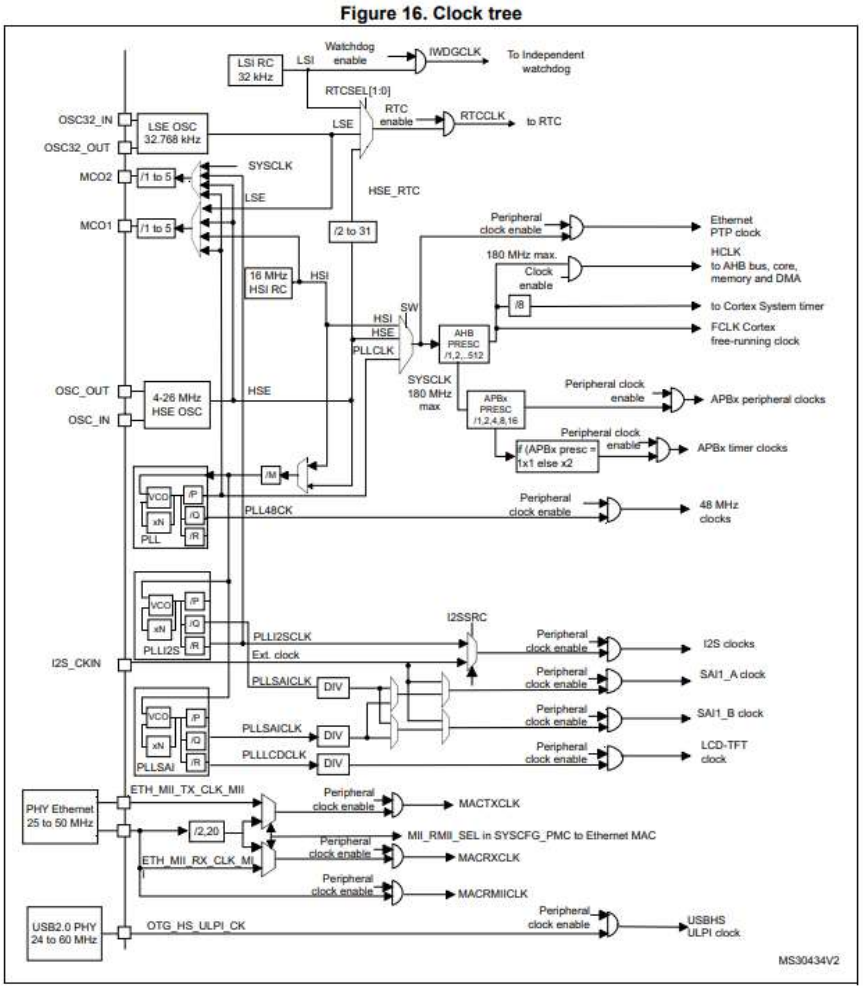

Рисунок 3.5 – Представление системы тактирования STM32F4XX [13]

3.6.1.1 Настройка тактирования шины AHB1

После сброса питания, тактирование данной шины отключено по умолчанию, однако доступно программное включение тактирования. Функция `RCC_AHB1PeriphClockCmd` предназначена для включения или отключения тактирования периферийной шины AHB1. Функция ничего не возвращает, но принимает такие следующие аргументы:

– `uint32_t RCC_AHB1Periph`;

– `FunctionalState NewState`;

Аргумент `RCC_AHB1Periph` имеет тип `uint32_t`, который определен в стандартной библиотеке `stdint.h` языка С. Данный аргумент задает периферийное устройство шины AHB1 для которого будет включено тактирование. Этот параметр может быть любой комбинацией значений, представленных в таблице 3.1.

Таблица 3.1

Значение параметра	Тактируемая периферия
1	2
`RCC_AHB1Periph_GPIOA`	Порт ввода/вывода GPIOA
`RCC_AHB1Periph_GPIOB`	Порт ввода/вывода GPIOB
`RCC_AHB1Periph_GPIOC`	Порт ввода/вывода GPIOC
`RCC_AHB1Periph_GPIOD`	Порт ввода/вывода GPIOD
`RCC_AHB1Periph_GPIOE`	Порт ввода/вывода GPIOE
`RCC_AHB1Periph_GPIOF`	Порт ввода/вывода GPIOF
`RCC_AHP1Periph_GPIOG`	Порт ввода/вывода GPIOG
`RCC_AHB1Periph_GPIOI`	Порт ввода/вывода GPIOI
`RCC_AHB1Periph_CRC`	Подсистема CRC
`RCC_AHB1Periph_DMA1`	Устройство DMA1
`RCC_AHB1Periph_DMA2`	Устройство DMA2
`RCC_AHB1Periph_ETH_MAC`	Ethernet MAC
`RCC_AHB1Periph_ETH_MAC_TX`	Ethernet Transmission

Продолжение таблицы 3.1

1	2
RCC_AHB1Periph_ETH_MAC_TX	Ethernet Transmission
RCC_AHB1Periph_ETH_MAC_RX	Ethernet Reception
RCC_AHB1Periph_OTG_HS	USB OTG HS

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение тактирование одной или нескольких периферийных систем.

3.6.1.2 Настройка тактирования шины AHB2

После сброса питания, тактирование данной шины отключено по умолчанию, однако доступно программное включение тактирования. Функция RCC_AHB2PeriphClockCmd предназначена для включения или отключения тактирования периферийной шины AHB2. Функция ничего не возвращает, но принимает такие следующие аргументы:

– uint32_t RCC_AHB2Periph;

– FunctionalState NewState;

Аргумент RCC_AHB2Periph имеет тип uint32_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент задает периферийное устройство шины AHB2 для которого будет включено тактирование. Этот параметр может быть любой комбинацией значений, представленных в таблице 3.2.

Таблица 3.2

Значение параметра	Тактируемая периферия
RCC_AHB2Periph_DCMI	DCMI
RCC_AHB2Periph_CRYP	CRYP
RCC_AHB2Periph_HASH	HASH
RCC_AHB2Periph_RNG	RNG
RCC_AHB2Periph_OTG_FS	USB OTG FS

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение тактирование одной или нескольких периферийных систем.

3.6.1.3 Настройка тактирования шины APB1

После сброса питания, тактирование данной шины отключено по умолчанию, однако доступно программное включение тактирования. Функция `RCC_APB1PeriphClockCmd` предназначена для включения или отключения тактирования периферийной шины APB1. Функция ничего не возвращает, но принимает такие следующие аргументы:

– `uint32_t RCC_APB1Periph`;

– `FunctionalState NewState`;

Аргумент `RCC_APB1Periph` имеет тип `uint32_t`, который определен в стандартной библиотеке `stdint.h` языка С. Данный аргумент задает периферийное устройство шины APB1 для которого будет включено тактирование. Этот параметр может быть любой комбинацией значений, представленных в таблице 3.3.

Таблица 3.3

Значение параметра	Тактируемая периферия
`RCC_APB1Periph_TIM6`	Таймер TIM6
`RCC_APB1Periph_TIM7`	Таймер TIM7
`RCC_APB1Periph_SPI2`	Интерфейс SPI2
`RCC_APB1Periph_SPI3`	Интерфейс SPI3
`RCC_APB1Periph_USART2`	USART2
`RCC_APB1Periph_USART3`	USART3
`RCC_APB1Periph_WWDG`	WWDG

Аргумент `NewState` имеет тип `FunctionalState`, который может принимать два значения: `ENABLE` и `DISABLE`. Данный аргумент отвечает за разрешение тактирование одной или нескольких периферийных систем.

3.6.1.3 Настройка тактирования шины APB2

После сброса питания, тактирование данной шины отключено по умолчанию, однако доступно программное включение тактирования. Функция `RCC_APB2PeriphClockCmd` предназначена для включения или отключения тактирования периферийной шины APB2. Функция ничего не возвращает, но принимает такие следующие аргументы:

– uint32_t RCC_APB2Periph;
– FunctionalState NewState;

Аргумент RCC_APB1Periph имеет тип uint32_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент задает периферийное устройство шины APB1 для которого будет включено тактирование. Этот параметр может быть любой комбинацией значений, представленных в таблице 3.4.

Таблица 3.4

Значение параметра	Тактируемая периферия
RCC_APB2Periph_ADC1	ADC1
RCC_APB2Periph_ADC2	ADC2
RCC_APB2Periph_ADC2	ADC3
RCC_APB2Periph_EXTIIT	EXTIIT
RCC_APB1Periph_SYSCFG	SYSCFG

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение тактирование одной или нескольких периферийных систем.

3.6.2 Настройка портов ввода-вывода

Система портов ввода-вывода в микроконтроллерах STM32F4XX представляет из себя сложную систему, представленную на рисунке 3.6, которая может работать в различных режимах. Один из режимов работы – это работа как порт ввода данных, почти все порты могут быть назначены как аналоговые порты ввода данных, которые можно в дальнейшем преобразовать в цифровой сигнал. Также данные порты могут быть сконфигурированы как цифровые порты и системы микроконтроллера будут воспринимать сигнал, полученный с данного порта, как цифровой сигнал высокого или низкого уровня. Следующий режим работы – это работа как порт вывода данных, микроконтроллеры могут выставлять высокий или низкий уровень сигнала на порт вывода для работы с какой-либо внешней периферией. Третий режим работы – режим работы в качестве альтернативной функции, что значит, что порт будет переназначен для работы от внутренней периферии и она будет выставлять необходимые уровни на данном порту или считывать данные с него по протоколу, которые использует данная периферия.

Для того чтобы микроконтроллер мог работать с некоторыми

периферийными устройствами необходимо правильно сконфигурировать порты ввода-вывода и включить тактирование используемых портов. Для этого используются следующие функции системы:

– функции инициализации и конфигурации порта;

– функции чтения и записи;

– функция конфигурации порта на альтернативную функцию.

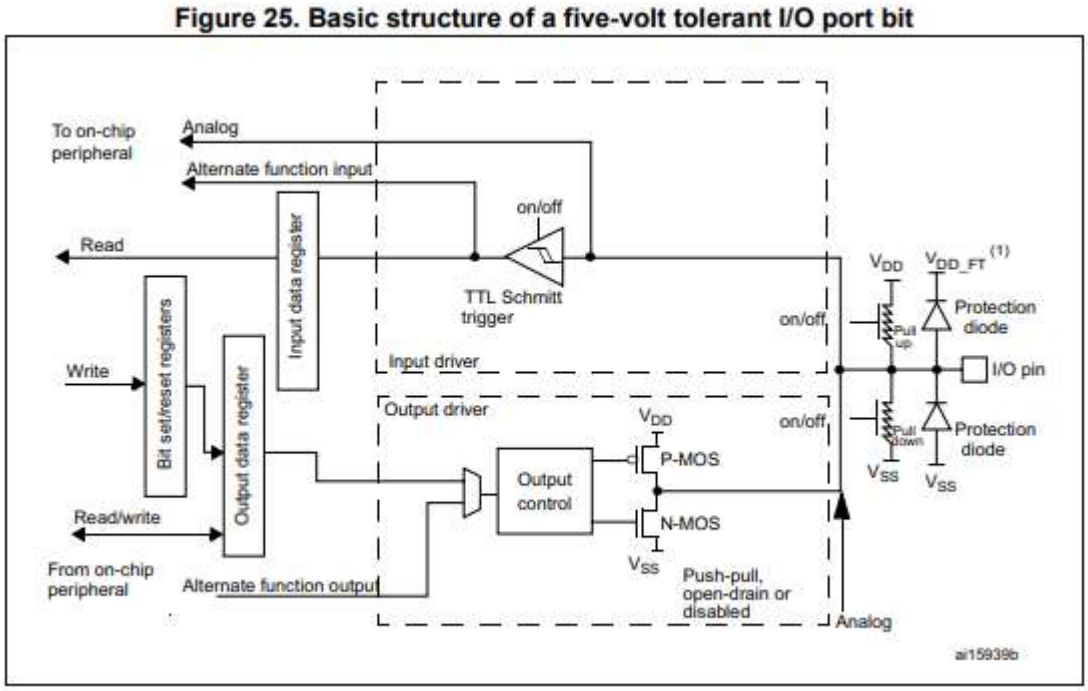

Рисунок 3.6 – Схема работы цифро-аналогового порта ввода-вывода [13]

3.6.2.1 Функции инициализации и конфигурации порта

Перед тем как произвести инициализацию определенного порта периферии необходимо включить тактирование данного порта используя функцию `RCC_AHB1PeriphClockCmd`, параметр `RCC_APB1Periph` указать в соответствии с портом, который необходимо проинициализировать и `FunctionalState` с значением `ENABLE`. Для того чтобы сконфигурировать определенный порт периферии используется функция `GPIO_Init`. Функция ничего не возвращает, но принимает такие параметры:

– `GPIO_TypeDef* GPIOx`;

– `GPIO_InitTypeDef* GPIO_InitStruct`.

Аргумент `GPIOx` имеет тип `GPIO_TypeDef*` и является указателем на регистр порта, определенный в заголовочном файле `stm32f4xx.h`.

Аргумент `GPIO_InitStruct` имеет тип `GPIO_InitTypeDef*`, который является указателем на структуру, определенную в заголовочном

файле stm32f4xx_gpio.h. По указателю хранятся конфигурационные настройки того, какой вывод был сконфигурирован, его скоростные параметры, режим работы, тип вывода и наличие подтягивающего резистора.

3.6.2.2 Функции чтения и записи

После того как порт был сконфигурирован, системе необходимо иметь возможность записывать значение в порт или же читать состояние порта.

Для того чтобы иметь возможность записать новое значение в определенный вывод порта используется функция GPIO_SetBits. Данная функция использует GPIOx_BSRR регистр, который доступен для чтения или изменения. Функция ничего не возвращает, но принимает такие параметры:

– GPIO_TypeDef* GPIOx;

– uint16_t GPIO_Pin.

Аргумент GPIOx имеет тип GPIO_TypeDef* и является указателем на регистр порта, определенный в заголовочном файле stm32f4xx.h.

Аргумент GPIO_Pin имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент может принимать значение от GPIO_Pin_x, где x – может принимать значения от 0 до 15.

Для того чтобы иметь возможность сбросить значение определенного вывода порта используется функция GPIO_ResetBits. Данная функция использует GPIOx_BSRR регистр, который доступен для чтения или изменения. Функция ничего не возвращает, но принимает такие параметры:

– GPIO_TypeDef* GPIOx;

– uint16_t GPIO_Pin.

Аргумент GPIOx имеет тип GPIO_TypeDef* и является указателем на регистр порта, определенный в заголовочном файле stm32f4xx.h.

Аргумент GPIO_Pin имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент может принимать значение от GPIO_Pin_x, где x – может принимать значения от 0 до 15.

Если вывод порта сконфигурирован на прием данных с порта, то к нему применима функция чтения состояния вывода порта GPIO_ReadInputDataBit. Данные функция возвращает значение bitstatus имеющее тип uint8_t, который определен в стандартной библиотеке stdint.h языка С. Данная функция также принимает такие аргументы:

– GPIO_TypeDef* GPIOx;

– uint16_t GPIO_Pin.

Аргумент GPIOx имеет тип GPIO_TypeDef* и является указателем на

регистр порта, определенный в заголовочном файле stm32f4xx.h.

Аргумент GPIO_Pin имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент может принимать значение от GPIO_Pin_x, где x – может принимать значения от 0 до 15.

3.6.2.3 Функция настройки порта на альтернативную функцию

Конфигурация порта на другие периферийные устройства микроконтроллера осуществляется посредством функции GPIO_PinAFConfig. Данная функция ничего не возвращает, но принимает такие параметры:

– GPIO_TypeDef* GPIOx;
– uint16_t GPIO_PinSource;
– uint8_t GPIO_AF.

Аргумент GPIOx имеет тип GPIO_TypeDef* и является указателем на регистр порта, определенный в заголовочном файле stm32f4xx.h.

Аргумент GPIO_PinSource имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент необходим для того чтобы выбрать вывод, который будет сконфигурирован для альтернативной функции и может принимать значение от GPIO_PinSourcex, где x – может принимать значения от 0 до 15.

Аргумент GPIO_AF имеет тип uint8_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент выбирает какую альтернативную функцию будет использовать вывод. Значения, которые может использоваться для данного аргумента можно узнать в заголовочном файле stm32fxx_gpio.h.

3.6.3 Подсистема WWDG

Сторожевой таймер используется для обнаружения сбоя программного обеспечения, обычно сгенерированного внешними помехами или необработанными логическими условиями в программе, которая перестает работать в нормальном режиме. Система сторожевого таймера, представленная на рисунке 3.7, генерирует сброс MCU по истечению запрограммированного периода времени, если только программа не обновит содержимое счетчика до того, как бит T6 будет очищен. Сброс MCU также генерируется, если 7-разрядное значение счетчика с направлением счета вниз (в регистре управления) обновится до того, как это значение достигнет значения контрольного регистра.

Для того чтобы активировать сторожевой таймер необходимо использовать функцию WWDG_Enable. Данная функция разрешает работу сторожевого таймера и загружает значение счетчика сторожевого таймера.

Функция не возвращает никакого значения, но принимает аргумент:

– `uint8_t Counter`.

Аргумент `Counter` имеет тип `uint8_t`, который определен в стандартной библиотеке `stdint.h` языка С. Данный аргумент является начальным значением сторожевого таймера. Диапазон значений данного аргумента от 0x40 до 0x7F, значения, выходящие за границы, могут сгенерировать немедленный сброс.

Для установки граничного значения для сторожевого таймера используется функция `WWDG_SetWindowValue`. Данная функция ничего не возвращает, но принимает такой аргумент:

– `uint8_t WindowValue`.

Аргумент `WindowValue` имеет тип `uint8_t`, который определен в стандартной библиотеке `stdint.h` языка С. Данный аргумент устанавливает граничное значение, с которым будет сравниваться значение счетчика с направлением счета вниз. Аргумент должен иметь значение меньше 0x80.

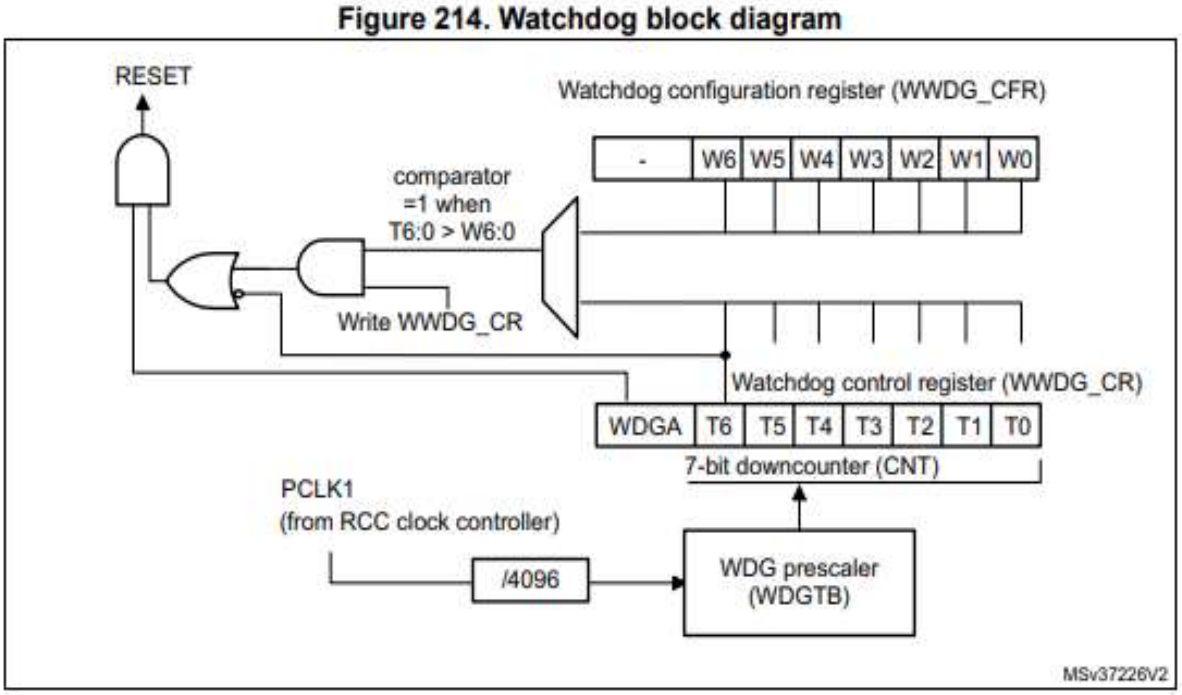

Рисунок 3.7 – Блок диаграмма сторожевого таймера [13]

3.6.4 Подсистема таймеров TIM

Подсистема таймеров включает в себя множество таймеров, которые можно использовать для работы других подсистем микроконтроллера, но для работы с ними используются одинаковые методы для инициализации и установки некоторых базовых полей, а также генерация прерываний,

установка флагов таймера и их чтение. Рассмотрим основные функции для работы с данной подсистемой.

Для работы с периферийными таймерами реализованы функции:

– инициализации и конфигурации;

– работы с прерываниями;

3.6.4.1 Функции инициализации и конфигурации

Для инициализации таймера в микроконтроллере необходимо использовать функцию `TIM_TimeBaseInit`, которая инициализирует периферийное устройство TIMX Time Base Unit согласно указанным параметрам в структуре, которая является аргументов в функции. Данная функция ничего не возвращает, но принимает такие аргументы:

– `TIM_TypeDef* TIMx`;

– `TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct`.

Аргумент `TIMx` имеет тип `TIM_TypeDef*` и является указателем на регистр порта, определенный в заголовочном файле `stm32f4xx.h`. Значение x в `TIMx` имеет диапазон от 1 до 14, в зависимости от выбранного периферийного таймера.

Аргумент `TIM_TimeBaseInitStruct` имеет тип `TIM_TimeBaseInitTypeDef*`, который является указателем на структуру, которая содержит конфигурационную информацию для периферийного таймера.

Для настройки режима работы счетчика используется функция `TIM_CounterModeConfig`. Данная функция ничего не возвращает, но принимает такие аргументы:

– `TIM_TypeDef* TIMx`;

– `uint16_t TIM_CounterMode`.

Аргумент `TIMx` имеет тип `TIM_TypeDef*` и является указателем на регистр порта, определенный в заголовочном файле `stm32f4xx.h`. Значение x в `TIMx` имеет значения 1, 2, 3, 4, 5, 8, в зависимости от выбранного периферийного таймера.

Аргумент `TIM_CounterMode` имеет тип `uint16_t`, который определен в стандартной библиотеке `stdint.h` языка C. Данный аргумент может принимать несколько значений: `TIM_CounterMode_Up` – для установки режима счета вверх, `TIM_CounterMode_Down` – для установки режима счета вниз и три режима работы счета счетчика с выравниванием по центру `TIM_CounterMode_CenterAlignedX`, где X – значение от 1 до 3.

Для того чтобы установить новое значение в регистр счетчика используется функция `TIM_SetCounter`. Данная функция ничего не возвращает, но принимает такие аргументы:

– `TIM_TypeDef* TIMx`;

– uint32_t Counter.

Аргумент TIMx имеет тип TIM_TypeDef* и является указателем на регистр порта, определенный в заголовочном файле stm32f4xx.h. Значение x в TIMx имеет диапазон от 1 до 14, в зависимости от выбранного периферийного таймера.

Аргумент Counter имеет тип uint32_t, который определен в стандартной библиотеке stdint.h языка C. Данный аргумент указывает новое значение счетчика после выполнения данной функции.

Для того чтобы счетчик продолжил автоматически свою работу после того, как досчитает до нужного значения, используется метод TIM_SetAutoreload. Данный метод ничего не возвращает, но принимает такие параметры:

– TIM_TypeDef* TIMx;

– uint32_t Autoreload.

Аргумент TIMx имеет тип TIM_TypeDef* и является указателем на регистр порта, определенный в заголовочном файле stm32f4xx.h. Значение x в TIMx имеет диапазон от 1 до 14, в зависимости от выбранного периферийного таймера.

Аргумент Autoreload имеет тип uint32_t, который определен в стандартной библиотеке stdint.h языка C. Данный аргумент указывает новое значение регистра автозагрузки.

Для запуска или же остановки работы периферийного счетчика необходимо использовать функцию TIM_Cmd. Данная функция ничего не возвращает, но принимает такие параметры:

– TIM_TypeDef* TIMx;

– FunctionalState Newstate.

Аргумент TIMx имеет тип TIM_TypeDef* и является указателем на регистр порта, определенный в заголовочном файле stm32f4xx.h. Значение x в TIMx имеет диапазон от 1 до 14, в зависимости от выбранного периферийного таймера.

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение работы периферийного таймера, указанного в первом аргументе.

3.6.4.1 Функции работы с прерываниями

Для того чтобы не использовать постоянный опрос флагов таймеров, можно использовать прерывания, доступные для работы с ними, чтобы разрешить определенному периферийному таймеру использовать подсистему прерываний необходимо использовать функцию TIM_ITConfig. Данная функция ничего не возвращает, но принимает такие параметры:

– TIM_TypeDef* TIMx;
– uint16_t TIM_IT;
– FunctionalState NewState.

Аргумент TIMx имеет тип TIM_TypeDef* и является указателем на регистр порта, определенный в заголовочном файле stm32f4xx.h. Значение x в TIMx имеет диапазон от 1 до 14, в зависимости от выбранного периферийного таймера.

Аргумент TIM_IT имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка C. Данный аргумент выбирает источник прерывания или несколько источников.

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение использования прерывания, указанного в аргументе TIM_IT.

3.6.5 Аналого-цифровой преобразователь

Для обработки данных с датчиков, которые отправляют аналоговый сигнал на микроконтроллер используется аналого-цифровой преобразователь, блок диаграмма которого представлена на рисунке 3.8. В микроконтроллерах, используемой серии, в периферии присутствует 12-разрядный АЦП с последовательным приближением. Данный АЦП имеет 19 каналов, позволяющих измерять сигналы от 16 внешних источников. Аналого-цифровое преобразование каналов может быть выполнено в одиночном, непрерывном, сканирующем или прерывистом режимах. Результат сохраняется в 16-разрядном регистре данных с выравнивание по левому краю. Функции аналогового сторожевого таймера позволяют приложению определять, падает ли напряжение на входе за пределами, определенными пользователем, выше или ниже пороговых значений.

Для работы АЦП реализованы функции:
– инициализации и конфигурации;
– преобразования;
– работы с прерываниями.

3.6.5.1 Функции инициализации и конфигурации

Для того, чтобы начать с АЦП, необходимо сконфигурировать корректно канал, по которому будет подключен внешний источник, разрешить преобразование данных, режим работы АЦП. Для этого используется функция ADC_Init. Данная функция ничего не возвращает, но принимает такие значения:

– ADC_TypeDef* ADCx;
– ADC_InitTypeDef* ADC_InitStruct.

Аргумент ADCx имеет тип ADC_TypeDef* и является указателем на регистр порта АЦП для которого производится инициализация, определенный в заголовочном файле stm32f4xx.h, x – может принимать значение от 1 до 3.

Аргумент ADC_InitStruct имеет тип ADC_InitTypeDef*, который является указателем на структуру, содержащую конфигурационную информацию для АЦП, выбранного в предыдущем параметре.

Для того чтобы запустить работы АЦП после инициализации используется функция ADC_Cmd. Данная функция ничего не возвращает, но принимает такие параметры:

– ADC_TypeDef* ADCx;

– FunctionalState NewState.

Аргумент ADCx имеет тип ADC_TypeDef* и является указателем на регистр порта АЦП для которого производится инициализация, определенный в заголовочном файле stm32f4xx.h, x – может принимать значение от 1 до 3.

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение работы АЦП, который был выбран в предыдущем аргументе.

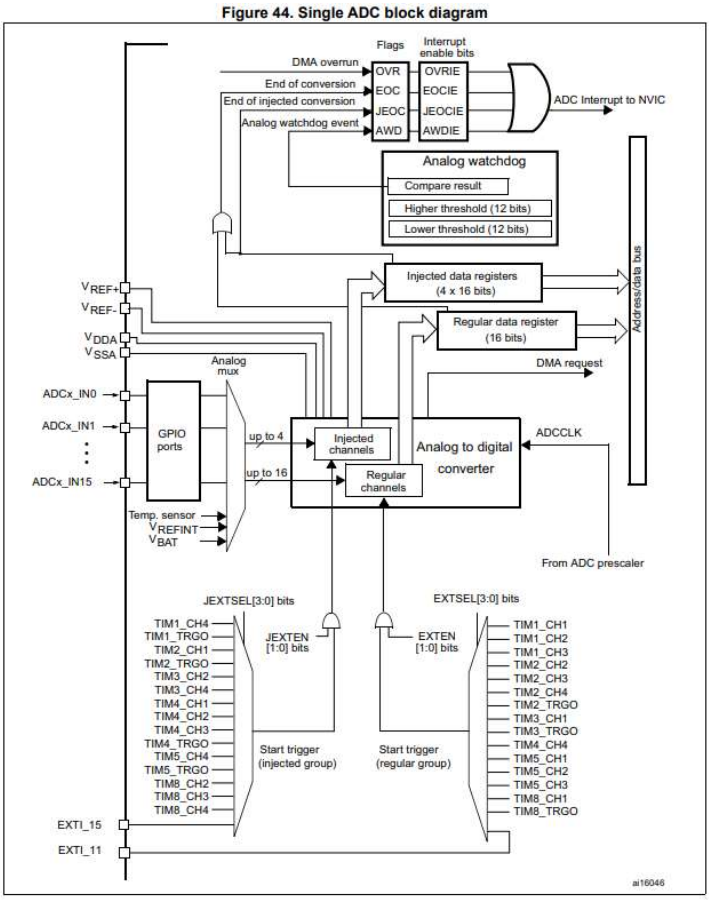

Рисунок 3.8 – Блок диаграмма АЦП STM32F4XX [13]

3.6.5.2 Функции преобразования

Для того чтобы разрешить преобразование обычных каналов выбранному АЦП с помощью программных средств используется функция ADC_SoftwareStartConv. Данная функция ничего не возвращает, но принимает такие параметры:

– ADC_TypeDef* ADCx.

Аргумент ADCx имеет тип ADC_TypeDef* и является указателем на регистр порта АЦП для которого производится инициализация, определенный в заголовочном файле stm32f4xx.h, x – может принимать значение от 1 до 3.

После того как было разрешено преобразование обычных каналов выбранному АЦП, необходимо контролировать момент, когда значения будут получены, для этого используется функция ADC_GetSoftwareStartStatus. Данная функция возвращает статус флага АЦП, если преобразование закончено, то статус будет иметь значение SET, иначе RESET. Данная функция также принимает такие параметры:

– ADC_TypeDef* ADCx.

– Аргумент ADCx имеет тип ADC_TypeDef* и является указателем на регистр порта АЦП для которого производится инициализация, определенный в заголовочном файле stm32f4xx.h, x – может принимать значение от 1 до 3.

Для того чтобы включить или отключить режим непрерывного преобразования АЦП используется функция ADC_ContinuousModeCmd. Данная функция ничего не возвращает, но принимает такие параметры:

– ADC_TypeDef* ADCx;

– FunctionalState NewState.

Аргумент ADCx имеет тип ADC_TypeDef* и является указателем на регистр порта АЦП для которого производится инициализация, определенный в заголовочном файле stm32f4xx.h, x – может принимать значение от 1 до 3.

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение работы АЦП в режиме непрерывного преобразования, который был выбран в предыдущем аргументе.

Для того чтобы получить последнее преобразованное значение для регулярных каналов для АЦП с выбранным номером, используется функция ADC_GetConversionValue. Данная функция возвращает конвертированное значение, данное значение имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка С. Принимает такие параметры:

– ADC_TypeDef* ADCx.

Аргумент ADCx имеет тип ADC_TypeDef* и является указателем на регистр порта АЦП для которого производится инициализация, определенный в заголовочном файле stm32f4xx.h, x – может принимать значение до 3.

3.6.5.3 Функции работы с прерываниями

Для того чтобы выбрать необходимое прерывание и включить его или отключить используется функция ADC_ITConfig. Данная функция ничего не возвращает, но принимает такие параметры:

– ADC_TypeDef* ADCx;

– uint16_t ADC_IT;

– FunctionalState NewState.

Аргумент ADCx имеет тип ADC_TypeDef* и является указателем на регистр порта АЦП для которого производится конфигурация прерываний, определенный в заголовочном файле stm32f4xx.h, x – может принимать значение от 1 до 3.

Аргумент ADC_IT имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент выбирает источник прерывания или несколько источников.

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение использования прерывания, указанного в аргументе ADC_IT.

Для получения состояния флага прерывания АЦП используется метод ADC_GetFlagStatus. Данная функция возвращает значение FlagStatus с типом boolean, который определен в стандартной библиотеке stdint.h языка С. Принимает такие параметры:

– ADC_TypeDef* ADCx;

– uint8_t ADC_FLAG.

Аргумент ADCx имеет тип ADC_TypeDef* и является указателем на регистр порта АЦП для которого производится чтение флага, определенный в заголовочном файле stm32f4xx.h, x – может принимать значение от 1 до 3.

Аргумент ADC_FLAG имеет тип uint8_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент может принимать значения флагов, которые представлены в таблице 3.5.

Таблица 3.5

Аргумент	Источник прерывания
1	2
ADC_FLGA_AWD	Аналоговый сторожевой таймер

Продолжение таблицы 3.5

1	2
ADC_FLGA_AWD	Аналоговый сторожевой таймер
ADC_FLAG_EOC	Конец конвертации
ADC_FLAG_JEOC	Конец преобразования группы
ADC_FLAG_JSTRT	Старт преобразования группы
ADC_FLAG_STRT	Старт обычного преобразования группы
ADC_FLAG_OVR	Переполнение

Для того чтобы иметь возможность очистить флаг, используется функция ADC_ClearFlag. Данная функция ничего не возвращает, но принимает такие параметры:

– ADC_TypeDef* ADCx;

– uint8_t ADC_FLAG.

Аргумент ADCx имеет тип ADC_TypeDef* и является указателем на регистр порта АЦП для которого производится очистка флага, определенный в заголовочном файле stm32f4xx.h, x – может принимать значение от 1 до 3.

Аргумент ADC_FLAG имеет тип uint8_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент может принимать значения флагов, которые представлены в таблице 3.5.

3.6.6 Подсистема USART

Универсальный синхронный асинхронный приемник-передатчик (USART) предлагает гибкие средства для полнодуплексного обмена данными с внешним оборудованием, требующим промышленного стандарта NRZ - асинхронный формат последовательных данных. USART предлагает очень широкий диапазон скоростей передачи с использованием генератора скорости передачи данных. Поддерживает синхронную одностороннюю связь с полудуплексной однопроводной коммуникацией. Он также поддерживает LIN, Smartcard протокол. Высокоскоростная передача данных возможна при использовании DMA для конфигурации с несколькими буферами.

Для работы по USART интерфейсу реализованы функции:

– инициализации и конфигурации;
– передачи и приема данных;
– высоко скоростной передачи данных по DMA;
– полудуплексной передачи данных;
– управления прерываниями.

Система универсального синхронно асинхронного приемника-передатчика представлена на рисунке 3.9.

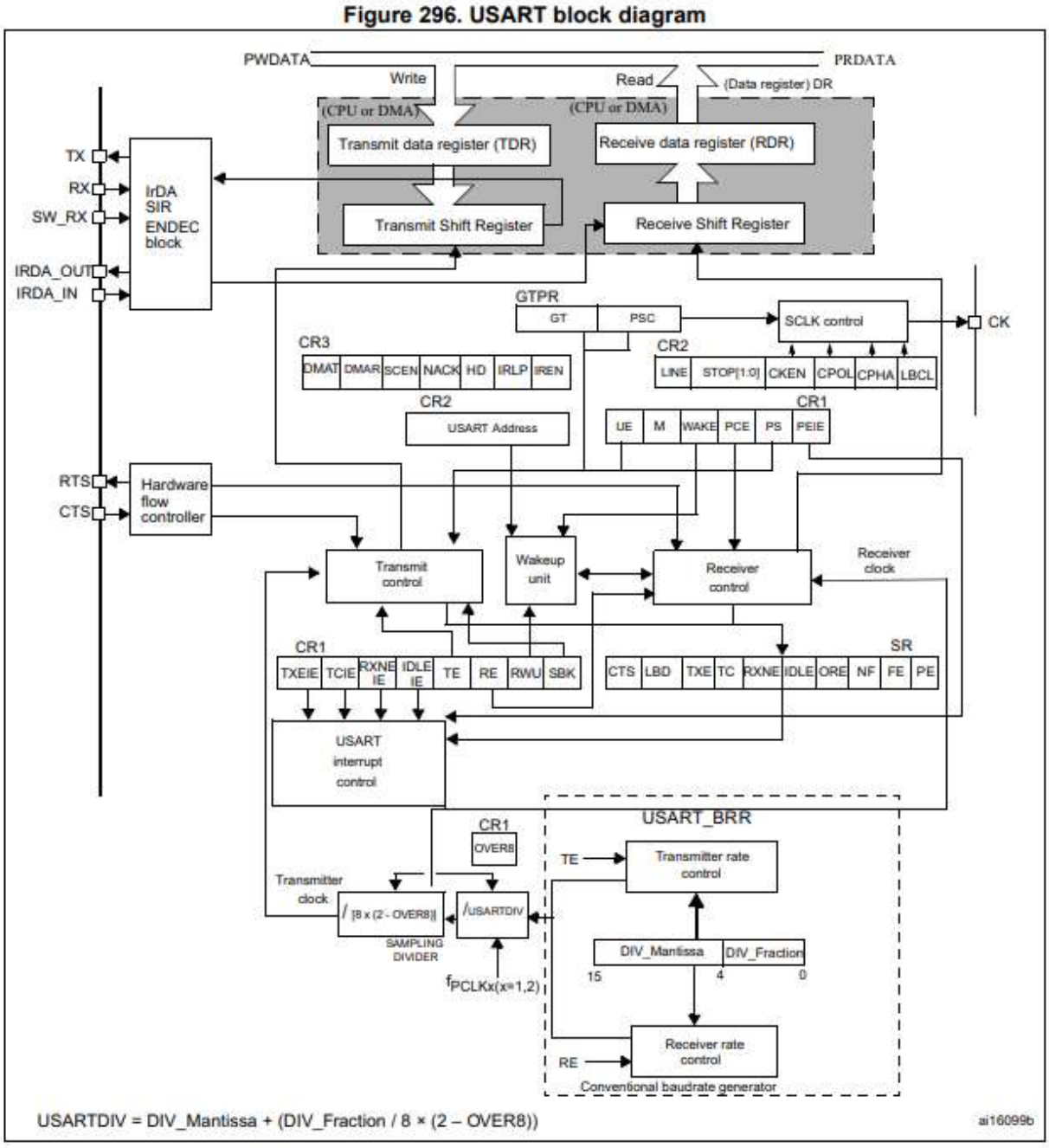

Рисунок 3.9 – Система ассинхронной приемопередачи данных STM32F4XX [13]

3.6.6.1 Функции инициализации и конфигурации

Для конфигурации параметров интерфейса USART используется функция USART_Init. Данной функцией можно сконфигурировать скорость передачи данных, длину слова, количество стоп битов, аппаратное управление потоком и режим работы(приемник/передатчик). Данная функция ничего не возвращает, но принимает такие параметры:

– USART_TypeDef* USARTx;

– USART_InitTypeDef* USART_InitStruct.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления портом USARTx для которого производится инициализация, определенный в заголовочном файле stm32f4xx.h.

Аргумент USART_InitStruct имеет тип USART_InitTypeDef*, который является указателем на структуру содержащую конфигурационную информацию специфичную для данного периферийного устройства USART.

Для того чтобы запустить работу устройства USART используется функция USART_Cmd. Данная функция ничего не возвращает, но принимает такие параметры:

– USART_TypeDef* USARTx;

– FunctionalState NewState.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления портом USARTx для которому выдается разрешение на работы, определенный в заголовочном файле stm32f4xx.h.

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение работы USARTx.

Для определения мастера и подчиненного устройству USART необходимо присвоить адрес, так как к устройству-мастеру может быть подключено на RX вход несколько подчиненных устройств и устройствам необходимо распознавать, что запрос пришел непосредственно к нему. Для этого используется функция USART_SetAddress. Данная функция ничего не возвращает, но принимает такие параметры:

– USART_TypeDef* USARTx;

– uint8_t USART_Address.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления

портом USARTx которому присваивается адрес, определенный в заголовочном файле stm32f4xx.h.

Аргумент USART_Address имеет тип uint8_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент является адресом, который будет передан периферийному устройству USARTx.

3.6.6.2 Функции приема и передачи данных

Для организации приема данных по периферийному интерфейсу USART используется функция USART_ReceiveData. Данная функция возвращает значение, которое имеет тип uint16_t, принятое по USART интерфейсу. Принимает такие аргументы:

– USART_TypeDef* USARTx.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления портом USARTx для которого производится прием данных, определенный в заголовочном файле stm32f4xx.h.

Для организации передачи данных по периферийному интерфейсу USART используется функция USART_SendData. Данная функция ничего не возвращает, но принимает такие аргументы:

– USART_TypeDef* USARTx;

– uint16_t Data.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления портом USARTx по которому производится передача данных, определенный в заголовочном файле stm32f4xx.h.

Аргумент Data имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент является данными, которые будут переданы по интерфейсу USART.

3.6.6.3 Функция полудуплексной передачи данных

Для организации полудуплексной передачи данных необходимо проинициализировать периферийное устройство посредством функции USART_Init. После этого установить адрес для данного устройства. Разрешить работу с помощью функции USART_Cmd. Далее необходимо использовать функцию разрешающую полудуплексную передачу данных USART_HalfDuplexCmd. Данная функция ничего не возвращает, но принимает такие параметры:

– USART_TypeDef* USARTx;

– FunctionalState NewState.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления портом USARTx по которому производится передача данных, определенный в заголовочном файле stm32f4xx.h.

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение работы USARTx.

3.6.6.4 Функции высоко скоростной передачи данных по DMA

Для разрешения или блокировки работы USART по DMA интерфейсу используется функция USART_DMACmd. Данная функция ничего не возвращает, но принимает такие параметры:

– USART_TypeDef* USARTx;

– uint16_t USART_DMAReq;

– FunctionalState NewState.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления портом USARTx по которому производится передача данных, определенный в заголовочном файле stm32f4xx.h.

Аргумент USART_DMAReq имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка C. Данный аргумент может иметь два значения:

– USART_DMAReq_Tx, когда осуществляется запрос на отправку данных по DMA интерфейсу;

– USART_DMAReq_Rx, когда осуществляется запрос на прием данных по DMA интерфейсу.

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение работы USARTx по DMA интерфейсу.

3.6.6.5 Функции работы с прерываниями

При работе по интерфейсу DMA и другим интерфейсам необходимо иметь возможность знать состояние USART. Для того чтобы выбрать необходимое прерывание и включить его или отключить используется

функция USART_ITConfig. Данная функция ничего не возвращает, но принимает такие параметры:

– USART_TypeDef* USARTx;

– uint16_t USART_IT;

– FunctionalState NewState.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления портом USARTx для которого конфигурируются прерывания, определенный в заголовочном файле stm32f4xx.h.

Аргумент USART_IT имеет тип uint16_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент выбирает источник прерывания или несколько источников.

Аргумент NewState имеет тип FunctionalState, который может принимать два значения: ENABLE и DISABLE. Данный аргумент отвечает за разрешение использования прерывания, указанного в аргументе USART_IT.

Для получения состояния флага прерывания АЦП используется метод USART_GetFlagStatus. Данная функция возвращает значение FlagStatus с типом boolean, который определен в стандартной библиотеке stdint.h языка С. Принимает такие параметры:

– USART_TypeDef* USARTx;

– uint8_t ADC_FLAG.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления портом USARTx для которого будет прочитано состояние флага, определенный в заголовочном файле stm32f4xx.h.

Аргумент USART_FLAG имеет тип uint8_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент может принимать значения флагов, которые представлены в таблице 3.6.

Таблица 3.6

Аргумент	Источник прерывания
1	2
USART_FLGA_CTS	Изменение флага CTS (недоступно для UART4 и UART5)
USART_FLAG_LBD	Флаг обнаружения прерывания LIN
USART_FLAG_TXE	Регистр количества данных для передачи пуст

Продолжение таблицы 3.6

1	2
USART_FLAG_TC	Флаг успешного завершения передачи данных
USART_FLAG_RXNE	Регистр количества данных для приема пуст
USART_FLAG_IDLE	Флаг обнаружения незанятой шины
USART_FLAG_ORE	Флаг ошибки OverRun
USART_FLAG_NE	Флаг ошибки дребезга
USART_FLAG_FE	Флаг ошибки фрэйма
USART_FLAG_PE	Флаг ошибки четности

Для того чтобы иметь возможность очистить флаг, используется функция USART_ClearFlag. Данная функция ничего не возвращает, но принимает такие параметры:

– USART _TypeDef* USART x;

– uint8_t USART _FLAG.

Аргумент USARTx, где x может быть 1, 2, 3, 4, 5, 6, 7 или 8, в зависимости от выбранного периферийного устройства USART, имеет тип USART_TypeDef*, который является указателем на регистр управления портом USARTx для которого будет очищено состояние флага, определенный в заголовочном файле stm32f4xx.h.

Аргумент USART_FLAG имеет тип uint8_t, который определен в стандартной библиотеке stdint.h языка С. Данный аргумент может принимать значения флагов, которые представлены в таблице 3.6.

3.7 Блок исполнительного устройства

Блок исполнительного устройства представляет из себя любое устройство, которое использует питание сети помещения, но к которому может быть подключен микроконтроллер, который посредством драйвера управления может управлять этим исполнительным устройством. Одним из предполагаемых к использованию устройством является шаговый двигатель, который может выполнять различные задачи в зависимости от места применения. Беспроводной интерфейс предполагается использовать такой же, как и при основном микроконтроллере.

Питание исполнительного устройства планируется сделать из общей сети, так как исполнительные устройства требуют большие силы тока, ввиду чего использование батареек является не рациональным.

4 РАЗРАБОТКА ПРИНЦИПИАЛЬНОЙ СХЕМЫ

На данном этапе будут выбраны конкретные модели интегральных микросхем, выбраны схемы их включения, рассчитаны вторичные источники питания, выбраны интерфейсы для сопряжения с MCU. На принципиальной схеме, представленной на чертеже ГУИР.400201.019 Э3, реализованы блоки коммутации микросхемы сервера и микроконтроллера, подключение датчиков к микроконтроллеру, а также коммутацию беспроводного интерфейса с микроконтроллерами, которые подключаются к беспроводным датчикам и исполнительным устройствам.

4.1 Обоснование выбора схемы и расчет дополнительных элементов

Основным элементом принципиальной схемы является микроконтроллер STM32. После того как были определены основные задачи, возложенные на данный микроконтроллер, было принято решение использовать контроллер STM32F407, который имеет все необходимые интерфейсы для взаимодействия с датчиками, которые будут подключены к нему, возможно взаимодействие с устройством передачи данных по беспроводному интерфейсу, а также есть возможность взаимодействия по RS-485 интерфейсу.

4.1.1 Преобразование питания

Так как входное напряжение, которое будет получено по шине сервера, будет иметь +24 В, необходимо преобразовать питание первоначально до 5 вольт. Для этих целей предполагается использовать DC/DC преобразователь TCR 1-2450. Выбор данного преобразователя питания обусловлен тем, что диапазон входных напряжений от 6.5 до 36 В, выход преобразователя 5В. Типичная схема подключения данного преобразователя представлена на рисунке 4.1.

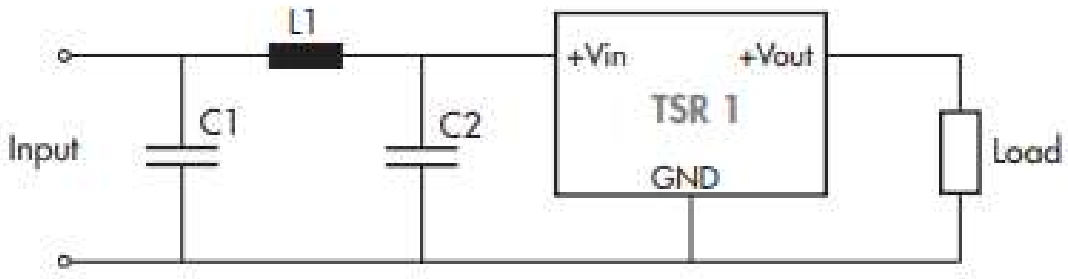

Рисунок 4.1 – Схема подключения TCR 1-2450 [14]

После получения 5В на выходе данного каскада, необходимо было также получить фиксированное питание с выходным напряжение в 3.3В для того чтобы можно было использовать его для питания микроконтроллера и датчиков, которые рассчитаны на напряжение менее чем 5В. Для этого используется линейный регулятор напряжения LM1117, который позволит получить стабильное напряжение. Типичная схема подключения данного линейного регулятора напряжения представлена на рисунке 4.2.

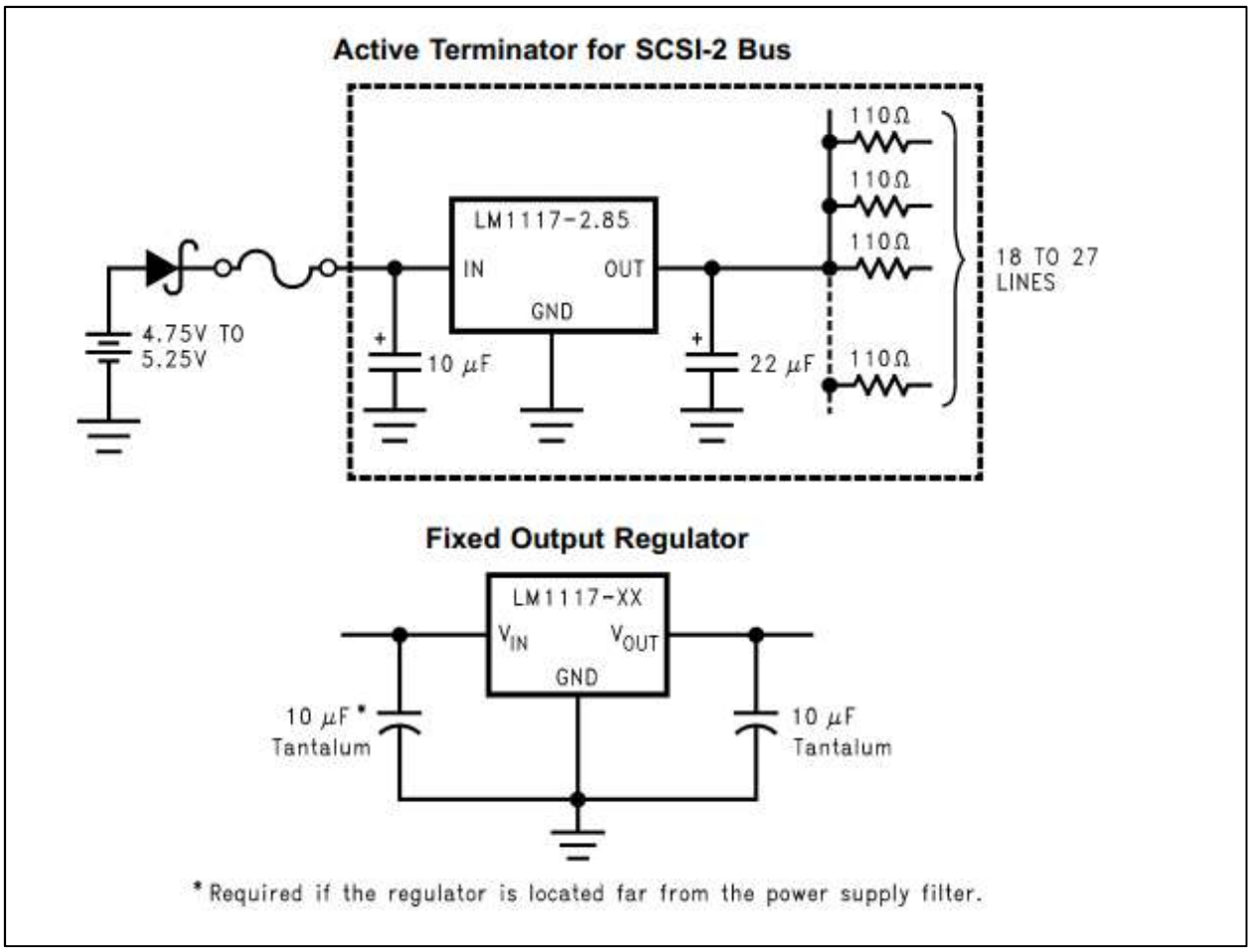

Рисунок 4.2 – Схема подключения LM1117 [16]

4.1.2 Интерфейс RS-485

Для коммутации сервера и основного микроконтроллера был выбран интерфейс RS-485. Данное решение обусловлено надежностью передачи данных на большие расстояния. Для реализации данного интерфейса была выбрана микросхема Texas Instruments ISO35DW. Данная микросхема является полнодуплексным драйвером, как было рассмотрено в подразделе 1.13.

Однако при реализации данного интерфейса необходимо было позаботиться о различных проблемах:

– защита от внешних помех источника питания;

– защита от повышенного напряжения;

– защита микросхемы от повышенного тока.

Для обеспечения защиты от внешних помех источника питания предусмотрено разделение питания логической и шинной частей. Для питания шинной части использованы микросхемы TI SN6501DBVR и Analog Devices ADP151AUJZ-3.3. Микросхема SN6501DBVR является генератором импульсов и предназначена для использования в схемах изолированных источников питания. В свою очередь, микросхема ADP151AUJZ-3.3 является малошумящим линейным регулятором напряжения.

Для того чтобы обеспечить защиту микросхемы ISO35DW от повышенного напряжения со стороны дифференциальных линий используется сборка полупроводниковых ограничителей напряжения SM712TC. При возникновении на линиях связи повышенного напряжения, вследствие воздействия внешних факторов, линии будут замкнуты на землю.

Для защиты микросхемы приемопередатчика от помех со стороны дифференциальных линий используются подтягивающие резисторы и со стороны приемника используется терминатор R13, который позволяет оборвать линию связи, если поступит высокое напряжение на плату.

Для гальванической развязки RS-485 интерфейса используется рекомендуемая микросхема ISO35DW. Основные компоненты, которые были необходимы для включения платы были рассмотрены на рисунках 4.3-4.6. На рисунке 4.3 показана схема включения микросхемы ISO35DW. Данная микросхема с обвязкой должна стоять с обеих сторон связи, поэтому на схеме электрической принципиальной показаны данные гальванические развязки в элементах DA8 и DA10.

Также, для защиты схемы используются резисторы R8-R11 и R12, R14-R16 соответственно, в качестве делителей напряжения. Это сделано для того, чтобы при поступлении большого напряжения на линии связи и, если другие методы защиты не успели сработать, обеспечить меньшее напряжение на входе микросхемы.

4.1.3 Выбор датчиков и их схема подключения

Исходя из функциональных особенностей, которые были установлены в ходе функционального проектирования системы, необходимо было определиться с датчиками, которые будут использоваться. Далее рассмотрим датчики и их способ подключения.

4.1.3.1 Датчик освещенности

Исходя из требований, которые были обозначены в подразделе 3.3.1, было принято решение использовать датчик TCRT5000, который

соответствует всем изложенным требованиям. Типичная схема подключения датчика представлена на рисунке 4.4.

Данный датчик требует напряжение питания +5В. А также данный датчик передает аналоговый сигнал по выходу, который можно преобразовать посредством АЦП в микроконтроллере. Данный датчик освещенности потребляет в районе 5 мкА, что также способствовало выбору данного датчика.

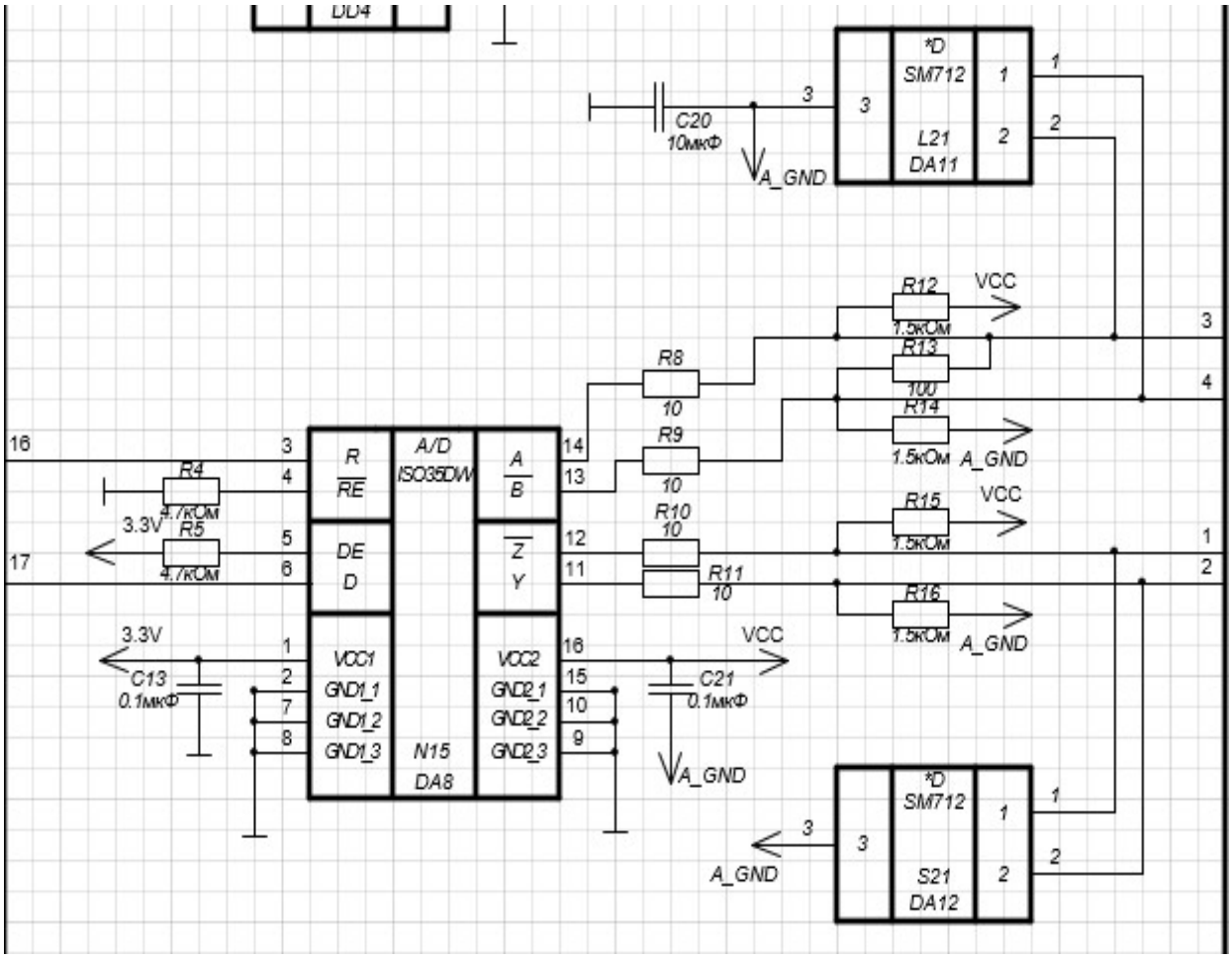

Рисунок 4.3 – Гальваническая развязка

4.1.3.2 Датчик присутствия

Исходя из требований, которые были обозначены в подразделе 3.3.2, было принято решение использовать датчик HC-SR501, который соответствует всем изложенным требованиям. Датчик имеет низкое потребление тока при выполнении расчетов и равен 65 мА. Датчик требует входное напряжение от 5 до 20 вольт. Данный датчик имеет цифровой выход, который сигнализирует о передвижениях в радиусе действия.

4.1.3.3 Датчик температуры и влажности

Исходя из требований, которые были обозначены в подразделе 3.3.3, было принято решение использовать датчик DHT22, описание которого представлено в подразделе 1.5. Питание данного датчика будет браться от вторичного источника питание с 3.3В на выходе. Типичная схема подключения данного датчика представлена на рисунке 4.5.

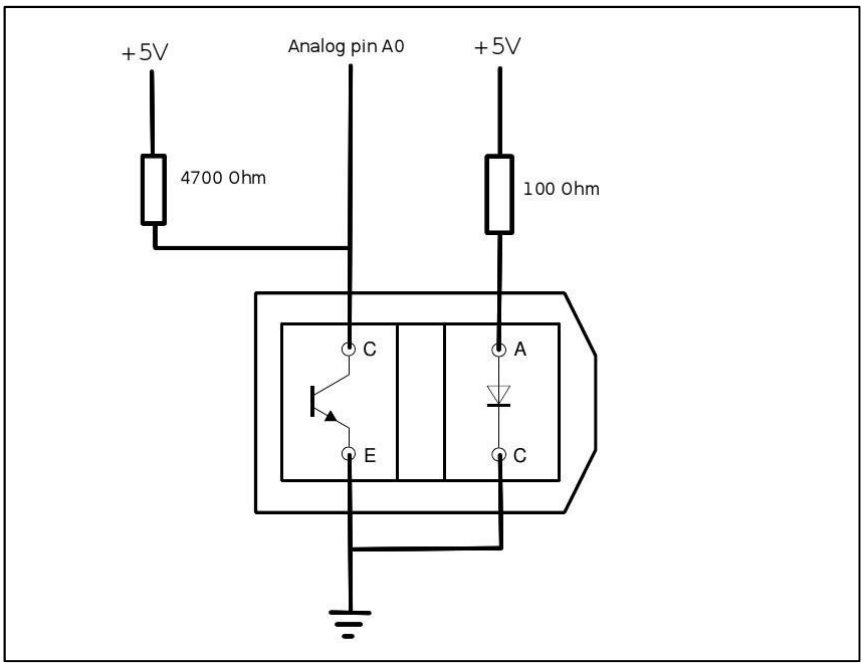

Рисунок 4.4 – Схема подключения датчика TCRT5000

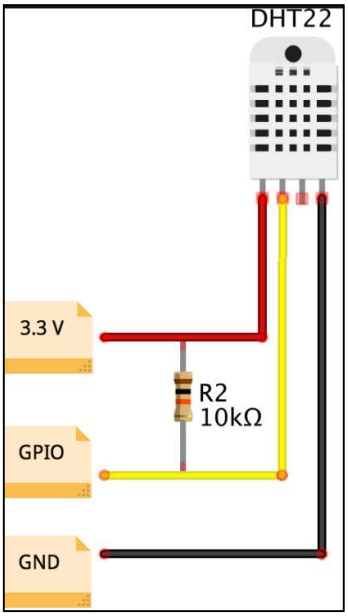

Рисунок 4.5 – Схема подключения датчика DHT22

4.1.3.4 Датчик CO2

Исходя из требований, которые были обозначены в подразделе 3.3.4, было принято решение использовать датчик MH-Z19, описание которого представлено в подразделе 1.9. Питание данного датчика будет браться от вторичного источника питание с 5В на выходе.

4.1.4 Беспроводной интерфейс и его подключение

Для подключения датчиков и исполнительных устройств по беспроводному интерфейсу необходимо было выбрать легкое в подключении устройство. Для этих целей был выбран nRF24L01, описанный в подразделе 1.4, данный интерфейс имеет SPI интерфейс для взаимодействия с модулем, к которому он подключен, а также беспроводной интерфейс использует частоту 2.4 ГГц для обмена данными. Питание данного датчика также соответствует требованиям, которые были указаны ранее.

Данный интерфейс использует питание от 3.3 В и не требует дополнительных элементов для подключения.

4.1.5 Подключение беспроводных датчиков и исполнительных устройств

Датчики, которые будут находиться удаленно от основного микроконтроллера, будут иметь питание от батарейки, чтобы была возможность расположить их в любой точке помещения, которая находится в радиусе действия беспроводного интерфейса, который будет подключен к ним.

Исполнительные устройства в свою очередь будут получать питание от сети помещения. Одним из исполнительных устройств, реализуемых в данной системе является шаговый двигатель 17HS4402. Для управления им требуется использование драйвера шагового двигателя A4988, который требует входное питание от 8 до 35 вольт. Данный драйвер может преобразовывать питание для управляющего микроконтроллера, который подключен к нему, тем самым давая возможность не использовать дополнительное питание для беспроводного интерфейса и микроконтроллера. Для включения данного исполнительного устройства необходимо было использовать устройство с беспроводным интерфейсом, микроконтроллер, который будет получать информацию с данного беспроводного интерфейса. Также необходимо было использовать драйвер для устройства шагового двигателя. Схема включения шагового двигателя представлена на рисунке 4.6.

Исходя из того, что в данной схеме подключения требуется использование микроконтроллера, который может использовать питание 3.3 вольт, было принято решение использовать для всех беспроводных устройств также микроконтроллеры STM32.

Питание для исполнительного устройства планируется подводить из общей сети 24 В. Так как устройство в активном режиме использует большое количество энергии и обычные батарейки будут приходить в негодность после нескольких активаций шагового двигателя.

Беспроводной датчик влажности и присутствия подключаются по схеме, представленной на рисунках 4.7-4.8 соответственно.

Обе схемы используют питание от литий-ионной батарейки, так как обе схемы имеют низкое энергопотребление.

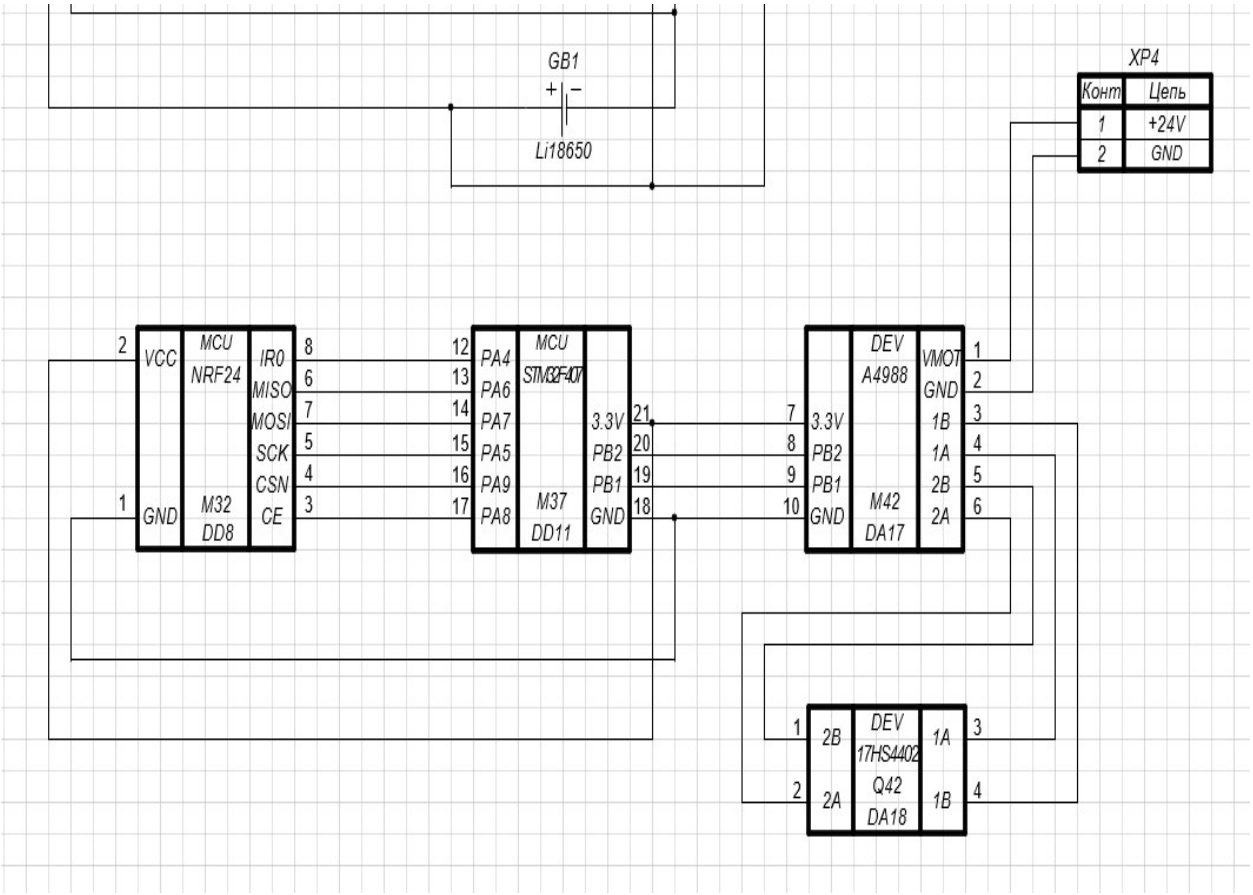

Рисунок 4.6 – Типичная схема подключения драйвера

4.2 Расчет потребления схемы и источников вторичного питания

Основными потребителями тока в схеме являются микроконтроллеры и их внутренняя периферия. Первостепенно необходимо рассмотреть потребление схемы основного устройства, которое включает в себя микроконтроллер STM32, датчики, подключенные к нему, и устройство беспроводной связи.

Микроконтроллер STM32F407 потребляет в активном режиме работы в районе 200мА, данное значение сильно зависит от того, сколько и какую периферию использует микроконтроллер. Данная микросхема будет использовать источник питания с напряжением в 3.3 В.

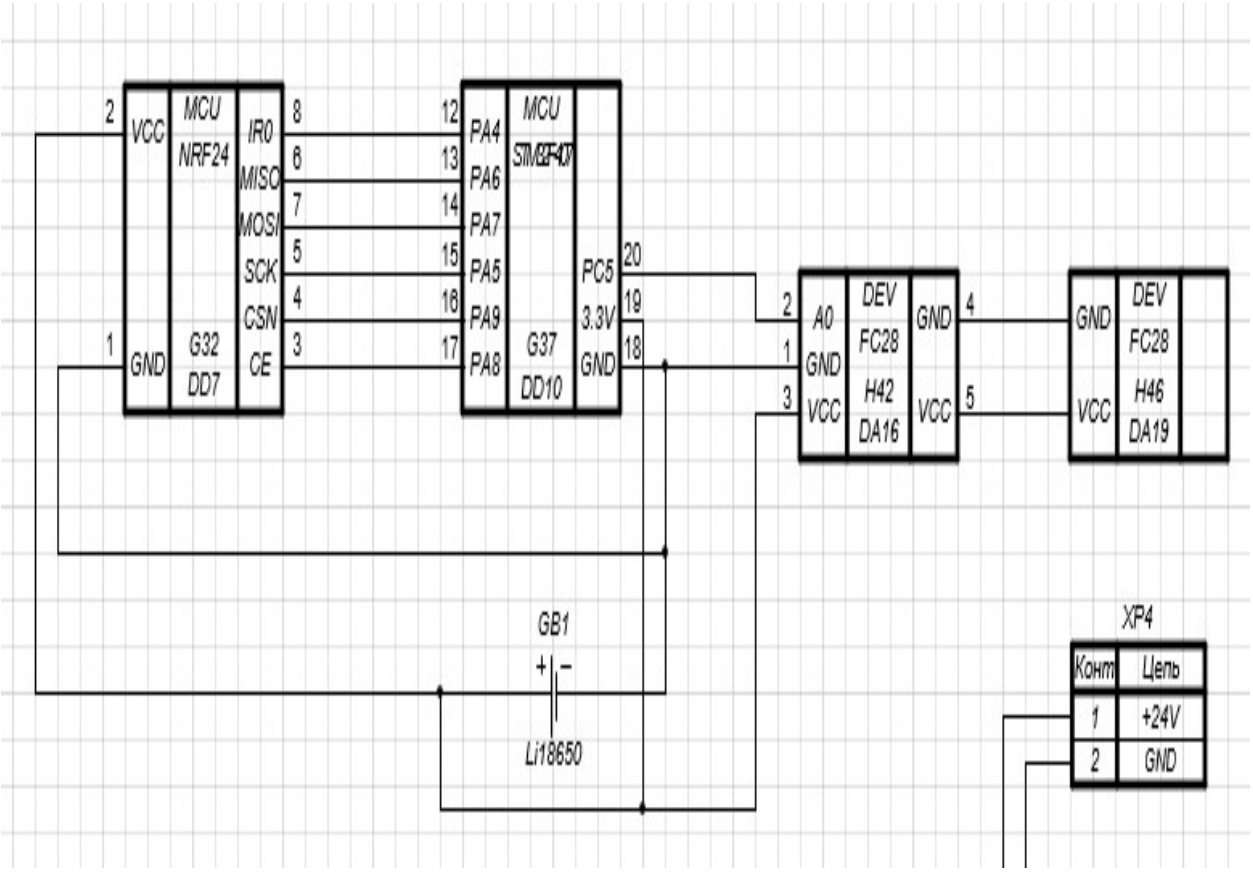

Рисунок 4.7 – Схема включения датчика влажности

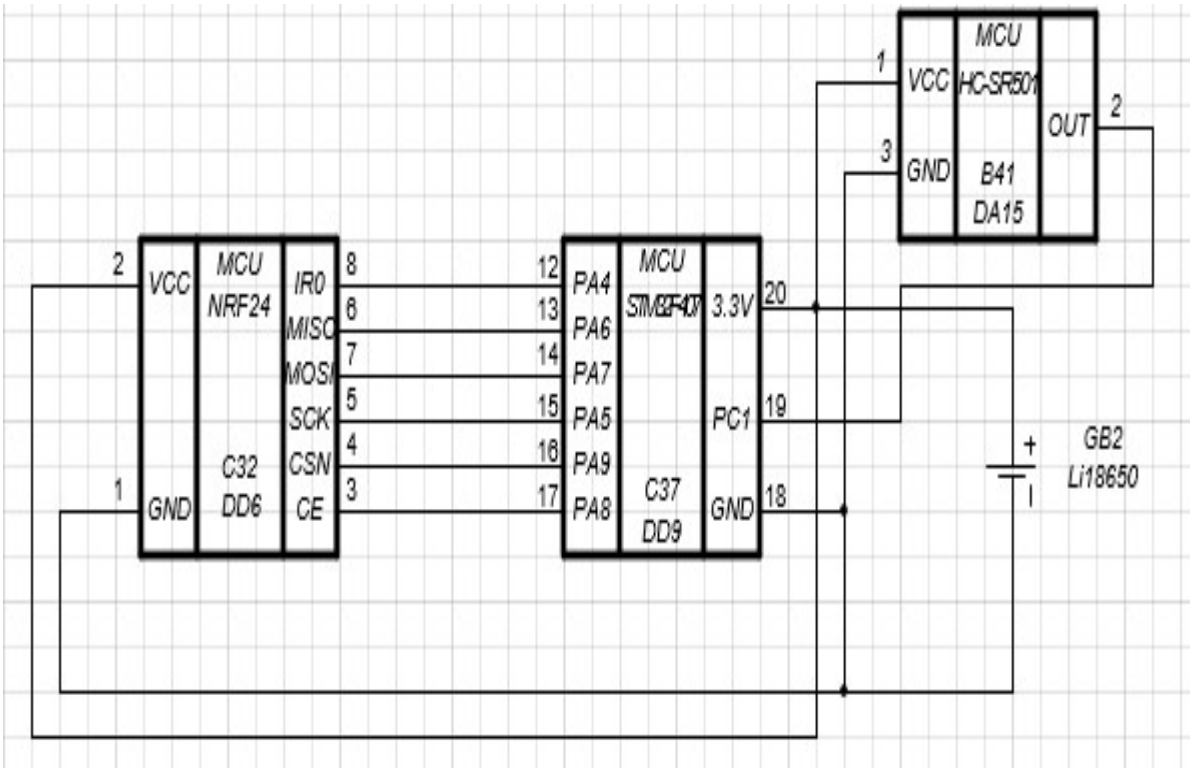

Рисунок 4.8 – Схема включения датчика присутствия

Микросхема датчика TCRT5000 требует напряжение в 3.3 В и потребляет в обычном режиме 60 мА.

Микросхема датчика DHT22 требует напряжение в 5 В и потребляет 1.5 мА.

Микросхема датчика HC-SR501 требует напряжение 5 В и потребляет в режиме измерения до 65 мА.

Микросхема датчика MH-Z19 требует напряжение 3.3 В и в режиме измерения потребляет до 18мА.

Беспроводной интерфейс nRF24L01 в режиме приема/передачи данных требует напряжение 3.3 В и потребляет 25 мА.

Микросхема RS-485 требует напряжение 3.3 В. Ток потребления составляет до 39мА.

Учитывая все необходимые условия по питанию микросхем, можно использовать для данной части схемы первичное питание 5 В и 3.3 В для микросхем датчиков и микроконтроллера в зависимости от минимального допустимого напряжения питания.

Рассчитаем потребляемую мощность для микросхем, которые требуют напряжение 3.3 В:

$$P1 = 3.3 * (0.2 + 0.018 + 0.025 + 0.039 + 0.06) = 1.129 \text{ Вт}$$

Рассчитаем потребляемую мощность для микросхем, которые требуют напряжение 5 В:

$$P2 = 5 * (0.0015 + 0.065) = 0.3325 \text{ Вт}$$

Суммарная потребляемая мощность модуля составляет:

$$P = P1 + P2$$

где P_1 – потребляемая мощность от 3.3 В,

P_2 – потребляемая мощность от 5 В.

$$P = 1.129 + 0.3325 = 1.4615 \text{ Вт}$$

Учитывая все дополнительные потери, можно округлить потребление до 2 Вт. Из этого следует, что модуль потребляет до 400 мА от первичного напряжения в 5 В.

Микросхема датчика влажности требует напряжение питания 3.3 В и ток потребления составляет до 35 мА.

Исходя из этих данных и данных по потреблению датчика присутствия, описанные выше рассчитаем потребляемую мощность для каждого из устройств.

Потребляемая мощность микроконтроллера с беспроводным интерфейсом и датчиком влажности:

$$P3 = 3.3 * (0.2 + 0.025 + 0.035) = 0.858 \text{ Вт}$$

Потребляемая мощность микроконтроллера с беспроводным интерфейсом и датчиком HC-SR501:

$$P4 = 3.3 * (0.2 + 0.025 + 0.065) = 0.958 \text{ Вт}$$

Исходя из этих данных необходимо было подобрать аккумулятор, который будет питать данные датчики. Произведем теоретический расчет емкости аккумулятора формуле 4.1:

$$C_{ак} = I_н \cdot K \cdot t, \quad (4.1)$$

где t – продолжительность работы аккумуляторной батареи в часах,

$C_{ак}$ – емкость аккумулятора в А·ч,

$I_н$ – ток нагрузки в А,

K – поправочный коэффициент.

Потребляемая мощность в активном режиме беспроводных датчиков приблизительно равна 1 Вт, следовательно, потребляемый ток будет в диапазоне 0.3 А. Поправочный коэффициент возьмем усредненный для АКБ аккумуляторов и равный 1.428. Также, возьмем за основу, что микроконтроллер должен будет отработать минимум 10 часов в активном режиме работы. Значение емкости аккумулятора:

$$C_{ак} = 0.3 \cdot 1.428 \cdot 10 = 4.28 \text{ А} \cdot \text{ч}$$

Исходя из теоретического расчета и вариантов, которые предложены на рынке, были выбран литий-ионный аккумулятор Li18650, данный аккумулятор имеет емкость аккумулятора 3.4 А · ч. Данной емкости не хватит на полноценную работу в течении 10 часов, но этого вполне достаточно на работу в активном режиме на 80% от расчетного времени работы.

При использовании литий-ионных аккумуляторов необходимо использовать модуль контроля заряда-разряда. Типичным решением является использование контроллера TP4056 LBC. Данный контроллер имеет максимальный ток заряда до 1 А, а напряжение заряда до 4.2 В. Имеет вход для подключения датчика перегрева АКБ. Также контроллер имеет разъем питания Micro USB.

5 МОДЕЛИРОВАНИЕ

После того как была разработана принципиальная схема устройства, необходимо написать исполняемый код для моделирования работы системы.

Так как проектирование плат и введение их в эксплуатацию занимает большое количество времени и ресурсов, было принято решение протестировать систему на макетной плате STM32F407GV.

5.1 Настройка окружения

Для написания исполняемого кода была выбрана среда разработки под микроконтроллеры STM32 MDK Keil uVision.

Первым шагом в написании кода является создание проекта для выбранного микроконтроллера, в данном проекте – это STM32F407GV. Так как настройки для микроконтроллеров разной серии и имеющих на борту разные чипы существенно отличаются. Создание проекта представлено на рисунках 5.1-3.

Рисунок 5.1 – Создание проекта - 1

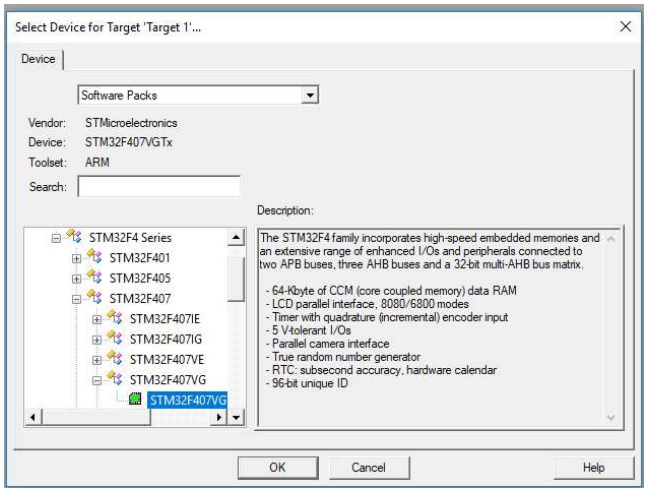

Рисунок 5.2 – Создание проекта - 2

В окне «Select Device for Target ‘Target 1’…», представленном на рисунке 5.2, необходимо выбрать чип, для которого создается проект - STM32F407VG.

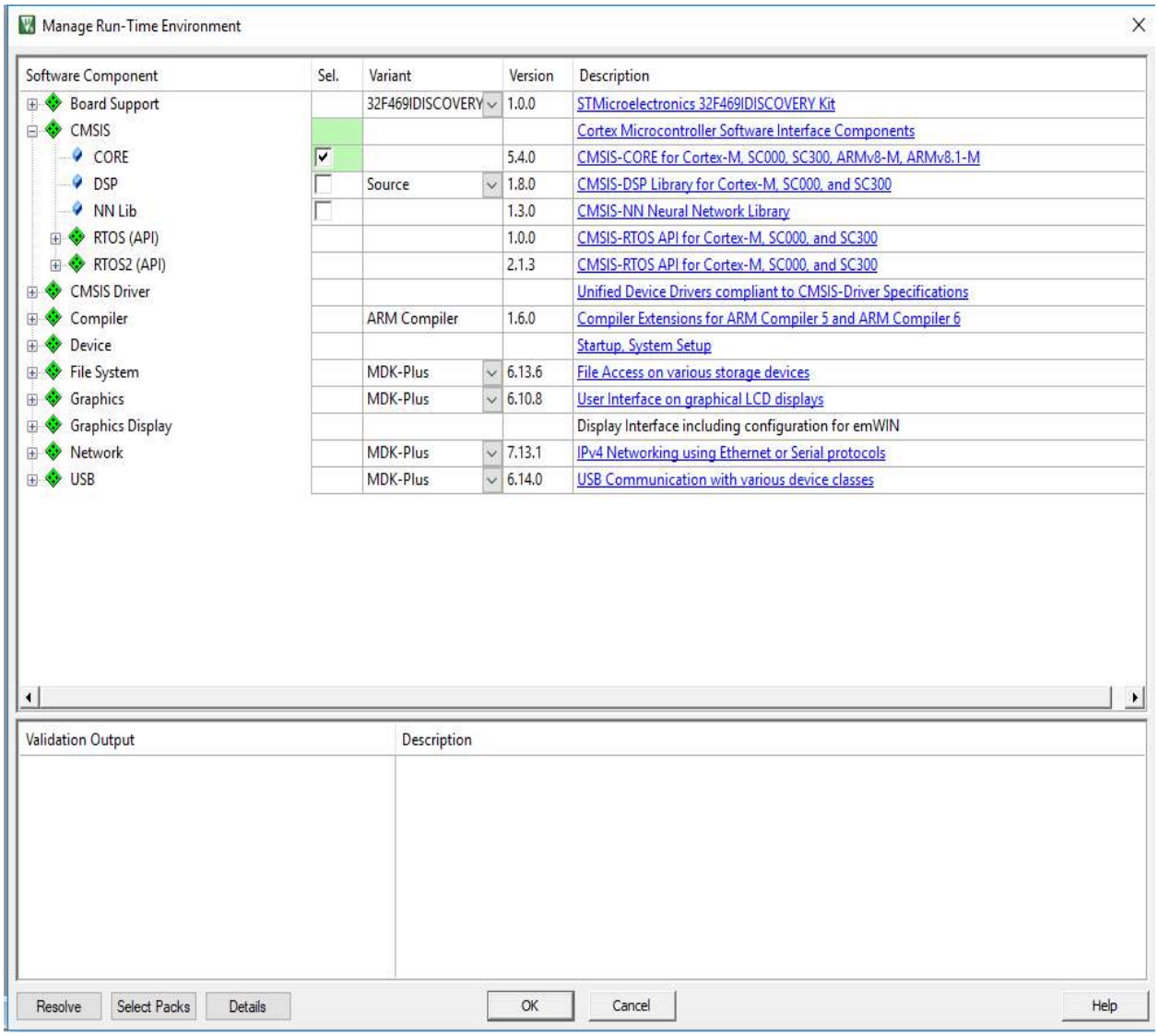

Рисунок 5.3 – Создание проекта - 3

В следующем диалоговом окне, представленном на рисунке 5.3, необходимо выбрать настройки окружения и библиотеки, которые будут подключены по умолчанию в проект. В данном проекте необходимо выбрать настройки для CMSIS.

Следующим шагом настройки является подключение директорий дополнительных библиотек, написанных пользователем, а также стандартных библиотек CMSIS от разработчиков компании ST Electronics. Данный шаг представлен на рисунке 5.4.

После подключения директориев необходимо добавить в проект файлы, которые будут использоваться из подключенных директориев для библиотек кода. Данный шаг представлен на рисунке 5.5.

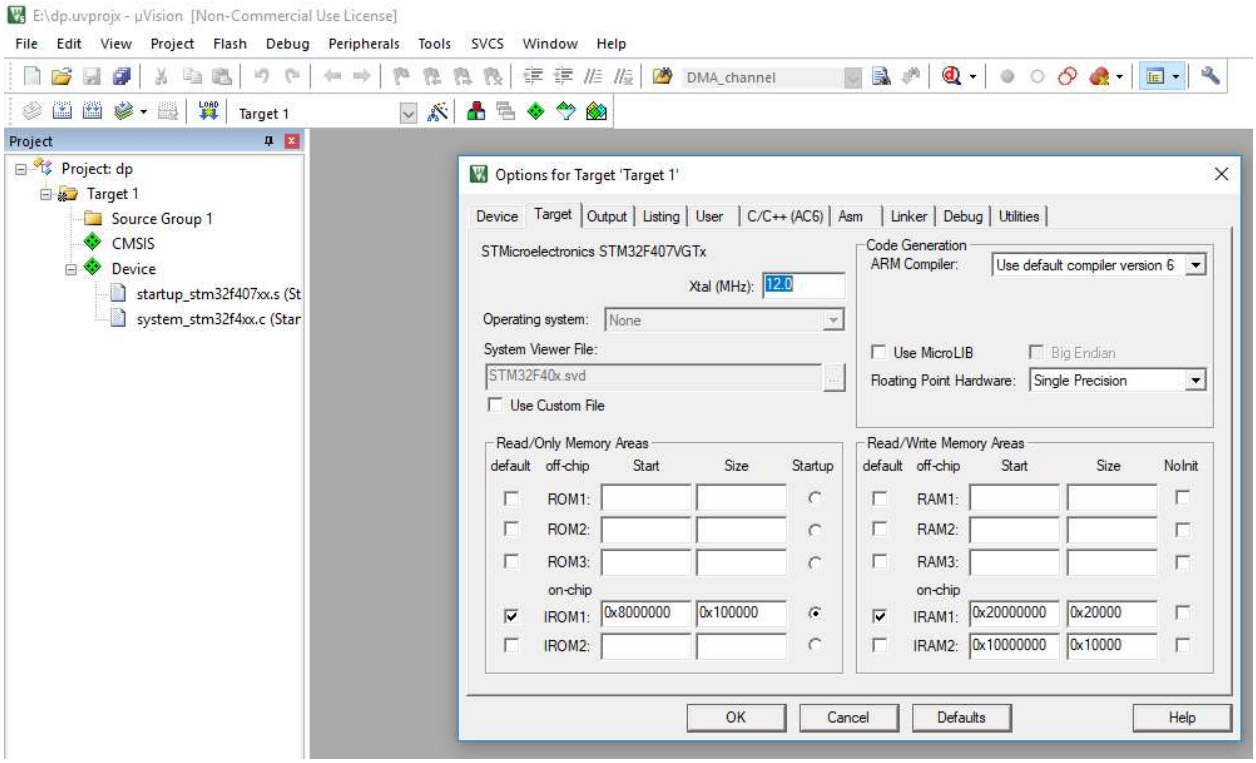

Рисунок 5.4 – Подключение директориев библиотек

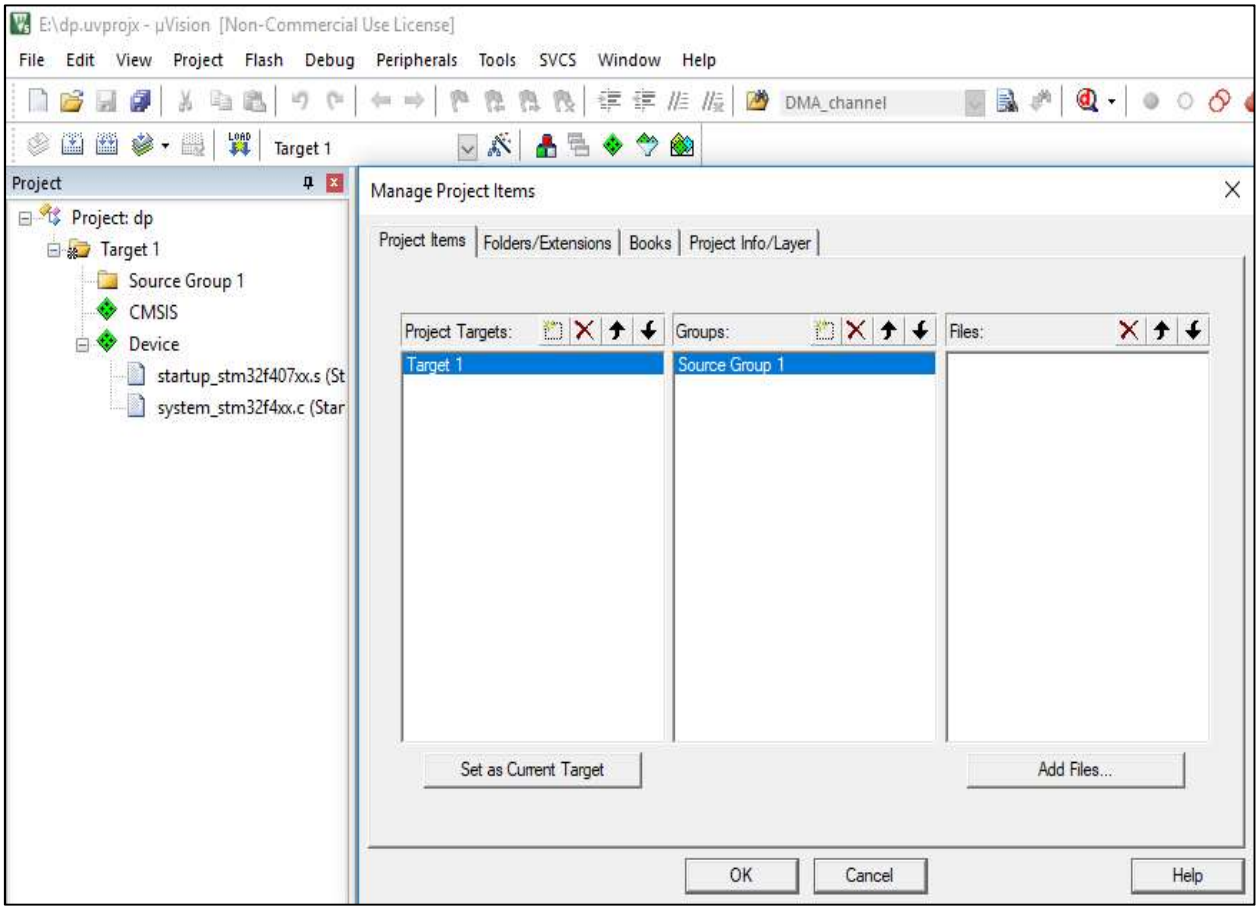

Рисунок 5.5 – Подключение файлов в проект

Для работы с микроконтроллером в режиме отладки, а также подключения к нему правильного программатора, необходимо выбрать во вкладке Debug, представленной на рисунке 5.6, отладчик, который подключен в данный момент к устройству. В данном случае это отладчик ST-Link Debugger.

Во вкладке настройки языка C/C++, представленной на рисунке 5.7 необходимо указать путь к папкам, в которых будут находиться файлы, которые будут непосредственно подключены к проекту. А также константные значения для проекта.

5.2 Моделирование работы RS-485 интерфейса

Для проверки работоспособности устройства и датчиков, подключенных к нему, была написана тестовая программа на языке С. Первым делом необходимо было настроить связь по UART интерфейсу, который послужит в дальнейшем интерфейсом для взаимодействия по RS-485 интерфейсу. Так как сервера, который будет посылать команды по данному интерфейсу нет, то была протестирована связь с обычным персональным компьютером без надстроек сетевых интерфейсов.

На персональном компьютере используется терминальная программа YAT, представленная на рисунке 5.8.

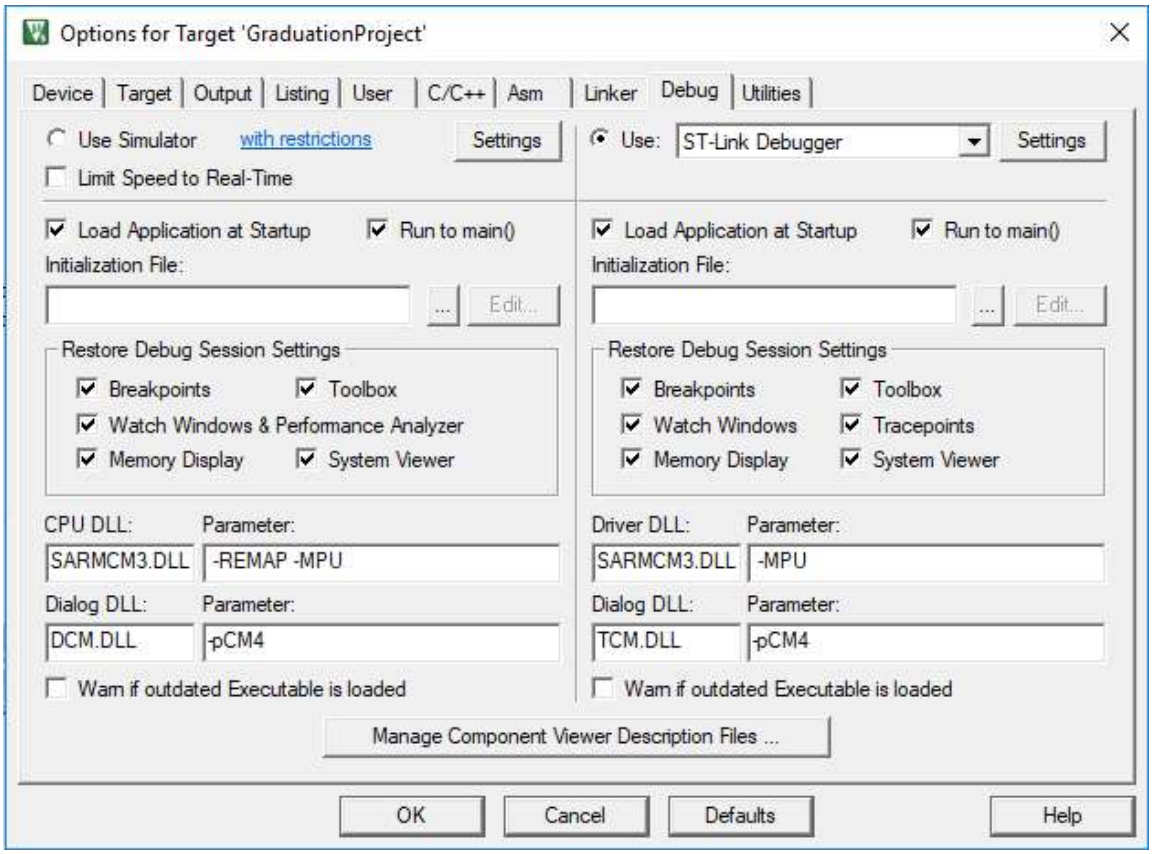

Рисунок 5.6 – Вкладка настройки отладчика

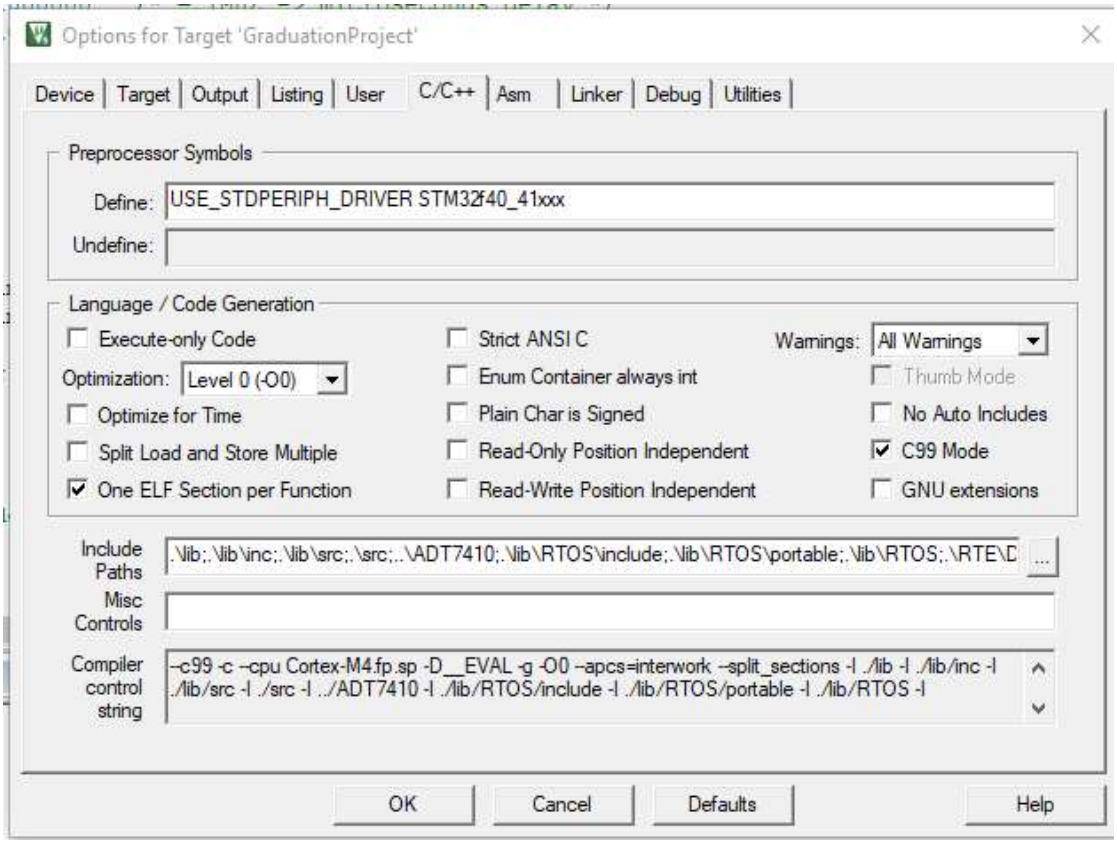

Рисунок 5.7 – Вкладка настройки языка C/C++

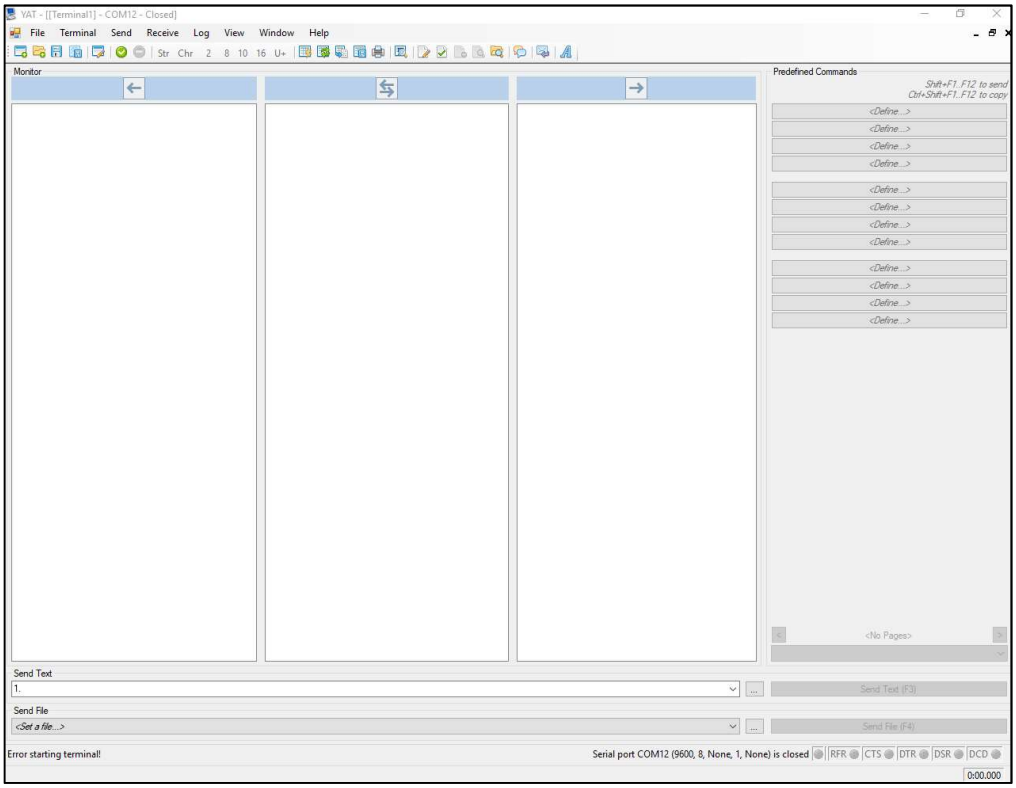

Рисунок 5.8 – Терминальная программа YAT

Данная программа удобна тем, что с помощью нее можно получать и отправлять запросы на любое устройство посредством последовательного порта используя настройки для приема и передачи данных, а также видеть данные сообщения в удобном для тестирования виде.

Первым делом проведем проверку подключения и передачи сообщения на разрабатываемое устройство.

На рисунке 5.9 видно, что после включения устройства, оно отсылает сообщение о том, какой чип используется в данный момент. Это сделано с целью отладки и проверки USART интерфейса, который в дальнейшем будет использоваться в качестве основы для ModBus протокола.

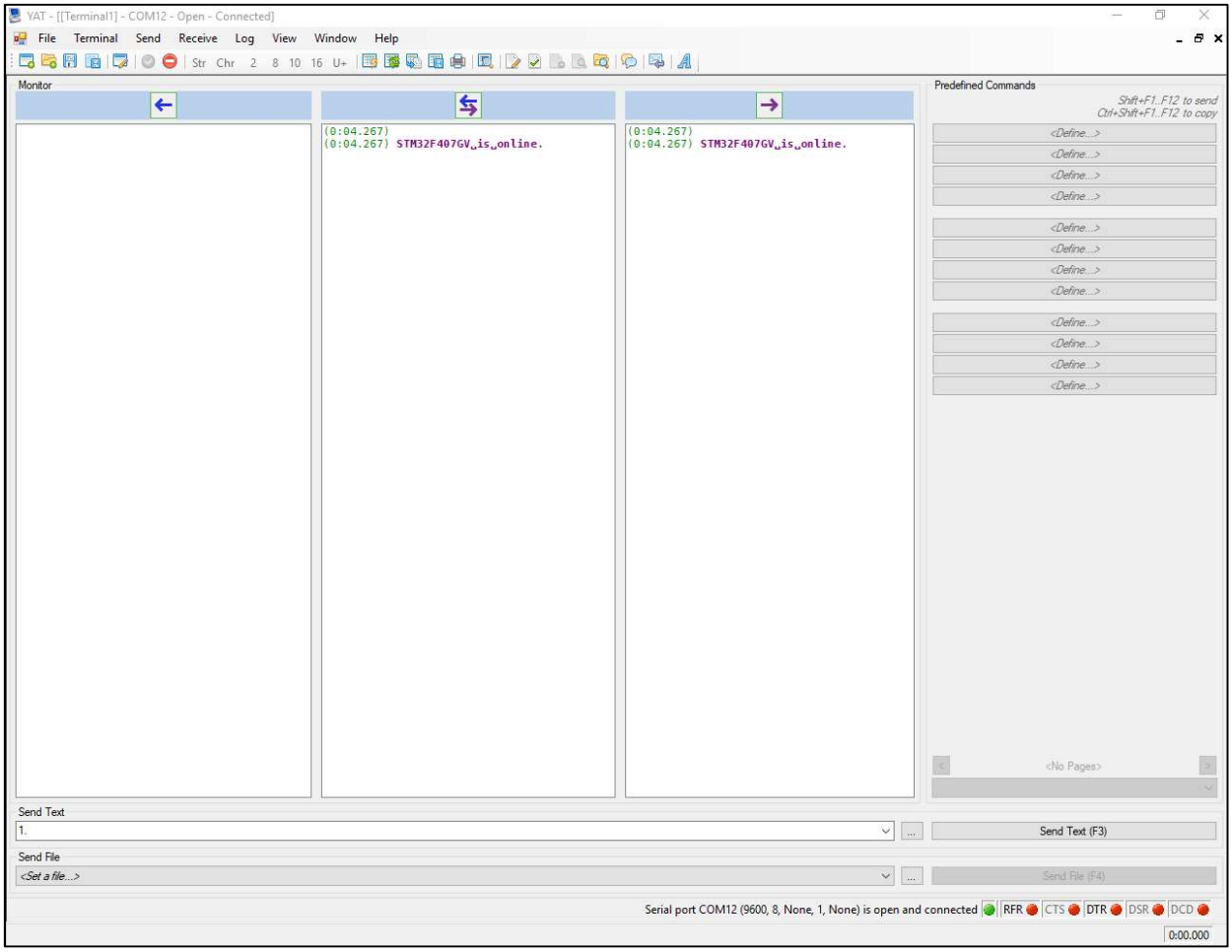

Рисунок 5.9 – Подключение к устройству

5.3 Моделирование работы датчиков

Для получения информации с датчика освещенности необходимо отправить с персонального компьютера команду «1.». На рисунке 5.10 показано получение данных с датчика при обычном освещении в комнате.

На рисунке 5.11 показаны значения датчика при разной освещенности в комнате, данного эффекта добивались закрытием фоторезистора от источника света.

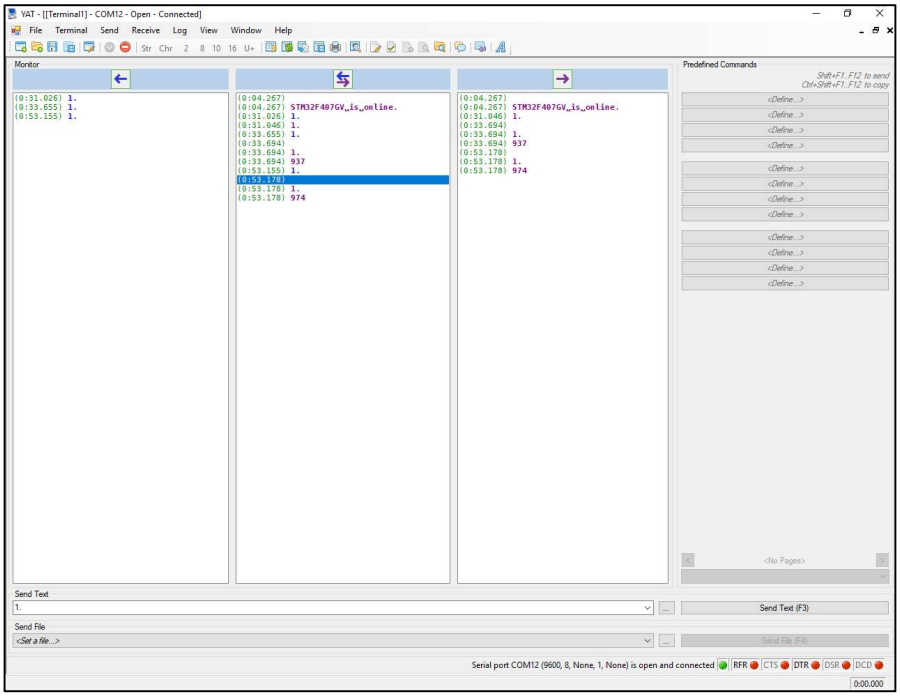

Рисунок 5.10 – Получение информации об освещенности

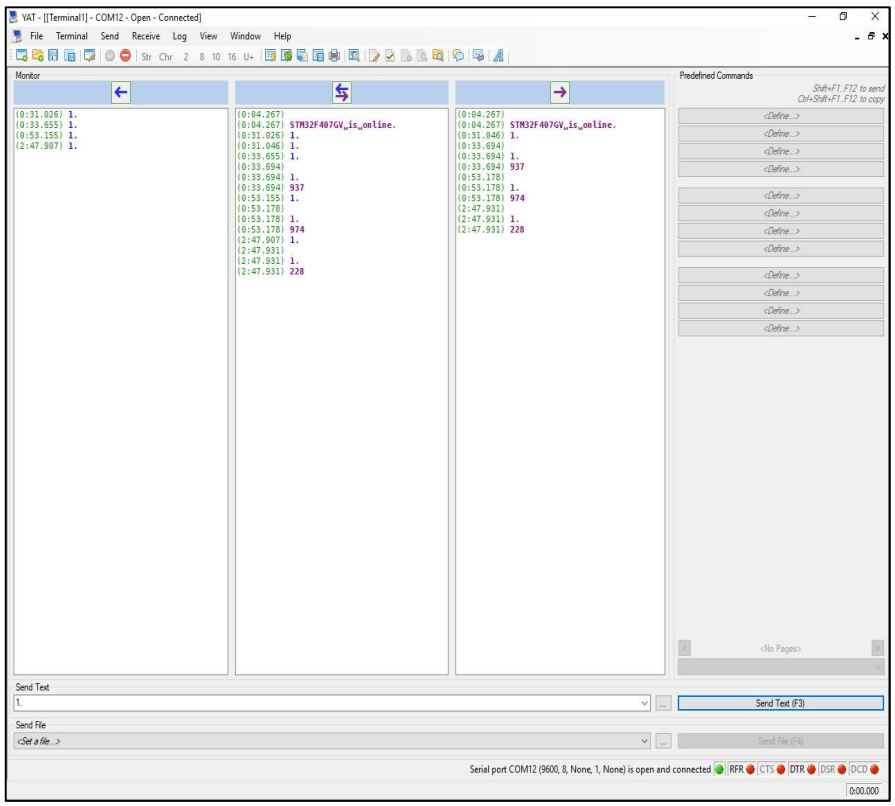

Рисунок 5.11 – Получение информации об освещенности – 2

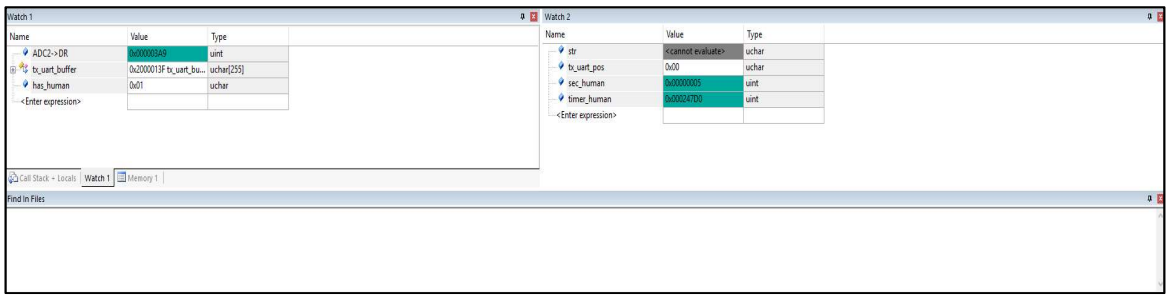

Рисунок 5.12 – Получение информации об присутствии

На рисунке 5.12 показаны отладочные данные, в которых видно, что значение переменной `has_human` приняло значение, которое соответствует тому, что в помещении было перемещение. Для тестирования каждые 10 секунд приходят новые данные о том, было ли движение в помещении или нет. Если было перемещение, то отправляется сообщение «has human». Если перемещения в помещении не было, то отправляется сообщение «no has human».

На рисунке 5.13 показана работа датчиков и получении данных с датчиков в тоже самое время.

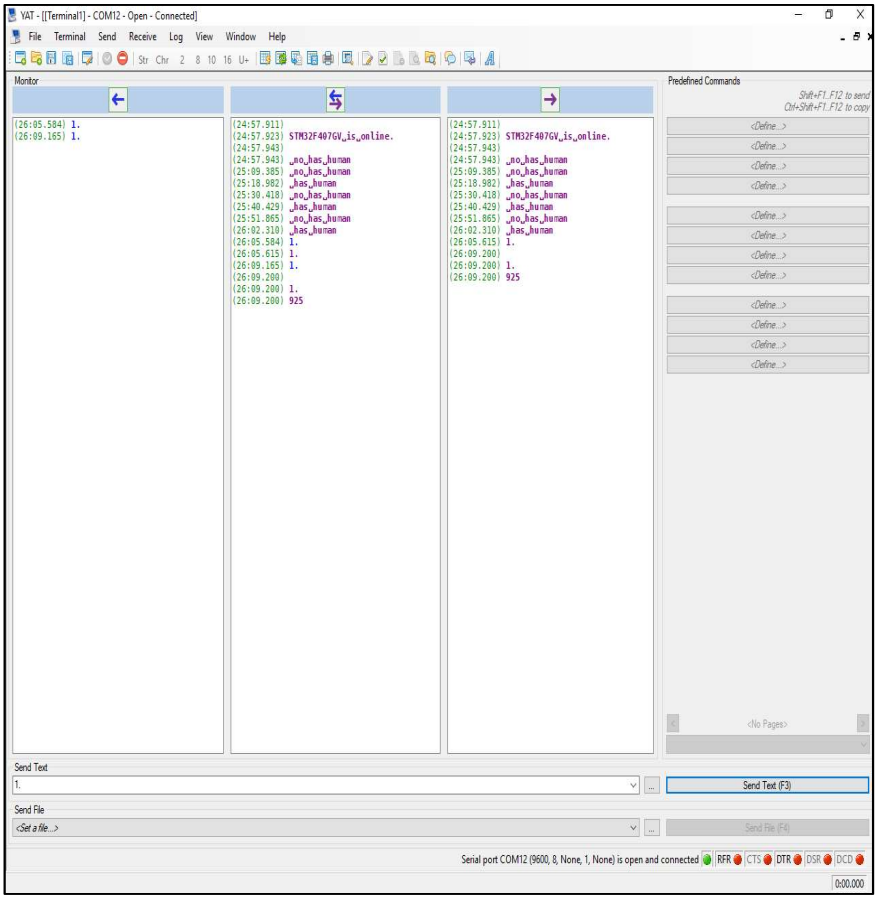

Рисунок 5.13 – Получение информации об присутствии - 2

На рисунке 5.14 также была произведена попытка подключения датчика влажности и температуры. Данные были успешно получены и отправлены на персональный компьютер с которого была отправлена команда.

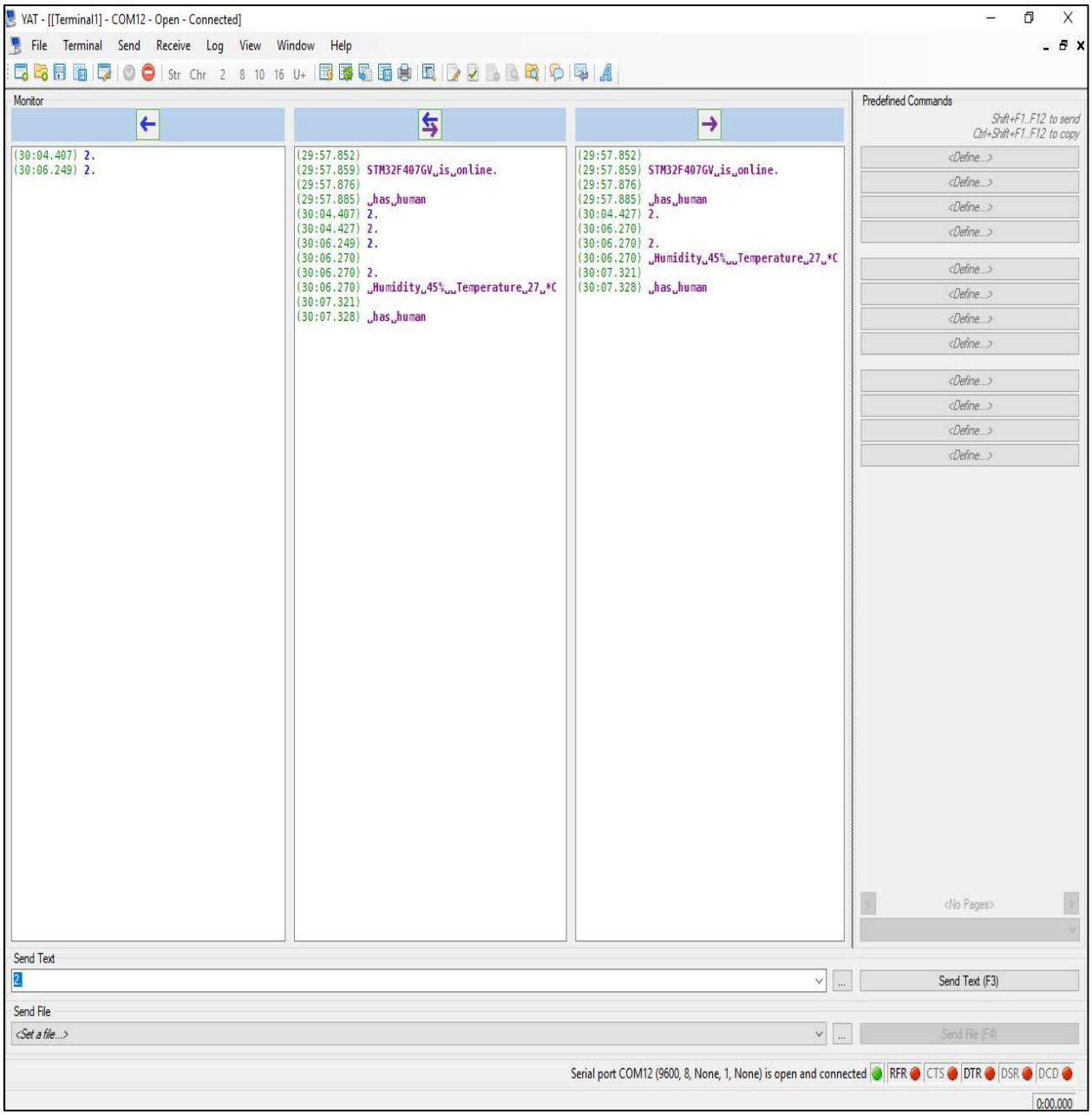

Рисунок 5.14 – Получение информации с датчика влажности и температуры

5.4 Алгоритм работы устройства

В данном подразделе рассмотрим основной алгоритм работы устройства, которое будет взаимодействовать с другими датчиками, исполнительными устройствами посредством своих проводных и

беспроводных интерфейсов. Схема алгоритма работы программы представлена на чертеже ГУИР.400201.019 ПД.

Далее рассмотрим алгоритм по шагам:

Шаг 1. Подготовка промежуточных данных, которые будет использоваться для передачи в функции инициализации.

Шаг 2. Включение тактирования внутренней периферии для GPIOB.

Шаг 3. Инициализация структуры GPIO_InitTypeDef с направление работы на входной сигнал, без подтягивающего резистора. Выставляем максимальную частоту работы входа.

Шаг 4. Вызов функции для инициализации входа для датчика HC-SR501.

Шаг 5. Включаем тактирование внутренней периферии для GPIOA.

Шаг 6. Включаем тактирование аналого-цифрового преобразователя №2 для обработки аналогового сигнала с датчика TCRT5000.

Шаг 7. Инициализация структуры GPIO_InitTypeDef с выбором режима работы альтернативной функции и вход обозначаем как аналоговый. Выбираем максимальную частоту работы входа.

Шаг 8. Вызов функции для инициализации входа для датчика TCRT5000.

Шаг 9. Вызов функции для базовой настройки всех внутренних АЦП.

Шаг 10. Заполнение полей АЦП №2 для обработки сигнала аналогового с канала, на который мультиплексирован вход TCRT5000.

Шаг 11. Вызов функции инициализации АЦП.

Шаг 12. Вызов функции, разрешающей работу АЦП №2.

Шаг 13. Вызов функции, разрешающей конвертацию данных в АЦП №2.

Шаг 14. Включение тактирования внутренней периферии GPIOC и GPIOD.

Шаг 15. Включение тактирования внутреннего UART5.

Шаг 16. Инициализация структуры GPIO_InitTypeDef для входного порта RS-485.

Шаг 17. Вызов функции конфигурации порта TX как альтернативной функции.

Шаг 18. Вызов функции конфигурации порта RX как альтернативной функции.

Шаг 19. Вызов функции инициализации порта RX интерфейса RS-485.

Шаг 20. Инициализация структуры GPIO_InitTypeDef для выходного порта RS-485.

Шаг 21. Вызов функции инициализации порта TX интерфейса RS-485.

Шаг 22. Инициализация полей структуры UART5 для интерфейса RS-485.

Шаг 23. Вызов функции, инициализирующей интерфейс UART5.

Шаг 24. Инициализация структуры GPIO_InitTypeDef для интерфейса SPI к беспроводному модулю nRF24.

Шаг 25. Вызов функции, инициализирующей данные поля.

Шаг 26. Инициализация структуры SPI_InitTypeDef для взаимодействия с беспроводным модулем nRF24.

Шаг 27. Вызов функции инициализации SPI.

Шаг 28. Вызов функции разрешения работы интерфейса SPI для беспроводного модуля nRF24.

Шаг 29. Разрешение работы прерываний SPI для беспроводного модуля nRF24.

Шаг 30. Старт бесконечного цикла.

Шаг 31. Если прошла секунда с предыдущего момента, то шаг 32, иначе шаг 35.

Шаг 32. Если прошло 60 секунд с предыдущего срабатывания датчика HC-SR51, то шаг 33, иначе шаг 34.

Шаг 33. Обновление информации о том, что в помещении никого нет и переход к шагу 35.

Шаг 34. Увеличение счетчика для датчика HC-SR51 и переход к шагу 35.

Шаг 35. Вызов функции обновления значения освещенности помещения.

Шаг 36. Вызов функции обновления значения влажности и температуры помещения.

Шаг 37. Если было получено сообщение от мастер-устройства, то шаг 38, иначе шаг

Шаг 38. Разбор сообщения, полученного от мастер-устройства.

Шаг 39. Если была получена команда «1», то шаг 40, иначе шаг 41.

Шаг 40. Отправка данных, которые были собраны с датчиков и исполнительных устройств, переход к шагу 51.

Шаг 41. Если была получена команда «2», то шаг 42, иначе шаг 43.

Шаг 42. Отправка данных о подключенных датчиках и исполнительных устройствах, переход к шагу 51.

Шаг 43. Если была получена команда «3», то шаг 44, иначе шаг 45.

Шаг 44. Отправка информации о состоянии датчиков и исполнительных устройств.

Шаг 45. Если была получена команда «4», то шаг 46, иначе шаг 47.

Шаг 46. Отправка сообщения подключенным устройствам с ведущим-устройством, переход к шагу 51.

Шаг 47. Если была получена команда «5», то шаг 48, иначе шаг 49.

Шаг 48. Подключение нового беспроводного устройства и переход к шагу 51.

Шаг 49. Если была получена другая команда, то шаг 50, иначе шаг 51.

Шаг 50. Отправка сообщения об ошибке мастер-устройству, переход к шагу 51.

Шаг 51. Сбор данных от беспроводных датчиков.

Шаг 52. Сбор данных с беспроводных исполнительных устройств.

Шаг 53. Конец бесконечного цикла, переход к шагу 30.

Шаг 54. Конец исполнения программы.

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ПРОИЗВОДСТВА АППАРАТНОГО КОМПЛЕКСА СИСТЕМЫ АВТОМАТИЗАЦИИ УПРАВЛЕНИЯ ХОЗЯЙСТВОМ

6.1 Характеристика аппаратно-программного комплекса

Разрабатываемая в данном дипломном проекте «Аппаратная система автоматизации управления хозяйством» является функциональным прибором, который позволяет получать данные с датчиков, подключенных беспроводным соединением или же посредством проводного соединения, а также управление исполнительными устройствами, которые находятся в подчинении разрабатываемого устройства. Данный прибор может быть использован в любой сфере, как в частном секторе, так и в помещениях офисных или же любых других. Его пользователями могут быть как частные лица, так и владельцы предприятий.

Его преимущество — это возможность подключения большого количества датчиков и исполнительных устройств и их вариация в зависимости от потребностей пользователя. Данная аппаратная система позволяет получить систему автоматизации, которая может в последствии расширяться различными устройствами от разных производителей.

6.2 Расчет экономического эффекта от производства аппаратно-программного комплекса

Для производства данной системы были использованы следующие материалы (см. таблицу 6.1) и комплектующие (см. таблицу 6.2):

Таблица 6.1 – Расчет затрат на основные и вспомогательные материалы

Наименование материала	Ед. Изм.	Норма расхода материала	Цена, р.	Сумма, р.
1. Припой ПОС 61, 100г	г	0,28	21,00	5,88
2. Флюс ЛТИ-120, 20 мл	мл	0,4	4,40	1,76
3. МГТФ 0.12 кв.мм, Провод монтажный, 1м	м	2,8	0,76	2,13
3. Текстолит двухстороний 1,5мм, 100х100мм	мм	0,9	5,00	4,50
5. Кембрик, 1м	м	1,2	0,59	0,71
6. Стойка дистанценирующая	шт	4	0,34	1,36
Итого				16,34
Всего с учетом транспортных расходов, $Р_м$				17,97

Все материалы и комплектующие взяты по ценам, сложившимся на рынке на текущую дату. Коэффициент транспортных расходов взят 1,1.

Таблица 6.2 – Расчет затрат на комплектующие изделия

Наименование материала	Количество на изд, шт.	Норма расхода материала	Цена, р.	Сумма, р.
1. Микроконтролер STM32F407VET6	1	1	21,00	21,00
2. NRF24L01 модуль беспроводной связи	1	1	7,80	7,80
3. DHT22 цифровой датчик температуры	1	1	18,30	18,30
4. KY-038 датчик звука	1	1	6,25	6,25
5. TCRT5000 датчик освещенности	1	1	6,30	6,30
6. MH-Z19 датчик CO_2	1	1	113,77	113,77
7. HC-SR501 датчик присутствия	1	1	7,80	7,80
8. Посадочное место для МК	1	1	1,00	1,00
9. Корпус пластиковый	1	1	9,96	9,96
Итого:				192,18
Всего с учетом транспортных расходов, Рк				211,40

Расчет общей суммы прямых затрат на производство аппаратной части изделия предоставлен в таблице 6.3.

Таблица 6.3 – Расчет общих прямых затрат

Показатель	Сумма, р.
1. Сырье и материалы	17,97
2. Покупные комплектующие изделия	192,18
Всего прямые затраты на производство аппаратной части ($З_р^{ач}$)	210,15

Расчет затрат на основную заработную плату разработчиков программной части комплекса представлена в таблице 6.4.

При Расчете зарплаты использовались данные среднемесячной зарплаты в Республике Беларусь для сотрудников IT отрасли. Премия не начисляется.

Основная зарплата определяется по формуле (6.1):

$$З_о = К_{пр} \sum_{i=1}^{n} З_{чi} \cdot t_i, \qquad (6.1)$$

где $К_{пр}$ – коэффициент премий (в нашем Расчете)

n – категории исполнителей, занятых разработкой;

$З_{чi}$ – часовая заработная плата исполнителя i-й категории, р.;

t_i – трудоемкость работ, выполняемых исполнителем i-й категории, определяется исходя из сложности разработки программного

обеспечения и объема выполняемых им функций, ч.

Часовая заработная плата каждого исполнителя определяется путем деления его месячной заработной платы (оклад плюс надбавки) на количество рабочих часов в месяце (Расчетная норма рабочего времени на 2020г. для 5-дневной недели составляет 168 ч по данным Министерства труда и социальной защиты населения на момент проведения Расчетов).

Таблица 6.4. – Расчет основной зарплаты разработчиков программной части

Категория разработчика	Месячная заработная плата, р.	Часовая заработная плата, р.	Трудоемкость работ, ч	Итого, р.
Инженер - программист	2150,00	12,80	168	2150,00
Техник - программист	1500,00	8,93	168	1500,00
Итого				3650,00
Премия (50–100 %)				1,00

Дополнительная зарплата разработчиков определяется по формуле (6.2)

$$З_д = \frac{З_о \cdot Н_д}{100}, \quad (6.2)$$

где $Н_д$ – норматив дополнительной зарплаты, 15%

Отчисления в фонд социальной защиты населения и обязательное страхование БелГосстрах (Зсз) определяется в соответствии с действующим законодательством по формуле (6.3)

$$Р_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100} \quad (6.3)$$

где $Н_{соц}$ – ставка отчислений в фонд социальный защиты населения (ФСЗН) и БелГосстрах (в соответствии с действующим законодательством на 01.01.2020 г. составляет 34,6%)

Расчет общей суммы затрат на разработку программной части программно-управляемого комплекса в таблице 6.5.

Таблица 6.5 – Расчет затрат на разработку программного средства

Наименование статьи затрат	Расчет по формуле	Сумма, р.
1. Основная заработная плата разработчиков	Табл.6.4	3650,00
2. Дополнительная заработная плата разработчиков	Формула (6.1)	547,50
3. Отчисления на социальные нужды	Формула (6.1)	1452,33
Итого		5649,84

Формирование отпускной цены программно-аппаратного комплекса представлена в таблице 6.6.

Таблица 6.6 – Расчет отпускной цены

Показатель	Расчет по формуле (в таблице)	Сумма, р.
1. Затраты на производство аппаратной части ($З_р^{ач}$)	Табл.6.3	210,15
2. Затраты на разработку программной части ($З_р^{пч}$)	Табл.6.5	5649,84
3. Сумма затрат на производство программно-аппаратного комплекса	Зпр =210,15+5649,84	6159,99
4. Накладные расходы	$Р_{накл}$ = 6159,99·0,56	3449,59
5. Расходы на реализацию	Ррел= 6159,99·0,02	123,18
6. Полная себестоимость	Сп=6159,99+3449,59+123,18	9732,76
7. Плановая прибыль, включаемая в цену	Пед=9732,76·25/100	2433,19
8. Отпускная цена	Цотп=9732,76+2433,19	12165,95

Результатом производства аппаратно-программного комплекса является прирост чистой прибыли, полученный от их реализации.

$$\Delta П_ч = П_{ед} \cdot N_п \left(1 - \frac{Н_п}{100}\right), \quad (6.4)$$

где $N_п$ – прогнозируемый годовой объем производства и реализации, шт.;

$П_{ед}$ – прибыль, включаемая в цену, р.;

$Н_п$ – ставка налога на прибыль согласно действующему законодательству, % (по состоянию на 01.01.2020 г. – 18 %).

$$\Delta П_ч = 2433,19 \cdot 50 \ (1-0,18) = 99760,79 \text{ руб}$$

6.3 Расчет инвестиций в проектирование и производство аппаратно-программного комплекса

Инвестиции в производство программно-аппаратного комплекса включают в общем случае:

– инвестиции на его разработку;

– инвестиции в прирост основного капитала (затраты на приобретение необходимого для производства нового изделия оборудования, станков и т.п.);

– инвестиции в прирост собственного оборотного капитала (затраты на приобретение необходимых для производства нового изделия материалов, комплектующих, начатой, но незавершенной продукции и т.п.).

6.3.1 Расчет инвестиций на разработку аппаратно-программного комплекса

Инвестиции рассчитываем ($И_р$) по затратам на разработку нового изделия. Для начала необходимо рассчитать заработную плату технологам и инженерам предприятия-производителя, ввиду того что именно они являются первой статьёй первичных затрат.

Расчет заработной платы разработчиков нового изделия в таблице 6.7.

Таблица 6.7 – Расчет заработной платы разработчиков нового изделия

Категория исполнителя	Количество, чел.	Месячная зарплата, р.	Дневная зарплата, р.	Время, д.	Заработная плата, р.
1.Руководитель проекта	1	2350	111,90	84	9400,00
2. Инженер по электронной технике	1	2200	104,76	84	8800,00
3. Техник - технолог	2	1580	75,24	22	3310,48
4. Инженер - системотехник	1	2300	109,52	80	8761,90
Итого					30272,38
Премия, 30%					1,3
Всего основная заработная плата (Зо)					39354,10

Расчета затрат на разработку нового изделия в таблице 6.8.

Таблица 6.8 – Расчет инвестиций на разработку нового изделия

Наименование статьи затрат	Расчет по формуле (в таблице)	Сумма, р.
1. Основная заработная плата разработчиков $З_о$	табл. 6.7	39354,10
2. Дополнительная заработная плата разработчиков	Зд= Зо · 0,15	5903,11
3. Отчисления на социальные нужды	Рсоц = (Зо + Зд) · 0,346	15658,99
4. Инвестиции на разработку нового изделия	Ир=Зо + Зд + Рсоц	60916,20

Инвестиции в прирост основного капитала не требуются, т. к. производство нового изделия планируется осуществлять на действующем оборудовании в связи с наличием на предприятии-производителе свободных производственных мощностей.

6.3.2 Расчет инвестиций в прирост оборотного капитала

1) Определяется годовая потребность в материалах по формуле:

$$П_м = Р_м \cdot N_п, \quad (6.5)$$

где $Р_м$ – затраты на материалы на единицу изделия, р. (см. таблицу 6.1),
Nп – прогнозируемый годовой объем.

$$П_м = 17,97 \cdot 50 = 898,5 \text{ р.}$$

2) Определяется годовая потребность в комплектующих изделиях по формуле:

$$П_к = Р_к \cdot N_п, \quad (6.6)$$

где $Р_к$ – затраты на комплектующие изделия на единицу продукции, р.

$$Пк = 192,18 \cdot 50 = 9609 \text{ руб}$$

3) Определяются инвестиции в прирост собственного оборотного капитала в процентах от годовой потребности в материалах и комплектующих изделиях (исходя из среднего уровня по экономике: 20–30 %) по формуле:

$$\Delta И_{сок} = 0,20(П_м + П_к). \quad (6.7)$$

$$\Delta И_{сок} = 0,20(898,5 + 9609) = 2101,50 \text{ руб}$$

Оценка экономической эффективности разработки изделия зависит от результата сравнения инвестиции в разработку и прирост собственных оборотных средств и полученного годового прироста чистой прибыли.

Рассчитаем рентабельность инвестиций по формуле

$$Р_и = \frac{\Delta П_ч}{И_{пр}} \cdot 100 \%, \quad (6.8)$$

где $\Delta П_ч$ – прирост чистой прибыли, руб.;
$И_{пр}$ – инвестиции в производство ($И_{сок}$ + Ир), руб.

$$Р_и = (99760,79 /63017,70) \cdot 100\% = 158,30\%$$

Сравнивая инвестиции в разработку изделия и прирост собственных оборотных средств, с приростом годовой чистой прибыли можно сделать вывод, что инвестиции окупаются в течении года.

Рентабельность инвестиций в производство превысила 100%, следовательно, разработка данного продукта является целесообразным.

ЗАКЛЮЧЕНИЕ

В ходе работы над дипломным проектом был разработан прототип системы автоматизации хозяйства на основе микроконтроллера STM32F4XX. Прототип устройства представляет из себя упрощенную версию разработанной системы, так как полная реализация проекта является более дорогим и время затратным производством.

Данный прототип позволил проверить работу всех необходимых интерфейсов и датчиков, которые будут подключаться к нему. Для реализации необходимых функций были использована среда разработки Keil uVision5, предназначенный для разработки устройств, использующих ARM процессоры, а также библиотеки CMSIS, предоставленные разработчиками программного обеспечения для микроконтроллеров STM32F4XX.

Основная проблема работы с микроконтроллерами и эффективностью их использования заключается в том, что микроконтроллер проводит большую часть времени простаивая. Для решения данной проблемы было принято решение использования таймеры устройства.

Разработанная система позволяет оценить состояние окружающей среды в помещении или нескольких помещениях, которые будут оборудованы устройствами управления и датчиками, которые будут необходимы для измерения параметров среды. Датчики могут подключаться как напрямую к устройству управления, так и удаленно, посредством беспроводного интерфейса. Помимо снятия показаний состояния окружающей среды, существует возможность управлять объектами в помещении посредством исполнительных устройств.

Проведенный экономический расчет позволил определить цену разработки системы в соответствии с ценами на май 2020 года.

Основные достоинства модуля:

– средняя стоимость чипа STM32F4XX;
– высокая частота работы чипов STM32F4XX;
– простая обработка данных с датчиков;
– работа беспроводного интерфейса на 2.4 Ггц;
– минимальное использование проводов для связи с сервером.

Возможные пути улучшения модуля:

– добавление новых датчиков;
– добавление новых исполнительных устройств;
– увеличение производительности за счет внедрения системы RTOS.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] beltelecom.by [Электронный ресурс]. – Электронные данные. – Режим доступа: https://beltelecom.by/private/smart-home - Дата доступа: 09.04.2020.

[2] z-wave.by [Электронный ресурс]. – Электронные данные. – Режим доступа: https://z-wave.by/scenarii/ - Дата доступа – 09.04.2020.

[3] mall.industry.siemens.com [Электронный ресурс]. – Электронные данные. – Режим доступа: https://mall.industry.siemens.com/mall/en/WW/Catalog/Products/10165502?tree=CatalogTree#Overview – Дата доступа 09.04.2020.

[4] preliminary product specification[Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.sparkfun.com/datasheets/Components/nRF24L01_prelim_prod_spec_1_2.pdf - Дата доступа 13.04.2020 .

[5] RF24L01 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://images-na.ssl-images-amazon.com/images/I/616S6tRIJAL._AC_SX425_.jpg - Дата доступа 13.04.2020.

[6] DHT22 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf – Дата доступа 13.04.2020.

[7] KY-038 Microphone sound sensor module [Электронный ресурс]. – Электронные данные. – Режим доступа: https://datasheetspdf.com/pdf-file/1402048/Joy-IT/KY-038/1 - Дата доступа 13.04.2020.

[8] HC-SR501 PIR MOTION DETECTOR [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.mpja.com/download/31227sc.pdf - Дата доступа: 13.04.2020.

[9] HC-SR501 PIR MOTION DETECTOR [Электронный ресурс]. – Электронные данные. – Режим доступа: https://components101.com/hc-sr501-pir-sensor Дата доступа: 13.04.2020.

[10] TCRT5000 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://pdf1.alldatasheet.com/datasheet-pdf/view/252411/VISHAY/TCRT5000.heml – Дата доступа: 13.04.2020.

[11] MH-Z19 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.winsen-sensor.com/d/files/PDF/Infrared%20Gas%20Sensor/NDIR%20CO2%20SENSOR/MH-Z19%20CO2%20Ver1.0.pdf – Дата доступа 13.04.2020.

[12] 17HS4401S Datasheet [Электронный ресурс]. – Электронные данные. – Режим доступа: https://datasheetspdf.com/mobile-datasheet/17HS4401S.html : Дата доступа: 13.04.2020.

[13] Reference manual [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf Дата доступа: 13.04.2020.

[14] TSR1-2450 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://pdf1.alldatasheet.com/datasheet-pdf/view/312813/TRACOPOWER/TSR1-

2450.html Дата доступа: 05.06.2020.

[15] TI training & videos [Электронный ресурс]. – Электронные данные. – Режим доступа: https://training.ti.com/ti-precision-labs-isolation-what-galvanic-isolation?utm_source=google&utm_medium=cpc&utm_campaign=asc-int-iso-galvanic_isolation-cpc-tr-google-wwe&utm_content=galvanic_isolation&ds_k=galvanic+isolation&DCM=yescontext=1139747-1135015-1139269-1135013&gclid=CjwKCAjwwYP2BRBGEiwAkoBpAszudcJwL9JCxgSxLeaK8vhJJpdExIciOFjPi8gZavHC8TQwg8hNxhoCG_gQAvD_BwE&gclsrc=aw.ds Дата доступа: 31.05.2020.

[16] LM1117 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.alldatasheet.com/view.jsp?Searchword=Lm1117&gclid=CjwKCAjw2uf2BRBpEiwA31VZj29xGUfpukAcI1Y-2UC83LfD6zoEcB9wbRmpK_lxQpwpDs8EH_zVFRoCklkQAvD_BwE Дата доступа: 31.05.2020.

[17] JNR13S5001 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.chipdip.by/product/jnr13s500l Дата доступа: 31.05.2020.

[18] Датчик CO2 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://7lestnic.com/bez-rubriki/vse-uzlyrezultat-klasterizaciidatcik-co2.html#gallery-1 Дата доступа: 31.05.2020.

ПРИЛ,ОЖЕНИЕ А

(обязательное)

Код программы

Файл main.c

```
#include "stm32f4xx.h"
#include "tim.h"

uint16_t count_TIM2 = 0;

#ifdef __RTOS__
          xSemaphoreHandle data_temp_semaphore = NULL;
#endif

#define  DELAY_TICK_FREQUENCY_US  1000000     /*  =  1MHZ  ->
microseconds delay */
#define  DELAY_TICK_FREQUENCY_MS  1000         /*  =  1kHZ  ->
milliseconds delay */


static __IO uint32_t TimingDelay; // __IO -- volatile


/*
 *   Declare Functions
 */
extern void Delay_ms(uint32_t nTime);
extern void Delay_us(uint32_t nTime);

/* Port and pin with DHT22 sensor*/
#define DHT22_GPIO_PORT            GPIOA
#define DHT22_GPIO_CLOCK           RCC_AHB1Periph_GPIOA
#define DHT22_GPIO_PIN             GPIO_Pin_10

/* DHT22_GetReadings response codes */
#define DHT22_RCV_OK               0 // Return with no error
#define  DHT22_RCV_NO_RESPONSE        1  //  No  response  from
sensor
#define DHT22_RCV_BAD_ACK1         2 // Bad first half length
of ACK impulse
#define DHT22_RCV_BAD_ACK2        3 // Bad second half length
of ACK impulse
#define DHT22_RCV_RCV_TIMEOUT      4 // It was timeout while
receiving bits


void DHT22_Init(void);
uint32_t DHT22_GetReadings(void);
uint16_t DHT22_DecodeReadings(void);
uint16_t DHT22_GetHumidity(void);
```

```
uint16_t DHT22_GetTemperature(void);


#define _UART_PORT 5


#if _UART_PORT == 5
     #define UART_PORT         UART5
     #define  UART_TX_PIN           GPIO_Pin_12      //  PC12
(UART5_TX)
     #define UART_RX_PIN     GPIO_Pin_2    // PD2  (UART5_RX)
     #define UART_GPIO_PORT_TX GPIOC
     #define UART_GPIO_PORT_RX GPIOD
#endif

#define HEX_CHARS      "0123456789ABCDEF"

void UART_Init(void);

void UART_SendChar(char ch);

void UART_SendInt(uint32_t num);
void UART_SendHex8(uint16_t num);
void UART_SendHex16(uint16_t num);
void UART_SendHex32(uint32_t num);

void UART_SendStr(char *str);

void UART_SendBuf(char *buf, uint16_t bufsize);
void UART_SendBufPrintable(char *buf, uint16_t bufsize, char
subst);
void UART_SendBufHex(char *buf, uint16_t bufsize);
void  UART_SendBufHexFancy(char  *buf,  uint16_t  bufsize,
uint8_t column_width, char subst);

uint8_t get_uart_rx_buf_cnt(void);
uint8_t has_str(void);
void clr_uart_rx_buf_cnt(void);
void send_req_uart_str(void);
uint8_t check_command(void);

uint32_t response;
uint16_t humidity,temperature;
uint8_t has_human = 0, has_human_buf = 0;
uint32_t  timer_human  =  0,  sec_human  =  0,  max_delay_sec  =
1700000;


GPIO_InitTypeDef gpio_struct;

int main ()
{
```

```
        UART_Init();
        UART_SendStr("\nSTM32F407GV is online.\n");

        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    //
        gpio_struct.GPIO_Mode = GPIO_Mode_OUT;
        gpio_struct.GPIO_OType = GPIO_OType_PP;
        gpio_struct.GPIO_Pin = GPIO_Pin_12| GPIO_Pin_13;
        gpio_struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
        gpio_struct.GPIO_Speed = GPIO_Speed_100MHz;

        GPIO_Init(GPIOD, &gpio_struct);
    //
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
        //HC-SR501
        gpio_struct.GPIO_Mode = GPIO_Mode_IN;
        gpio_struct.GPIO_OType = GPIO_OType_PP;
        gpio_struct.GPIO_Pin = GPIO_Pin_0;
        gpio_struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
        gpio_struct.GPIO_Speed = GPIO_Speed_100MHz;

        GPIO_Init(GPIOB, &gpio_struct);

        //DHT22_Init();


        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC2, ENABLE);

        gpio_struct.GPIO_Mode = GPIO_Mode_AN;
        gpio_struct.GPIO_OType = GPIO_OType_PP;
        gpio_struct.GPIO_Pin = GPIO_Pin_2;
        gpio_struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
        gpio_struct.GPIO_Speed = GPIO_Speed_100MHz;

        GPIO_Init(GPIOA, &gpio_struct);

        ADC_RegularChannelConfig(ADC2,     ADC_Channel_2,     1,
ADC_SampleTime_15Cycles);

        ADC_CommonInitTypeDef ADC_CommonInitStruct;

        ADC_CommonInitStruct.ADC_DMAAccessMode                 =
ADC_DMAAccessMode_Disabled;
        ADC_CommonInitStruct.ADC_Mode = ADC_Mode_Independent;
        ADC_CommonInitStruct.ADC_Prescaler = ADC_Prescaler_Div2;
        ADC_CommonInitStruct.ADC_TwoSamplingDelay              =
ADC_TwoSamplingDelay_10Cycles;

        ADC_CommonInit(&ADC_CommonInitStruct);

        ADC_InitTypeDef ADC_struct;
```

```
        ADC_struct.ADC_Resolution = ADC_Resolution_12b;
        ADC_struct.ADC_ScanConvMode = DISABLE;
        ADC_struct.ADC_ContinuousConvMode = ENABLE;
        ADC_struct.ADC_ExternalTrigConvEdge                    =
ADC_ExternalTrigConvEdge_None;
        ADC_struct.ADC_ExternalTrigConv                        =
ADC_ExternalTrigConv_T1_CC1;
        ADC_struct.ADC_DataAlign = ADC_DataAlign_Right;
        ADC_struct.ADC_NbrOfConversion = 1;
        ADC_Init(ADC2, &ADC_struct);

        ADC_Cmd(ADC2, ENABLE);

        ADC_SoftwareStartConv(ADC2);

        uint16_t adc_buffer = 0;
        uint8_t command = 0;

        while(1)
        {
            if (has_human == 0)
            {
                has_human     =     GPIO_ReadInputDataBit(GPIOB,
GPIO_Pin_0);
            }
            if (has_human_buf == 0)
            {
                has_human_buf = 1;
                if (has_human)
                {
                    UART_SendStr("\n has human");
                }
                else
                {
                    UART_SendStr("\n no has human");
                }
                has_human = 0;
            }
            if (timer_human < max_delay_sec)
            {
                timer_human++;
            }
            else
            {
                timer_human = 0;
                sec_human++;
                if (sec_human == 10)
                {
                    has_human_buf = 0;
                    sec_human = 0;
                }
```

```
            }
            uint8_t adc_to_uart = 0;
            if (ADC_GetFlagStatus(ADC2, ADC_FLAG_EOC))
            {
                adc_buffer = ADC2->DR;
                adc_to_uart = adc_buffer;
            }
            if (has_str())
            {
                command = check_command();
                send_req_uart_str();
                switch(command)
                {
                    case 1 :
                        {
                            UART_SendStr("\n");
                            UART_SendInt(adc_buffer);
                            break;
                        }
                    case 2 :
                        {
                            UART_SendStr("\n    Humidity    45%
Temperature 27 *C\n");
                            break;
                        }
                    case 3 :
                         {
                            UART_SendStr("\nSTM32F407GV  has  no
function 3.\n");
                            break;
                        }
                }

            }
        }
    }

    uint16_t bits[40];

    uint8_t  hMSB = 0;
    uint8_t  hLSB = 0;
    uint8_t  tMSB = 0;
    uint8_t  tLSB = 0;
    uint8_t  parity_rcv = 0;

    static GPIO_InitTypeDef PORT;


    void DHT22_Init(void) {
         RCC_APB2PeriphClockCmd(DHT22_GPIO_CLOCK,ENABLE);
         PORT.GPIO_Mode = GPIO_Mode_OUT;
         PORT.GPIO_Pin = DHT22_GPIO_PIN;
```

```
          PORT.GPIO_Speed = GPIO_Speed_50MHz;
          PORT.GPIO_PuPd = GPIO_PuPd_DOWN;
          GPIO_Init(DHT22_GPIO_PORT,&PORT);
     }

     uint32_t DHT22_GetReadings(void) {
          uint32_t wait;
          uint8_t i;

          // Generate start impulse for sensor

          Delay_ms(2); // Host start signal at least 800us
          // Switch pin to input with Pull-Up
          PORT.GPIO_Mode = GPIO_Mode_IN;
          PORT.GPIO_PuPd = GPIO_PuPd_UP;
          GPIO_Init(DHT22_GPIO_PORT,&PORT);

          // Wait for AM2302 to start communicate
          wait = 0;
          while  ((DHT22_GPIO_PORT->IDR  &  DHT22_GPIO_PIN)  &&
(wait++ < 200)) Delay_us(2);
          if (wait > 50) return DHT22_RCV_NO_RESPONSE;

          // Check ACK strobe from sensor
          wait = 0;
          while  (!(DHT22_GPIO_PORT->IDR  &  DHT22_GPIO_PIN)  &&
(wait++ < 100)) Delay_us(1);
          if ((wait < 8) || (wait > 15)) return DHT22_RCV_BAD_ACK1;

          wait = 0;
          while  ((DHT22_GPIO_PORT->IDR  &  DHT22_GPIO_PIN)  &&
(wait++ < 100)) Delay_us(1);
          if ((wait < 8) || (wait > 15)) return DHT22_RCV_BAD_ACK2;

          // ACK strobe received --> receive 40 bits
          i = 0;
          while (i < 40) {
               // Measure bit start impulse (T_low = 50us)
               wait = 0;
               while (!(DHT22_GPIO_PORT->IDR & DHT22_GPIO_PIN) &&
(wait++ < 20)) Delay_us(1);
               if (wait > 16) {
                    // invalid bit start impulse length
                    bits[i] = 0xffff;
                    while ((DHT22_GPIO_PORT->IDR & DHT22_GPIO_PIN)
&& (wait++ < 20)) Delay_us(1);
               } else {
                    // Measure bit impulse length (T_h0 = 25us,
T_h1 = 70us)
                    wait = 0;
                    while ((DHT22_GPIO_PORT->IDR & DHT22_GPIO_PIN)
&& (wait++ < 20)) Delay_us(1);
```

```
                    bits[i] = (wait < 16) ? wait : 0xffff;
               }

               i++;
          }

          for (i = 0; i < 40; i++) if (bits[i] == 0xffff) return
DHT22_RCV_RCV_TIMEOUT;

          return DHT22_RCV_OK;
     }

     uint16_t DHT22_DecodeReadings(void) {
          uint8_t parity;
          uint8_t  i = 0;

          hMSB = 0;
          for (; i < 8; i++) {
               hMSB <<= 1;
               if (bits[i] > 7) hMSB |= 1;
          }
          hLSB = 0;
          for (; i < 16; i++) {
               hLSB <<= 1;
               if (bits[i] > 7) hLSB |= 1;
          }
          tMSB = 0;
          for (; i < 24; i++) {
               tMSB <<= 1;
               if (bits[i] > 7) tMSB |= 1;
          }
          tLSB = 0;
          for (; i < 32; i++) {
               tLSB <<= 1;
               if (bits[i] > 7) tLSB |= 1;
          }
          for (; i < 40; i++) {
               parity_rcv <<= 1;
               if (bits[i] > 7) parity_rcv |= 1;
          }

          parity  = hMSB + hLSB + tMSB + tLSB;

          return (parity_rcv << 8) | parity;
     }

     uint16_t DHT22_GetHumidity(void) {
          return (hMSB << 8) + hLSB;
     }

     uint16_t DHT22_GetTemperature(void) {
          return (tMSB << 8) + tLSB;
```

```
}


#define UART_MAX_BUFFER 255

char rx_uart_buffer[UART_MAX_BUFFER];
uint8_t tx_uart_pos = 0;
uint8_t tx_req_sended = 0;
char tx_uart_buffer[UART_MAX_BUFFER];

void UART_Init(void)
{
     GPIO_InitTypeDef PORT;

#if _UART_PORT == 5
     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC           |
RCC_AHB1Periph_GPIOD,ENABLE);
     RCC_APB1PeriphClockCmd(RCC_APB1Periph_UART5,ENABLE);
#endif

     PORT.GPIO_Speed = GPIO_Speed_50MHz;
     PORT.GPIO_Mode  = GPIO_Mode_AF;
    PORT.GPIO_OType = GPIO_OType_PP;
    PORT.GPIO_PuPd  = GPIO_PuPd_UP;

   GPIO_PinAFConfig(UART_GPIO_PORT_TX,    GPIO_PinSource12,
GPIO_AF_UART5);
    GPIO_PinAFConfig(UART_GPIO_PORT_RX,     GPIO_PinSource2,
GPIO_AF_UART5);

     PORT.GPIO_Pin = UART_TX_PIN;
     GPIO_Init(UART_GPIO_PORT_TX,&PORT);

     PORT.GPIO_Pin = UART_RX_PIN;
     GPIO_Init(UART_GPIO_PORT_RX,&PORT);

     USART_InitTypeDef UART;

     UART.USART_BaudRate = 9600;
     UART.USART_HardwareFlowControl                          =
USART_HardwareFlowControl_None; // No flow control
     UART.USART_Mode  =  USART_Mode_Tx  |  USART_Mode_Rx;  //
RX+TX mode
     UART.USART_Parity = USART_Parity_No; // No parity check
     UART.USART_StopBits = USART_StopBits_1; // 1 stop bit
     UART.USART_WordLength  =  USART_WordLength_8b;  //  8-bit
frame
     USART_Init(UART_PORT,&UART);
     USART_Cmd(UART_PORT,ENABLE);

    USART_ITConfig(UART_PORT, USART_IT_RXNE, ENABLE);
```

```
        NVIC_EnableIRQ(UART5_IRQn);
    }

    void UART_SendChar(char ch)
    {
        while (!USART_GetFlagStatus(UART_PORT,USART_FLAG_TXE));
        USART_SendData(UART_PORT,ch);
    }

    void UART_SendInt(uint32_t num)
    {
        while (!USART_GetFlagStatus(UART_PORT,USART_FLAG_TXE));
        char str[10]; // 10 chars max for UINT32_MAX
        int i = 0;
        do str[i++] = num % 10 + '0'; while ((num /= 10) > 0);
        for (i--; i >= 0; i--) UART_SendChar(str[i]);
    }

    void UART_SendHex8(uint16_t num) {
        UART_SendChar(HEX_CHARS[(num >> 4)   % 0x10]);
        UART_SendChar(HEX_CHARS[(num & 0x0f) % 0x10]);
    }

    void UART_SendHex16(uint16_t num) {
        uint8_t i;
        for (i = 12; i > 0; i -= 4) UART_SendChar(HEX_CHARS[(num
>> i) % 0x10]);
        UART_SendChar(HEX_CHARS[(num & 0x0f) % 0x10]);
    }

    void UART_SendHex32(uint32_t num) {
        uint8_t i;
        for    (i    =    28;    i    >    0;    i    -=    4)
    UART_SendChar(HEX_CHARS[(num >> i) % 0x10]);
        UART_SendChar(HEX_CHARS[(num & 0x0f) % 0x10]);
    }

    void UART_SendStr(char *str)
    {
        while (*str) UART_SendChar(*str++);
    }

    void UART_SendBuf(char *buf, uint16_t bufsize) {
        uint16_t i;
        for (i = 0; i < bufsize; i++) UART_SendChar(*buf++);
    }

    void UART_SendBufPrintable(char *buf, uint16_t bufsize, char
subst) {
        uint16_t i;
        char ch;
        for (i = 0; i < bufsize; i++) {
```

```
            ch = *buf++;
            UART_SendChar(ch > 32 ? ch : subst);
        }
    }

    void UART_SendBufHex(char *buf, uint16_t bufsize) {
        uint16_t i;
        char ch;
        for (i = 0; i < bufsize; i++) {
            ch = *buf++;
            UART_SendChar(HEX_CHARS[(ch >> 4)   % 0x10]);
            UART_SendChar(HEX_CHARS[(ch & 0x0f) % 0x10]);
        }
    }

    void  UART_SendBufHexFancy(char  *buf,  uint16_t  bufsize,
uint8_t column_width, char subst) {
        uint16_t i = 0,len,pos;
        char buffer[column_width];

        while (i < bufsize) {
            // Line number
            UART_SendHex16(i);
            UART_SendChar(':'); UART_SendChar(' '); // Faster
and less code than USART_SendStr(": ");

            // Copy one line
            if (i+column_width >= bufsize) len = bufsize - i;
else len = column_width;
            memcpy(buffer,&buf[i],len);

            // Hex data
            pos = 0;
            while (pos < len) UART_SendHex8(buffer[pos++]);
            UART_SendChar(' ');

            // Raw data
            pos = 0;
            do UART_SendChar(buffer[pos] > 32 ? buffer[pos] :
subst); while (++pos < len);
            UART_SendChar('\n');

            i += len;
        }
    }

    void UART5_IRQHandler(void)
    {
        if (USART_GetFlagStatus(UART_PORT,USART_IT_RXNE))
        {
            tx_uart_buffer[tx_uart_pos]                      =
USART_ReceiveData(UART_PORT);
```

```
            tx_uart_pos++;
            tx_req_sended = 0;
        }
    }
    void send_req_uart_str(void)
    {
        uint8_t loc_tx_uart_pos = tx_uart_pos;
        tx_uart_pos = 0;
        UART_SendBuf(tx_uart_buffer, loc_tx_uart_pos);
    }

    uint8_t get_uart_rx_buf_cnt(void)
    {
        return tx_uart_pos;
    }

    uint8_t check_command(void)
    {
        switch(tx_uart_buffer[1])
        {
            case '1' : return 1;
            case '2' : return 2;
            case '3' : return 3;
        }
        return 4;
    }

    uint8_t has_str(void)
    {
        if   (tx_uart_buffer[tx_uart_pos   -   1]   ==   '.'   &&
tx_req_sended == 0)
        {
            tx_req_sended = 1;
            return 1;
        }
        return 0;
    }

    void clr_uart_rx_buf_cnt(void)
    {
        tx_uart_pos = 0;
    }

    // SysTick interrupt handler
    void SysTick_Handler() {
         if (TimingDelay != 0) { TimingDelay--; }
    }

    // Do delay for mSecs milliseconds
    void Delay_ms(uint32_t mSecs) {
         SysTick_Config(SystemCoreClock                          /
DELAY_TICK_FREQUENCY_MS);
```

```
          TimingDelay = mSecs+1;
          while (TimingDelay != 0);
     }

     // Do delay for nSecs microseconds
     void Delay_us(uint32_t uSecs) {
          SysTick_Config(SystemCoreClock                              /
DELAY_TICK_FREQUENCY_US);
          TimingDelay = uSecs+1;
          while (TimingDelay != 0);
     }
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Спецификация

ПРИЛОЖЕНИЕ В

(обязательное)

Перечень элементов

ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов