**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 7](#_Toc42377737)

[1 ОБЗОР ЛИТЕРАТУРЫ 8](#_Toc42377738)

[1.1 Умный дом от «Белтелеком» 8](#_Toc42377739)

[1.2 Умный дом Z-Wave 9](#_Toc42377740)

[1.3 Протокол Modbus 10](#_Toc42377741)

[1.4 Радиопередатчик для беспроводной связи 10](#_Toc42377742)

[1.5 Датчики температуры и влажности 11](#_Toc42377743)

[1.6 Датчик звука 12](#_Toc42377744)

[1.7 Датчик присутствия 12](#_Toc42377745)

[1.8 Датчик освещенности TCRT5000 13](#_Toc42377746)

[1.9 Датчик CO2 14](#_Toc42377747)

[1.10 Шаговый двигатель 14](#_Toc42377748)

[1.11 Микроконтроллер STM32F4ХХ 15](#_Toc42377749)

[1.12 Преобразователи напряжения DC/DC 17](#_Toc42377750)

[1.13 Интерфейс RS-485 17](#_Toc42377751)

[1.14 Гальваническая развязка 18](#_Toc42377752)

[2 РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ 19](#_Toc42377753)

[2.1 Описание основных блоков устройства 19](#_Toc42377754)

[2.2Блок преобразователя питания 19](#_Toc42377755)

[2.3 Блок датчиков 20](#_Toc42377756)

[2.4 Блок беспроводных датчиков 20](#_Toc42377757)

[2.5 Блок беспроводной связи 20](#_Toc42377758)

[2.6 Блок исполнительных устройств 21](#_Toc42377759)

[2.7 Блок управления 21](#_Toc42377760)

[2.8 Блок интерфейса взаимодействия с ПК 22](#_Toc42377761)

[2.9 Блок сервера 23](#_Toc42377762)

[3 РАЗРАБОТКА ФУНКЦИОНАЛЬНОЙ СХЕМЫ 24](#_Toc42377763)

[3.1 Блок сервера 24](#_Toc42377764)

[3.2 Блок преобразователя питания 25](#_Toc42377765)

[3.3 Блок датчиков 25](#_Toc42377766)

[3.3.1 Датчик освещенности 25](#_Toc42377767)

[3.3.2 Датчик присутствия 27](#_Toc42377768)

[3.3.3 Датчик температуры и влажности 28](#_Toc42377769)

[3.3.4 Датчик СО2 28](#_Toc42377770)

[3.4 Блок беспроводного интерфейса 29](#_Toc42377771)

[3.5 Блок беспроводных датчиков 29](#_Toc42377772)

[3.6 Блок управления 30](#_Toc42377773)

[3.6.1 Подсистема RCC 31](#_Toc42377774)

[3.6.2 Настройка портов ввода-вывода 35](#_Toc42377775)

[3.6.3 Подсистема WWDG 38](#_Toc42377776)

[3.6.4 Подсистема таймеров TIM 39](#_Toc42377777)

[3.6.5 Аналого-цифровой преобразователь 42](#_Toc42377778)

[3.6.6 Подсистема USART 46](#_Toc42377779)

[3.7 Блок исполнительного устройства 52](#_Toc42377780)

[4 РАЗРАБОТКА ПРИНЦИПИАЛЬНОЙ СХЕМЫ 53](#_Toc42377781)

[4.1 Обоснование выбора схемы и расчет дополнительных элементов 53](#_Toc42377782)

[4.1.1 Преобразование питания 53](#_Toc42377783)

[4.1.2 Интерфейс RS-485 54](#_Toc42377784)

[4.1.3 Выбор датчиков и их схема подключения 55](#_Toc42377785)

[4.1.4 Беспроводной интерфейс и его подключение 58](#_Toc42377786)

[4.1.5 Подключение беспроводных датчиков и исполнительных устройств 58](#_Toc42377787)

[4.2 Расчет потребления схемы и источников вторичного питания 59](#_Toc42377788)

[5 МОДЕЛИРОВАНИЕ 63](#_Toc42377789)

[5.1 Настройка окружения 63](#_Toc42377790)

[5.2 Моделирование работы RS-485 интерфейса 66](#_Toc42377791)

[5.3 Моделирование работы датчиков 68](#_Toc42377792)

[5.4 Алгоритм работы устройства 71](#_Toc42377793)

[6 ТЕХНИКО‑ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ПРОИЗВОДСТВА АППАРАТНОГО КОМПЛЕКСА СИСТЕМЫ АВТОМАТИЗАЦИИ УПРАВЛЕНИЯ ХОЗЯЙСТВОМ 74](#_Toc42377794)

[6.1 Характеристика аппаратно-программного комплекса 74](#_Toc42377795)

[6.2 Расчет экономического эффекта от производства аппаратно-программного комплекса 74](#_Toc42377796)

[6.3 Расчет инвестиций в проектирование и производство аппаратно-программного комплекса 77](#_Toc42377797)

[6.3.1 Расчет инвестиций на разработку аппаратно-программного комплекса 78](#_Toc42377798)

[6.3.2 Расчет инвестиций в прирост оборотного капитала 79](#_Toc42377799)

[ЗАКЛЮЧЕНИЕ 81](#_Toc42377800)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 82](#_Toc42377801)

[ПРИЛ,ОЖЕНИЕ А 84](#_Toc42377802)

[ПРИЛОЖЕНИЕ Б 96](#_Toc42377803)

[ПРИЛОЖЕНИЕ В 97](#_Toc42377804)

[ПРИЛОЖЕНИЕ Г 98](#_Toc42377805)

# 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 тестового фреймворка в соответствии с BDD концепцией.

Также целью данного дипломного проекта является проведение автоматизированного тестирования веб-приложения, находящегося в процессе разработки. Необходимо написать ряд тестовых сценариев, покрывающих всю функциональность приложения, на их базе разработать автотесты, а также благодаря CI утилитам сделать процесс проведения тестирования приложения полностью независимым от человека.

В соответствии с поставленной целью были определены следующие задачи:

− разработка фреймворка по автоматизированному тестированию;

* написание тест-кейсов для проверки функциональности веб-приложения в соответствии с концепцией behavior driven development;
* программная реализация тест-кейсов и проведение автоматизированного тестирования веб-приложения;
* автоматизирование процесса проведения тестирования.

# 1 ОБЗОР ЛИТЕРАТУРЫ

Для достижения поставленных целей по реализации фреймворка автоматизированного тестирования стоит рассмотреть уже существующие аналоги на рынке, а также ознакомиться с понятиями и технологиями, которые понадобятся при разработке вышеуказанного решения.

## Обзор существующих аналогов

Разработку любого программного продукта стоит начинать с обзора аналогов. Фреймворк по автоматизированному тестированию не является исключением.

На данный момент на рынке существует множество как бюджетных, так и коммерческих тестирующих фреймворков. Самый приближенным примером является Cypress. На сайте разработчиков можно найти цикл статей, позволяющих более детально ознакомиться с фреймворком[1].

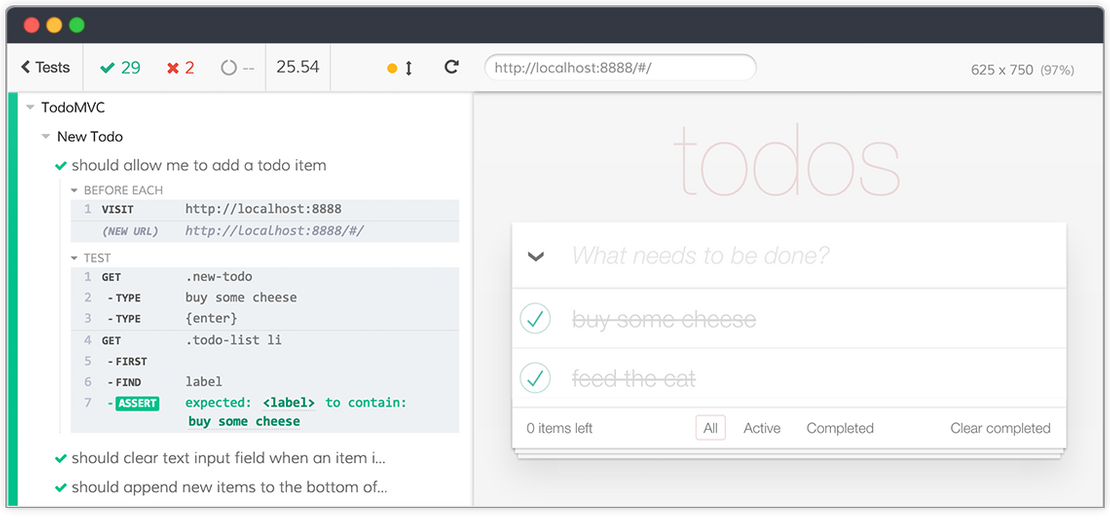


Рисунок 1.1 – Интерфейс фреймворка Cypress [1]

В основе Cypress лежит методология Test Driven Development. В отличие от прочих фреймворков, Cypress использует инновационный подход, основная идея которого заключается в том, что данный фреймворк является окружением для тестируемого продукта. Фактически, веб-приложение запускается в самом Cypress, что позволяет полностью контролировать процесс тестирования, взаимодействовать с любыми веб-элементами на высокой скорости, генерировать детальные отчеты в разных форматах.

Несмотря на все вышеупомянутые достоинства, Cypress также обладает рядом недостатков:

* Cypress позволяет писать тесты исключительно на языке JavaScript;
* В качестве тестовой среды Cypress поддерживает лишь среду Mocha;
* Cypress может взаимодействовать только с браузером Chrome.

Еще одним аналогом является фреймворк Citrus. Особенность данного фреймворка заключается в том, что Citrus позволяет проводить тестирование для любых протоколов обмена сообщениями или форматов данных. Citrus взаимодействует с такими технологиями, как Rest SOAP, JMS, HTTP и другие.

Citrus является open-source решением. Более детально ознакомиться с фреймворком можно на официальном сайте [2].

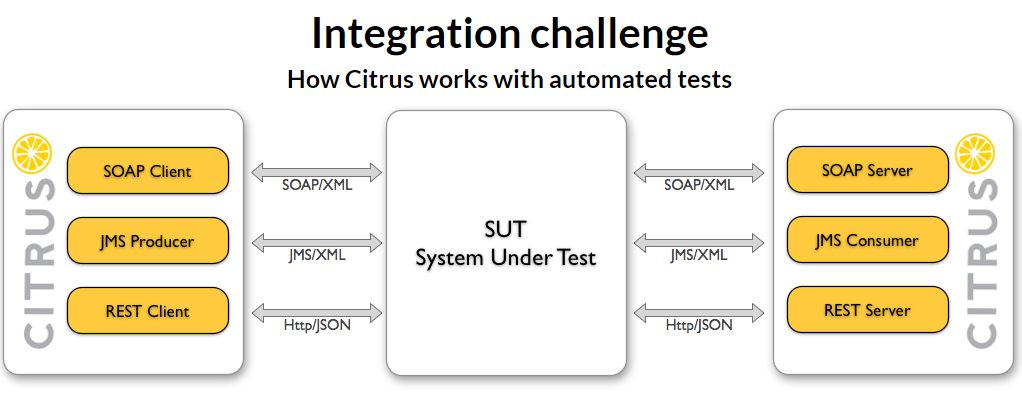


Рисунок 1.2 – Принцип работы фреймворка Citrus [2]

Citrus не создавался как самостоятельный инструмент проведения автоматизированного тестирования веб-приложений. Данный фреймворк является одним из лучших вариантов в случае необходимости проведения тестирования запросов, однако, на этом его функциональность ограничивается. Тем не менее, трудно назвать данную особенность недостатком фреймворка, так как тестирование UI-функциональности попросту не входит в концепцию его создания.

## 1.2 Понятие тестирования

Для написания тестового фреймворка в первую очередь необходимо разобраться с понятием тестирования. Существует множество различных определений, но все они сводятся к одному: тестирование – это проверка соответствия ожидаемого результата с фактическим. На сайте QA evolution [3] приведена подробная система классификаций, опираясь на которую можно выделить ряд понятий, относящихся к тестированию, под которое и разрабатывается автотестовый фреймворк:

* автоматизированное;
* регрессионное;
* модульное;
* функциональное;
* динамическое.

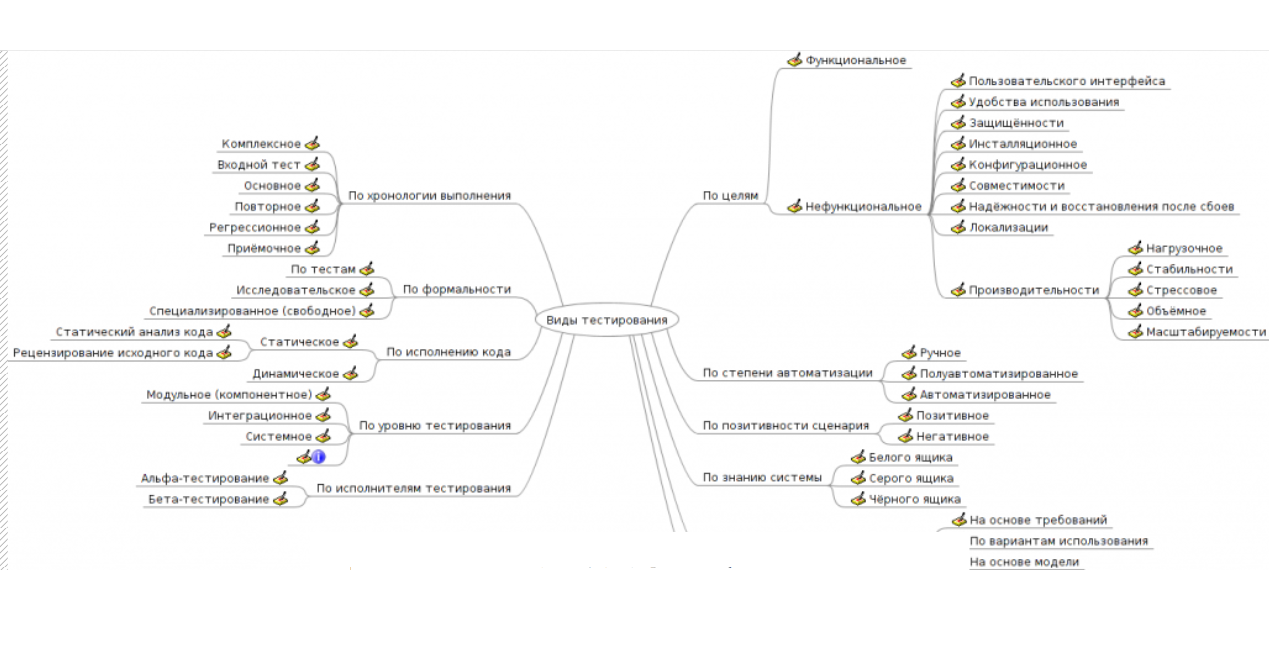


Рисунок 1.3 – Виды тестирования [3]

Стоит также иметь представление о процессах тестирования: по каким принципам разрабатываются тестовые сценарии, в каких ситуациях уместно использовать тестирование того или иного вида, как взаимодействовать с багами и многое другое. Во всех выше перечисленных вопросах поможет разобраться книга Романа Савина «Тестирование DOT COM» [4]. В книге приведено множество наглядных примеров, которые в полной мере отвечают на вопрос почему процесс обеспечения качества (quality assurance) настолько важно при разработке любого программного продукта.

## 1.4 Среда юнит-тестирования nUnit

Среди множества сред юнит-тестирования, использующихся на базе платформы NET, на данный момент выделяются две наиболее развитые: MSTest и nUnit. Документацию к MSTest можно найти на официальном сайте Microsoft [5], документацию к nUnit, соответственно, на сайте разработчиков среды [6]. Делать выбор стоит руководствуясь следующими факторами: функциональность, возможность генерации отчетов в различных форматах (таких как trx и html), удобство интеграции со сторонними инструментами. Учитывая все выше изложенные параметры, было решено остановится на nUnit. Не все актуальные фреймворки и окружения осуществляют поддержку MSTest. Также, nUnit обладает большим количеством полезных атрибутов, что значительно расширяет его функциональность.

## 1.5 Инструмент для автоматизации Selenium WebDriver

Фреймворк ориентирован на автоматизацию веб-приложений. Для достижения поставленной задачи необходимо выбрать соответствующий инструмент взаимодействия с веб-браузером. Selenium WebDriver является наиболее подходящим вариантом. Среди прочих альтернатив на рынке он обладает следующими преимуществами:

* широкий функционал – Selenium предоставляет возможность взаимодействия со всеми элементами веб-страницы, вызывать определенные функции браузера (работа с всплывающими окнами, вкладками, cookie и многое другое), использовать различные JavaScript-методы;
* кроссплатформенность – Selenium работает на разных языковых платформах, а также дает возможность взаимодействовать со всеми актуальными на сегодняшний день браузерами: Chrome, Firefox, Safari, Edge, Internet Explorer;
* Selenium WebDriver является бесплатным решением.

Всю необходимую документацию можно найти на официальном сайте разработчиков [7]. В качестве альтернативы можно воспользоваться циклом статей с русскоязычного сайта «kreisfahrer» [8], рассказывающих об особенностях работы с Selenium’ом. Здесь в первую очередь стоит обратить внимание на статью, посвященную работе с так называемыми ожиданиями. Ожидания позволяют обрабатывать ситуации, при которых элементы веб-страницы загружаются с медленной скоростью или не загружаются вовсе. Грамотная расстановка ожиданий в фреймворке улучшит скорость его работы, а также качество тестов, написанных с его помощью.

Еще одна полезная статья описывает все виды локаторов, с которыми может взаимодействовать Selenium. Локаторы – адреса веб-элементов, которые позволяют взаимодействовать с ними на уровне программного фреймворка. Selenium предоставляет инструменты для работы со следующими видами локаторов:

* XPath – язык запросов к элементам XML;
* CssSelector – позволяет находить элементы с определенными CSS-свойствами;
* Id – поиск элемента по уникальному идентификатору (если таковой есть в верстке веб-страницы);
* ClassName – поиск всех элементов в верстке веб-страницы, принадлежащих к определенному классу;
* LinkText – поиск элемента по ссылке, указывающей на адрес данного элемента;
* прочие, менее широко применяемые локаторы: Name, TagName, PartialLinkText.

Работа с локаторами является крайне важной частью при проведении автоматизированного тестирования любого веб-приложения. Вся сложность заключается в том, что верстка тестируемого сайта может меняться в процессе разработки. Отсюда вытекает необходимость правильного выбора локаторов, которые будут стабильно указывать на необходимые веб-элементы даже при условиях изменения верстки веб-приложения.

## 1.6 RestSharp

Большинство веб-приложений так или иначе реализовывают базу API-запросов. API-запросы необходимы для взаимодействия клиентской и серверной частей приложения. Также API-запросы позволяют оперативно пользоваться сторонними сервисами (примеры: Cloudinary, TestRail, Amazon и другие).

Возможность реализации API-запросов влечет за собой необходимость проверки их корректной работоспособности. Именно поэтому фреймворк по автоматизированному тестированию должен содержать в себе функциональность, позволяющую тестировать API-запросы.

Среди множества библиотек языка C#, реализующих API-запросы, для поставленной задачи больше других альтернатив подходит библиотека RestSharp. Данная библиотека позволяет не только реализовывать POST, GET, PUT и прочие запросы, но и проводить сериализацию и десериализацию в соответствии с форматами json и xml. С этими и прочими особенностями библиотеки более детально можно ознакомиться на официальном сайте RestSharp [9].

## 1.7 Концепция Behavior Driven Development

Подход BDD возник на основе другой, более распространенной концепции – Test Driven Development (TDD). Суть данной концепции заключается в том, что тесты для разрабатываемого приложения пишутся еще до непосредственного создания самого приложения. TDD имеет как положительные, так и отрицательные стороны. Среди плюсов ключевым является тот факт, что тесты при таком подходе участвуют в разработке приложения как такового. Множество дефектных ситуаций обрабатывается еще до начала имплементации логики, что может заметно ускорить разработку и улучшить ее качество.

Однако, такой подход сильно замедляет разработку самого приложения. Многие разработчики считают написание тестов еще до создания кода приложения неоправданно сложным процессом. В первую очередь это связано с формированием требований: не всегда ясно что и как необходимо тестировать. Именно для этого было создано BDD. Отличие Behavior Driven Development от Test Driven Development заключается в том, что при первом подходе больший уклон делается именно на создание требований, на основе которых уже и пишутся тесты для приложения. Подробнее про особенности использование BDD можно прочитать в книге Джона Смарта «BDD in action: Behavior-driven development for the whole software lifecycle» [10]. Также много важной информации можно найти на сайте компании по предоставлении услуг обеспечения качества A1QA [11].

Ключевой особенностью данного подхода стала необходимость четкого взаимодействия между разработчиками бизнес-логики приложения и разработчиками программного кода. Нередко люди первого типа не являются высококвалифицированными программистами, и, напротив, людям второго типа трудно сходу воспринимать бизнес-требования. Возникла необходимость создания единого языка, понятного в равной степени для всех. Таким языком стал Gherkin. Именно на языке Gherkin при использовании таких ключевых слов, как «Scenario» «Given», «When» и «Then» будут писаться тестовые сценарии на базе разрабатываемого фреймворка. На рисунке 1.4 предоставлена образная схема тестового сценария, написанного при поддержке данного языка.



Рисунок 1.4 – Схема тестового сценария на языке Gherkin [12]

## 1.8 Specflow

Specflow служит инструментом реализации BDD подхода на базе языка программирования C#. Принцип работы Specflow заключается в следующем: тестовые сценарии, написанные на языке Gherkin, хранятся в файлах с расширением «feature»; команды «Given», «When» и «Then» реализовываются на программном уровне при помощи соответствующих атрибутов; по необходимости добавляются хуки (предусловия и постусловия), а также необходимые теги в «feature» файлы.

С всей необходимой документацией по Specflow, а также примерами использования можно ознакомиться на официальном сайте разработчиков [13].

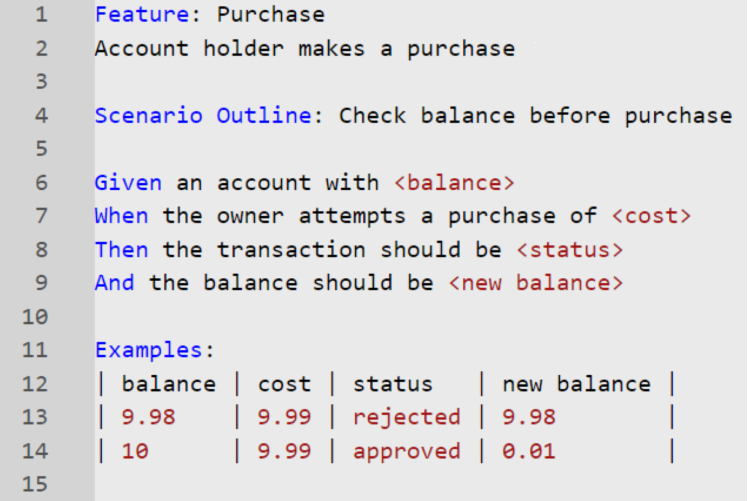


Рисунок 1.5 – Пример «feature» файла [13]

## 1.9 Понятие CI

Процесс проведения автоматизированного тестирования любого приложения по концепции BDD подразумевает постоянный, непрекращающийся запуск разработанных автотестов. Отсюда вытекает необходимость создания условий автоматизации самого процесса тестирования. Для подобных целей была создана методология непрерывной интеграции (Continuous Integration).

Концепция методологии CI проста и эффективна. В ходе создания приложения разработчикам постоянно приходится имплементировать новую функциональность. В связи с этим необходимо все время проверять корректную работоспособность программного продукта. После добавления новых изменений (или при прочих условиях – например, в случае обновления окружения), стоит проверять все приложение на наличие появившихся дефектов и багов. Для подобных целей и применяется автоматизированное тестирование. Ручной подход при таком методе создания программного продукта занимал бы невероятно много человеческого времени. В то же время, при наличии разработанных автотестов нет необходимости в задействовании человека вовсе. Вся функциональность приложения проверяется автоматически, что позволяет разработчикам в случае обнаружения неисправностей оперативно найти их источник и решить возникшую проблему.

В своей книге «Continuous Integration: Improving Software Quality and Reducing Risk» [14] Стив Матыас и Эндрю Гловер детально изложили особенности разработки по методологии непрерывной интеграции. Базовую информацию, дающую общее представление о том, что такое CI, можно найти в одноименной статье на сайте «Хабр» [15].

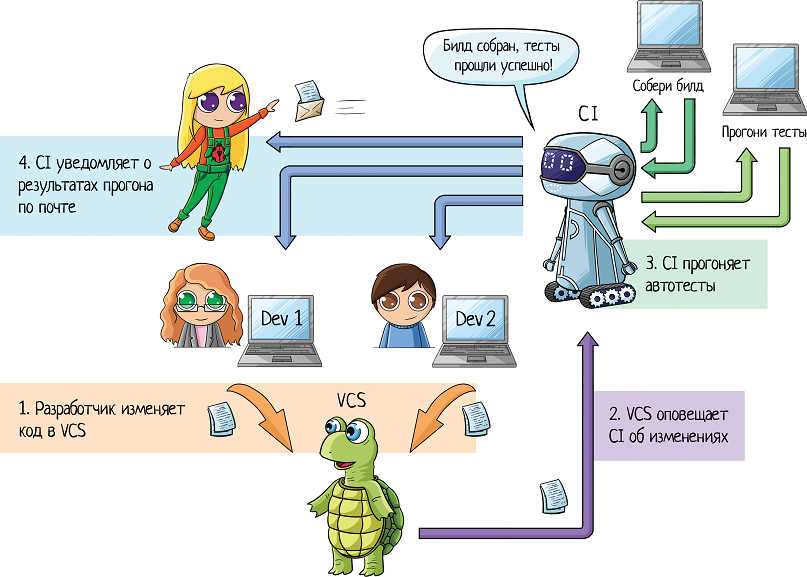


Рисунок 1.6 – Пример взаимодействия по методологии CI [15]

## 1.10 Jenkins

Jenkins – инструмент непрерывной интеграции, позволяющий реализовать на практике все принципы данной методологии. Jenkins оперирует понятием «Jobs» - процессы, отвечающие за сборку и запуск программных проектов, хранящихся в любой системе контроля версий (например, git), или с любого подключенного машинного устройства (при должном подключении и наличии необходимого окружения). Jenkins позволяет осуществлять сборку проекта по следующим критериям:

* сборка по запуску «Job»’ы (разработчик запускает сборку проекта вручную);
* сборка по ссылке (переход по ссылке осуществляет запуск сборки);
* сборка по изменению состояния репозитория в системе контроля версий (пример: добавление коммита в git-репозиторий);
* сборка по расписанию.

Последнему способу сборки стоит уделить отдельное внимание. Для реализации сборки по расписанию Jenkins использует так называемые «крон-выражения». На рисунке 1.7 представлена схема, взятая с официального сайта разработчиков [16]. Например, выражение «\* \* 1,15 1-11 \*» будет осуществлять сборку по одному разу первого и пятнадцатого числа каждого месяца, кроме декабря.

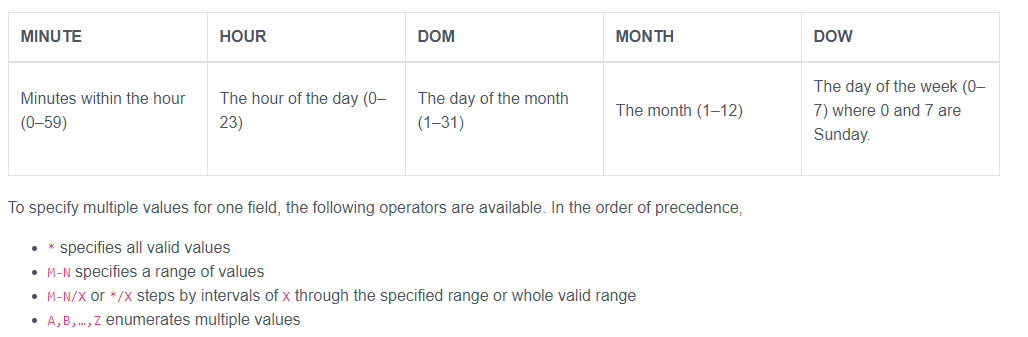


Рисунок 1.7 – Описание синтаксиса «крон-выражений» [16]

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив предметную область разрабатываемого тестового фреймворка, 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 разрабатываемом фреймворке по автоматизированному тестированию были выделены следующие блоки:

− блок конфигурации браузера;

− блок веб-элементов браузера;

− блок API-функциональности;

− блок расширений;

− блок тестовых сценариев;

− блок тестовой функциональности;

− блок модели тестируемого приложения.

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

Каждый из блоков содержит в себе определенную функциональность, необходимую для корректной работы фреймворка и стабильного запуска автотестов. Рассмотрим более детально работу и задачи каждого блока отдельно, а также их взаимодействие и обмен данных между блоками.

## 2.2Блок конфигурации браузера

Блок конфигурации браузера является сердцем автотестового фреймворка. Здесь находится логика, отвечающая за создание объекта браузера, его настройку, взаимодействие с инструментами самого браузера (например, переключение между вкладками, работа с всплывающими окнами, взаимодействие с куками, вызов JavaScript-методов, открытие и закрытие браузера).

Объект браузера создается в соответствии с параметрами, указанными в ресурсных файлах. Разработчик автотестов может выбрать один из четырех видов браузера, в котором будет проводится тестирование веб-приложения: Chrome. Safari, Firefox или Edge. Предоставляется возможность настройки времени ожидания открытия веб-страницы или появления элемента на ней, можно запускать тесты в headless (фоновом) режиме.

Данный блок взаимодействует с блоками веб-элементов и веб-сущностей браузера, а также с блоком тестовой функциональности. Первые два используют инструменты браузера в целях взаимодействия с объектами веб-страниц. В блоке тестовой функциональности реализованы хуки (запуск и закрытие браузера).

## 2.3 Блок веб-элементов браузера

Фреймворк осуществляет взаимодействие с веб-страницами браузера через элементы, из которых они состоят. К наиболее часто используемым веб-элементам относятся:

− текстовые поля (поля для ввода текстовой информации);

− лейблы (неизменяемые текстовые поля страницы – заголовки, статьи и тому подобное);

− чекбоксы (элементы управления параметрами в двух состояниях - включение и отключение);

− комбобоксы (поля выбора одного из нескольких вариантов);

− кнопки.

У всех элементов есть схожие свойства. Например, на все элементы можно нажать кнопкой мыши, у всех элементов можно взять размеры и координаты, а также их состояния на странице (можно ли кликнуть на элемент, виден ли элемент на веб-странице, существует ли элемент вовсе). В то же время, каждый элемент должен обладать собственной функциональностью. В текстовое поле, в отличие от кнопки, можно передавать определеные значения; комбобокс позволяет выбирать 1 из вариантов, содержащихся в нем и так далее.

Данный блок тесно взаимодействует с блоком страниц тестируемого приложения – все страницы так или иначе содержат в себе различные элементы, через взаимодействие с которыми будет осуществляться тестирование веб-приложения.

## 2.4 Блок API-функциональности

Блок API-функциональности содержит в себе логику реализации API-запросов. Данный блок должен решать две задачи. Во-первых, необходимо предоставить инструментарий для создания различных типов запросов с любыми входными параметрами. Должна быть реализована поддержка передачи данных не только в текстовом, но и в графическом формате. Для достижения данной цели стоит разработать механизм, преобразующий изображение в формат «multipart/form-data», и способный совершать обратный процесс.

Второй целью данного блока является реализация механизмов сериализации и десериализации по двум наиболее часто используемым на сегодняшний день форматам обмена данных посредством API механизма: json и xml. Все запросы должны быть представлены в виде моделей на программном уровне для удобного взаимодействия с ними.

Данный блок должен быть разработан в первую очередь для проведения тестирования API-функциональности, следовательно, он имеет прямое взаимодействие с блоком тестовой функциональности.

## 2.5 Блок расширений

Данный блок содержит в себе множество полезных утилит, которые используются в различных прочих блоках фреймворка. В первую очередь здесь присутствует функциональность логирования. Это необходимо для генерации максимально детальных и точных тестовых отчетов. В случае получения негативного результата теста, грамотная система логирования поможет быстро разобраться в каком месте веб-приложение функционирует некорректно. Логи должны быть задействованы во всех блоках автотестового фреймворка. Особенно это актуально для блока веб-элементов. Отчеты должны содержать в себе подробную информацию о том, какие действия были совершены с теми или иными элементами веб-страницы.

Другой немаловажной частью блока расширений является функциональность, направленная на взаимодействие с ресурсами. Тут стоит упомянуть два важных фактора. Во-первых, работа с ресурсами позволяет получать всю необходимую информацию для конфигурации тестового фреймворка и самих тестов. Во-вторых, ресурсные файлы могут хранить в себе данные, используемые для проведения тех или иных автотестов. Хорошим примером являются креды веб-приложения – логины и пароли пользователей с различными правами доступа.

Также в блоке расширения должны содержаться инструменты взаимодействия с файлами, изображениями, датами и прочими подобными сущностями, так как это значительно упростит процесс написания автотестов.

## 2.6 Блок тестовых сценариев

Данный блок должен напрямую иллюстрировать behavior driven development концепцию в действии. Весь блок будет состоять из «feature»-файлов, написанных при поддержке фреймворка Specflow.

Каждый из «feature»-файлов должен содержать в себе тестовый сценарий, написанный на языке Gherkin. Все тестовые сценарии должны покрывать основную функциональность тестируемого приложения.

В соответствии с языком Gherkin, «feature»-файлы содержат в себе так называемые степы тестирования. «Given» степы отвечают за создание окружения, в котором будет проходить тест. «When» степы задают последовательность действий, которая должна привести к определенному результату. «Then» степы должны проводят само тестирование, то есть, проверяют соответствие ожидаемого результата с фактическом.

Все тесты должны быть заимплементированы в блоке тестовой функциональности. Таким образом, данный блок выполняет две принципиально важные для проведения тестирования задачи: содержит в себе всю тестовую документацию, участвует в программной разработке самих автотестов.

## 2.7 Блок тестовой функциональности

Блок тестовой функциональности содержит в себе сами тесты в виде программного кода. Каждому степу из «feature» файлов соответствует определенный метод, который должен выполнять соответствующую функцию. Отсюда возникает необходимость разработать четкую структуру и разделить все тестовые методы на группы, соответствующие их назначению («given», «when» и «then» методы).

Особое внимание стоит уделить логике, отвечающей за конфигурацию тестового окружения. Необходимо разработать множество хуков, которые будут контролировать открытие браузера в соответствии с заданными параметрами, его корректное закрытие с очисткой использовавшихся ресурсов, реализовывать ряд прочих преднастроек, которые будут актуальны для многих тестов (переход на необходимую страницу браузера, получение данных из ресурсных файлов и тому подобное).

По понятным причинам блок тестовой функциональности наиболее тесно взаимодействует с блоком тестовых сценариев. По факту, блок тестовой функциональности всего лишь инструмент, который разрабатывается для реализации логики, хранящейся в «feature» файлах. Логика взаимодействия с браузером берется из блока конфигурации браузера.

Еще один блок, с которым взаимодействует тестовая функциональность – блок модели тестируемого приложения. Тестовые степы в большинстве случаев должны лишь собирать в себе методы, которые реализовывают страницы в соответствии со своей функциональностью.

## 2.8 Блок модели тестируемого приложения

Блок модели тестируемого приложения – полная абстракция сайта, над которым необходимо провести автоматизированное тестирование. Каждая страница, существующая в веб-приложении, должна иметь свою модель в рамках автотестового фреймворка. Сами же модели должны хранить в себе объекты страницы. Отсюда появляется связь с блоком веб-элементов браузера. Более того, так как все страницы-модели создаются для проверки функциональности своих оригиналов, необходимо добавить возможность скриптового воспроизведения данной функциональности. Хорошим примером тут может стать форма авторизации. В модели необходимо создать объекты страницы – текстовые поля логина и пароля, кнопку подтверждения, прочие элементы в случае их наличия на странице. Также модель должна содержать в себе функцию проведения авторизации – заполнение полей логина и пароля, нажатие на кнопку подтверждения. В целях проведения тестирования функциональности страницы авторизации, стоит также добавить методы проверки отображения как элементов страницы, так и самой страницы.

Структура модели веб-приложения должна соответствовать своему действительному аналогу. В случае наличия связей, одни страницы должны ссылаться на другие, общие сущности необходимо выносить в отдельные модели, а страницы их содержащие должны иметь методы перехода на данные сущности. Для достижения данных целей хорошей практикой будет использование паттерна «компоновщик». Возвращаясь к примеру со страницей авторизации, ее модель также должна содержать метод перехода на другую страницу, которая открывается в случае совершения успешной авторизации.

Как было упомянуто выше, блок модели тестируемого приложения должен предоставлять всю необходимую функциональность для блока тестовой функциональности. Основная идея заключается в том, что созданная абстракция тестируемого сайта должна содержать в себе эмуляцию любых действий реального пользователя, которые затрагивают функциональность веб-приложения.

# 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 STM32F4ХХ. 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 ICB при неизменной величине светового потока, а также тока в темноте Iтем 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.

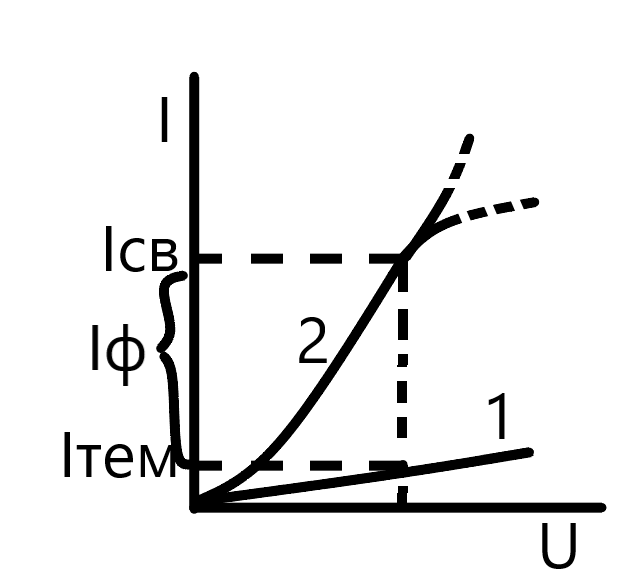


Рисунок 3.1 – Вольт-амперная характеристика полупроводника

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.

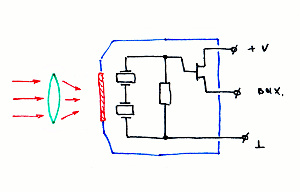


Рисунок 3.2 – Принцип работы датчика обнаружения присутствия

### 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.

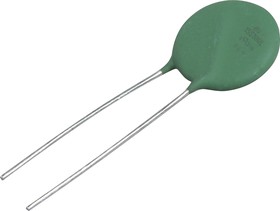


Рисунок 3.3 – Термистор [17]

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

### 3.3.4 Датчик СО2

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.

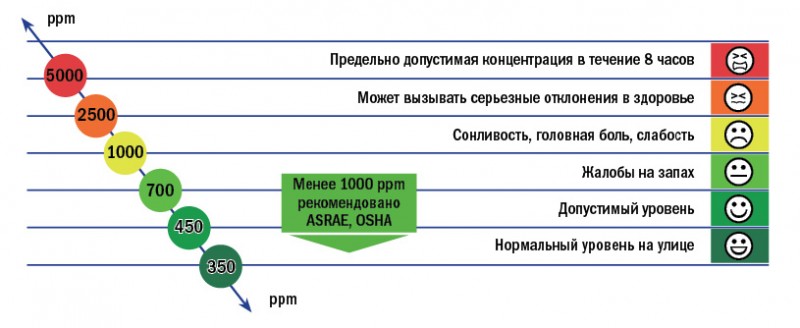


Рисунок 3.4 – Графики допустимых значений ppm [18]

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.

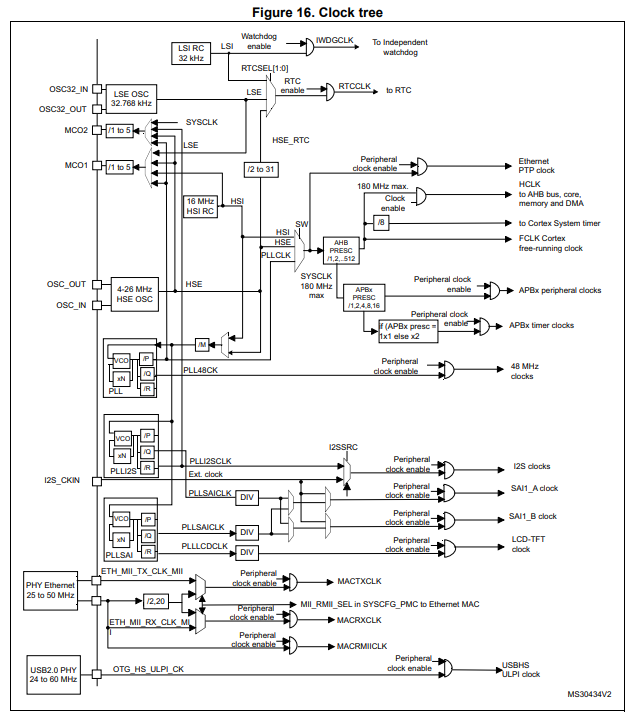


Рисунок 3.5 – Представление системы тактирования STM32F4XX [13]

#### 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 C. 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 C. 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 C. 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 C. 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.

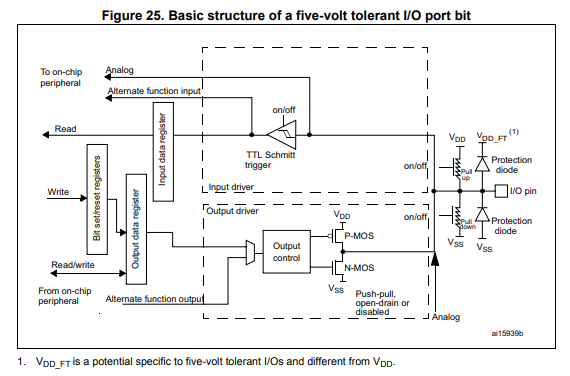


Рисунок 3.6 – Схема работы цифро-аналогового порта ввода-вывода [13]

#### 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 C. 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 C. 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 C. 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 C. 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 C. 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 C. 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 C. Данный аргумент является начальным значением сторожевого таймера. Диапазон значений данного аргумента от 0х40 до 0х7F, 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 C. Данный аргумент устанавливает граничное значение, с которым будет сравниваться значение счетчика с направлением счета вниз. Аргумент должен иметь значение меньше 0x80.

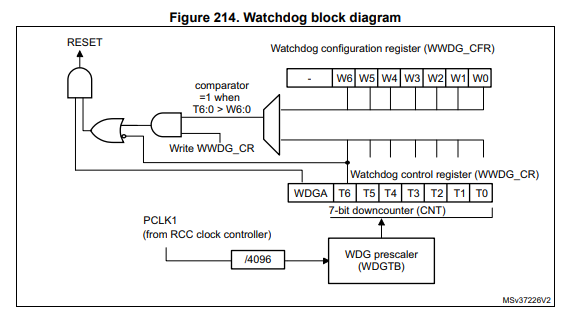


Рисунок 3.7 – Блок диаграмма сторожевого таймера [13]

### 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\_СenterAlignedX, 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, х – 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, х – 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.

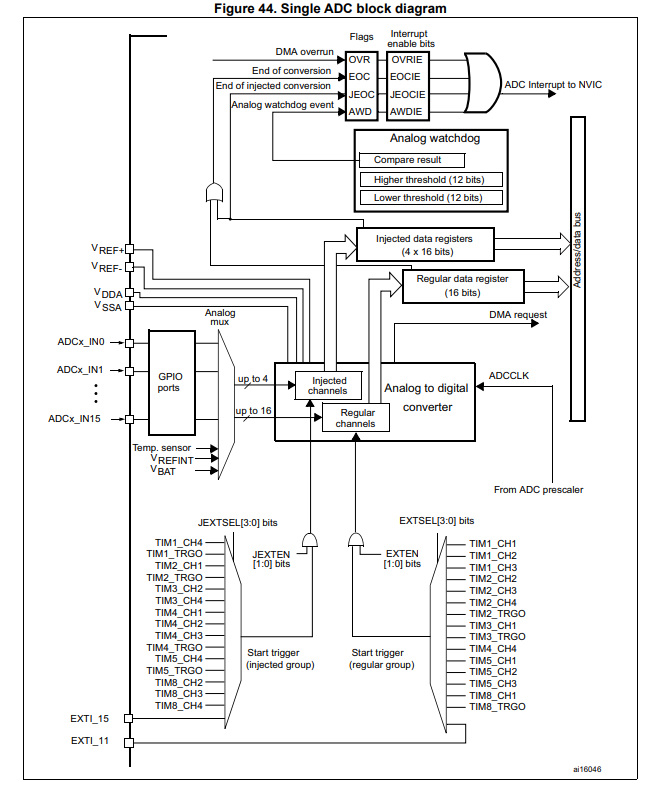


Рисунок 3.8 – Блок диаграмма АЦП STM32F4XX [13]

#### 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, х – 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, х – 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, х – 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 C. 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, х – 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, х – может принимать значение от 1 до 3.

Аргумент ADC\_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 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 C. 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, х – может принимать значение от 1 до 3.

Аргумент ADC\_FLAG имеет тип uint8\_t, который определен в стандартной библиотеке stdint.h языка C. 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, х – может принимать значение от 1 до 3.

Аргумент ADC\_FLAG имеет тип uint8\_t, который определен в стандартной библиотеке stdint.h языка C. 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.

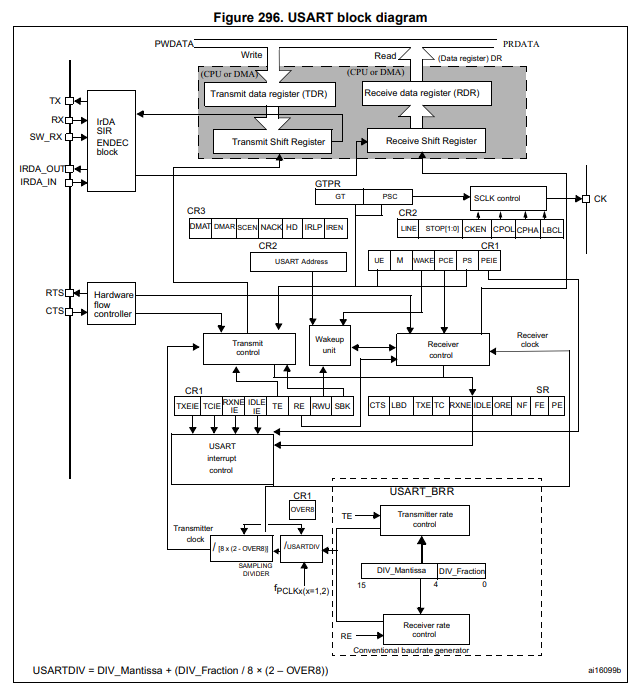


Рисунок 3.9 – Cистема ассинхронной приемопередачи данных STM32F4XX [13]

#### 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 C. 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 C. 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 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 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 C. 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 C. 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 C. 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.

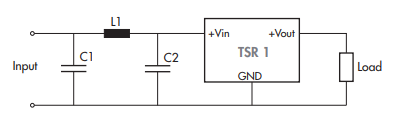


Рисунок 4.1 – Схема подключения TCR 1-2450 [14]

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.

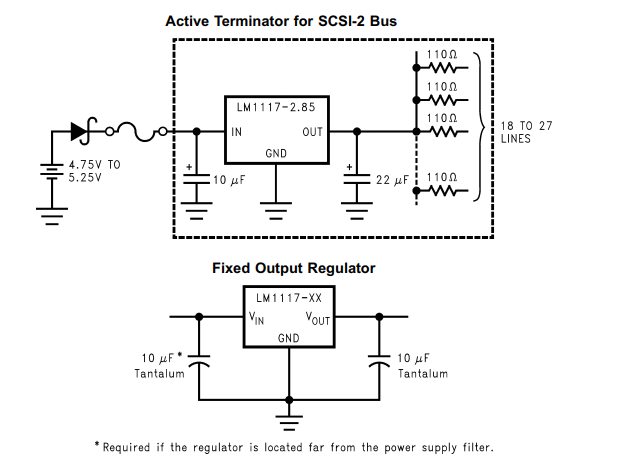


Рисунок 4.2 – Схема подключения LM1117 [16]

### 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.

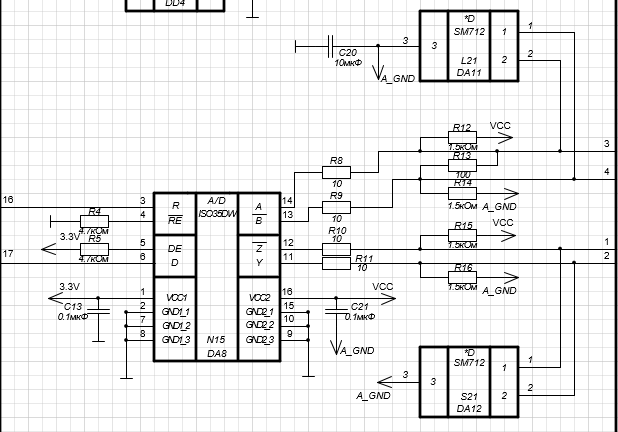


Рисунок 4.3 – Гальваническая развязка

#### 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.

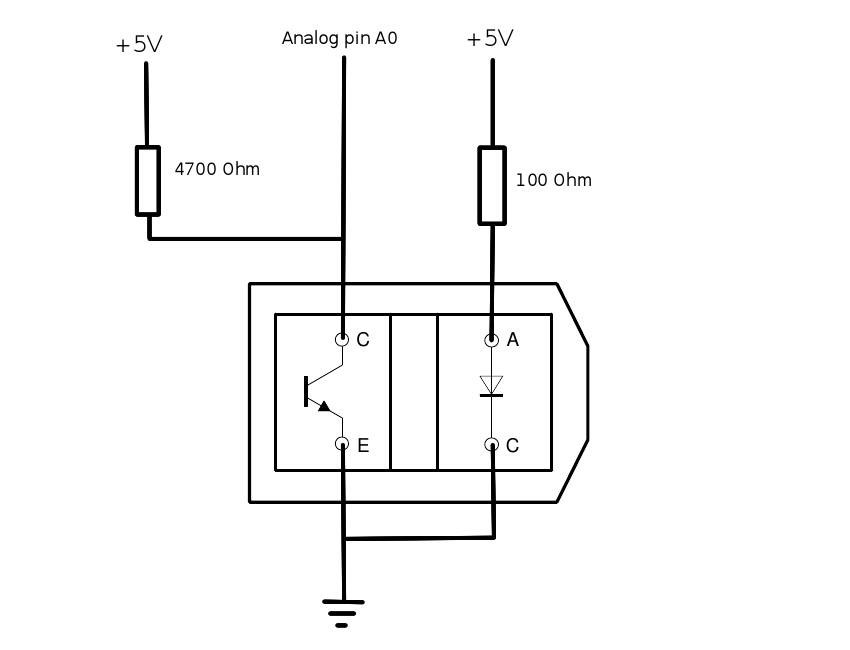


Рисунок 4.4 – Схема подключения датчика TCRT5000

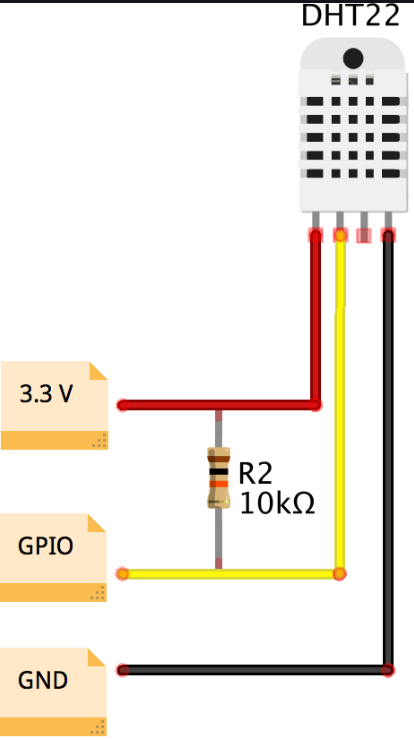


Рисунок 4.5 – Схема подключения датчика DHT22

#### 4.1.3.4 Датчик СО2

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.

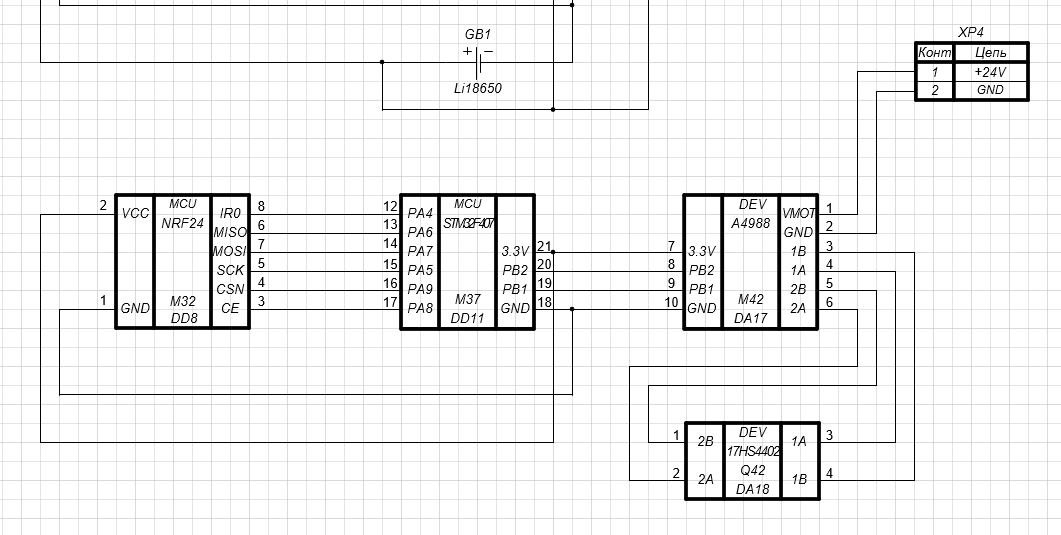


Рисунок 4.6 – Типичная схема подключения драйвера

## 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.

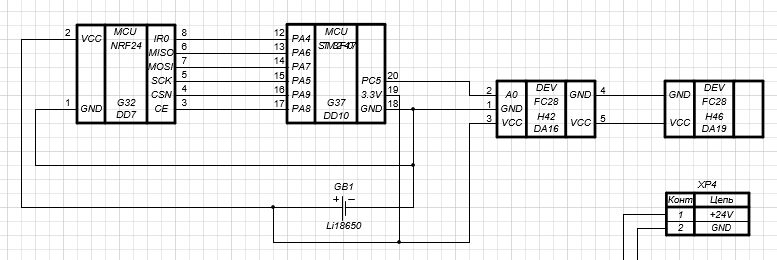


Рисунок 4.7 – Схема включения датчика влажности

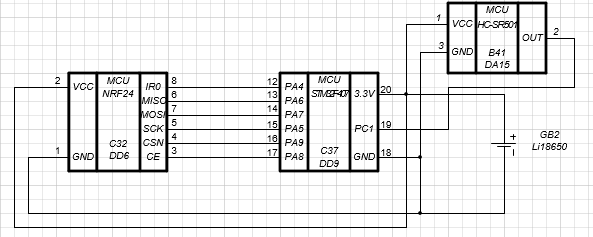


Рисунок 4.8 – Схема включения датчика присутствия

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:

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

Суммарная потребляемая мощность модуля составляет:

где – потребляемая мощность от 3.3 В,

– потребляемая мощность от 5 В.

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:

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

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

где – продолжительность работы аккумуляторной батареи в часах,

– емкость аккумулятора в А·ч,

– ток нагрузки в А,

– 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:

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.

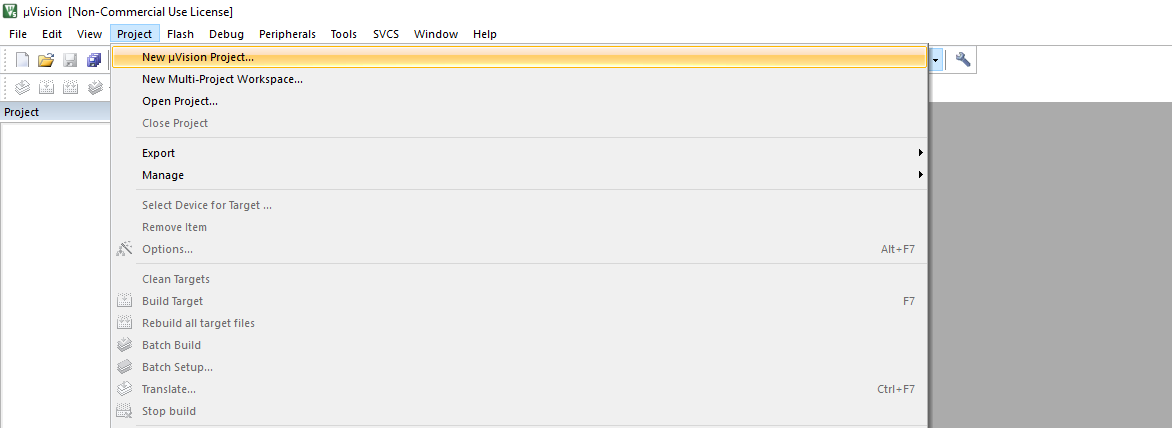


Рисунок 5.1 – Создание проекта - 1

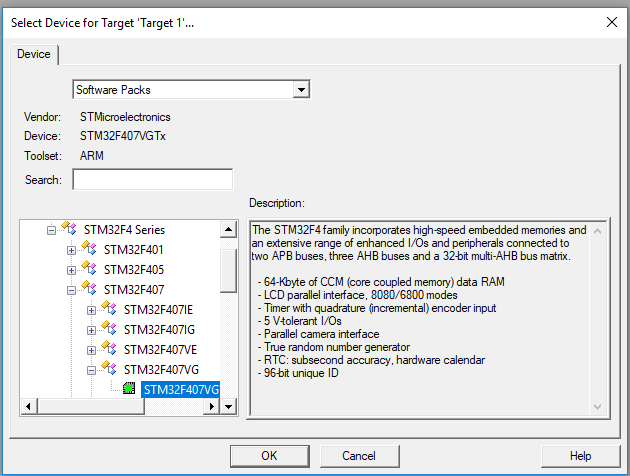


Рисунок 5.2 – Создание проекта - 2

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

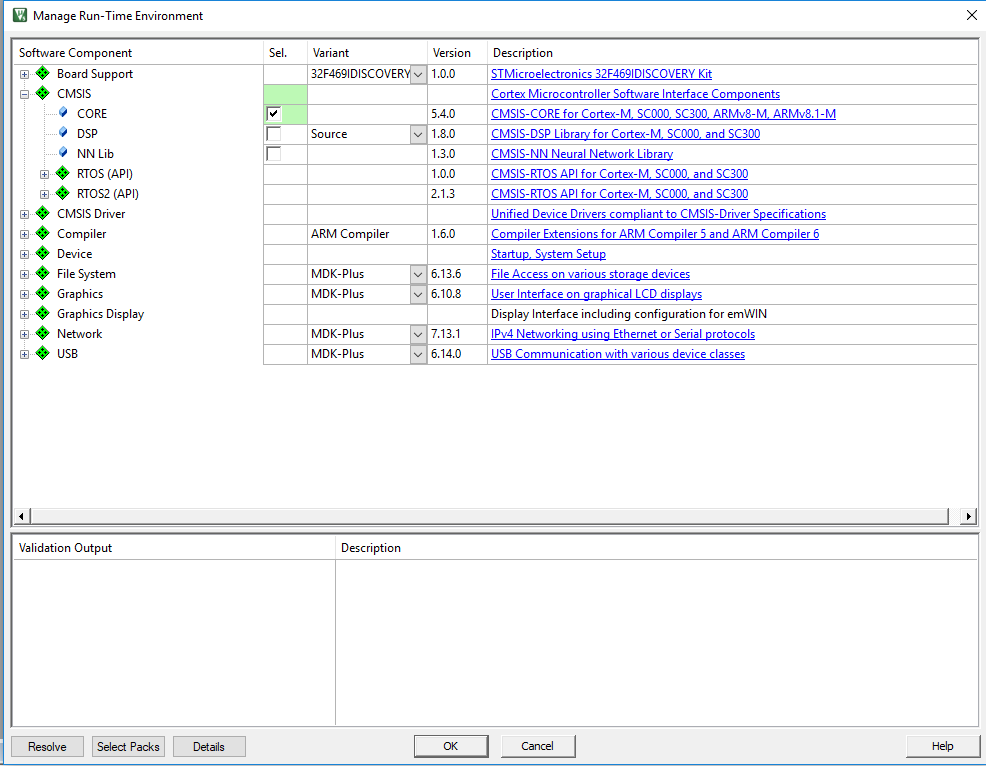


Рисунок 5.3 – Создание проекта - 3

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.

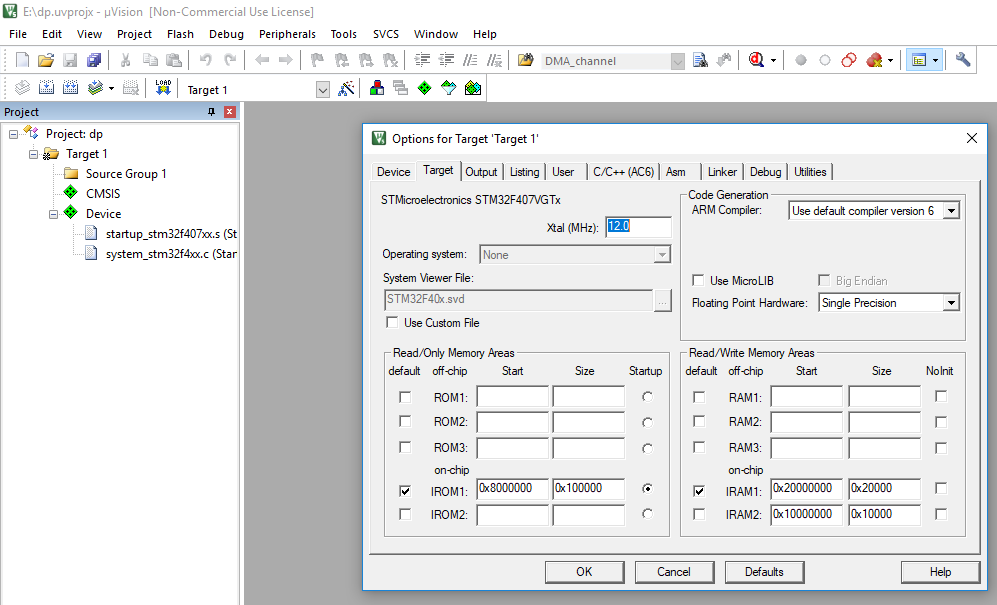


Рисунок 5.4 – Подключение директориев библиотек

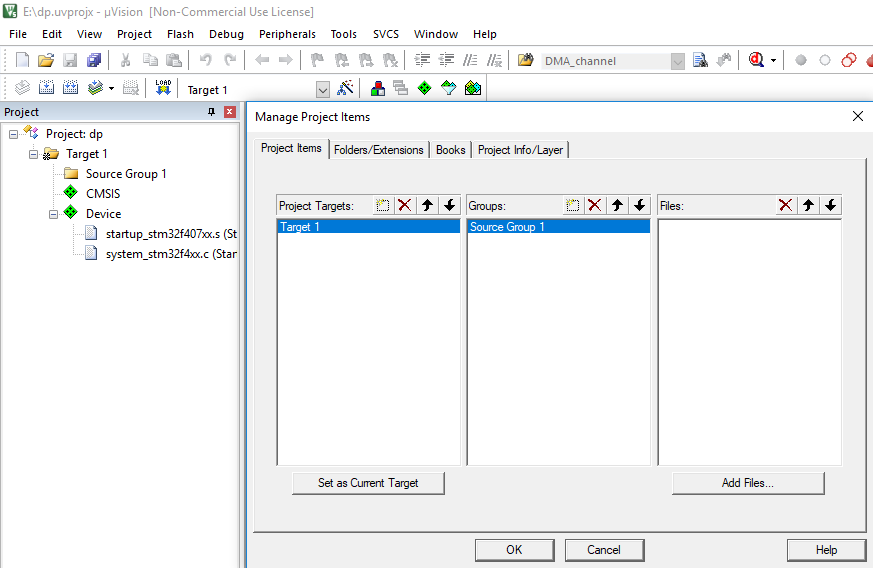


Рисунок 5.5 – Подключение файлов в проект

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 C. 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.

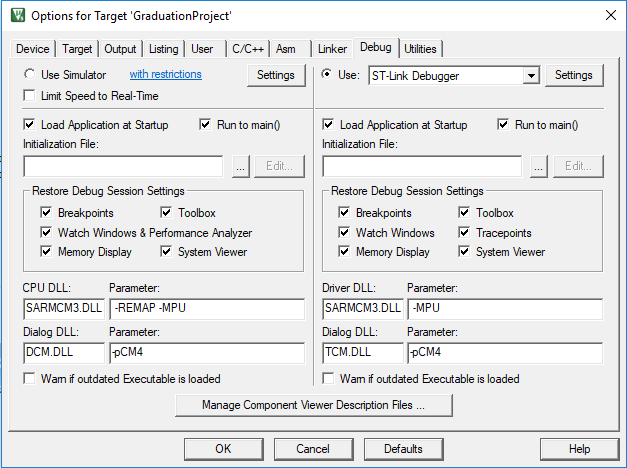


Рисунок 5.6 – Вкладка настройки отладчика

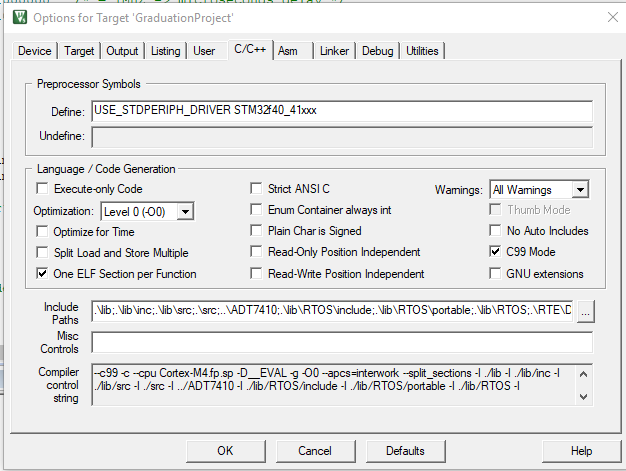


Рисунок 5.7 – Вкладка настройки языка C/C++

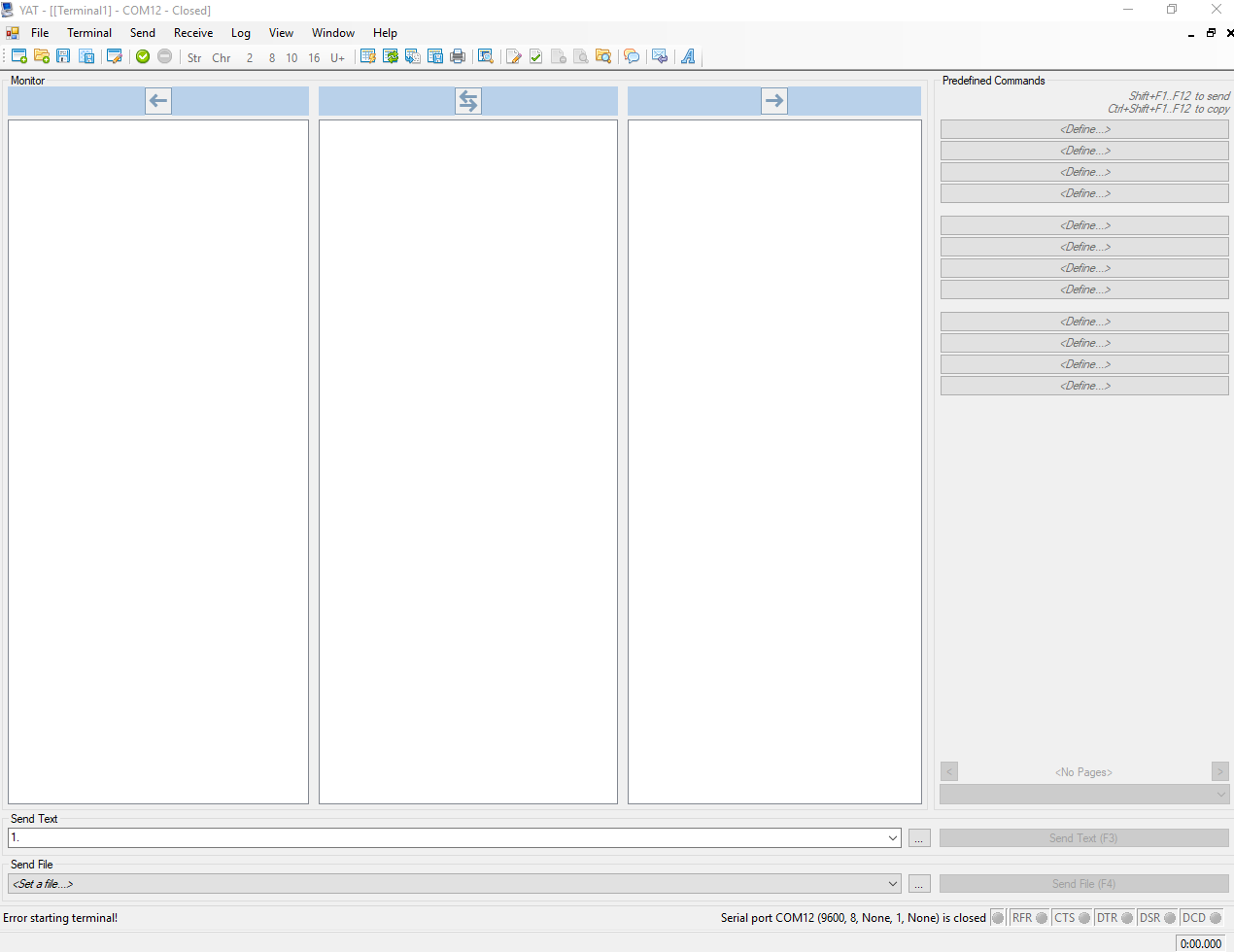


Рисунок 5.8 – Терминальная программа YAT

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.

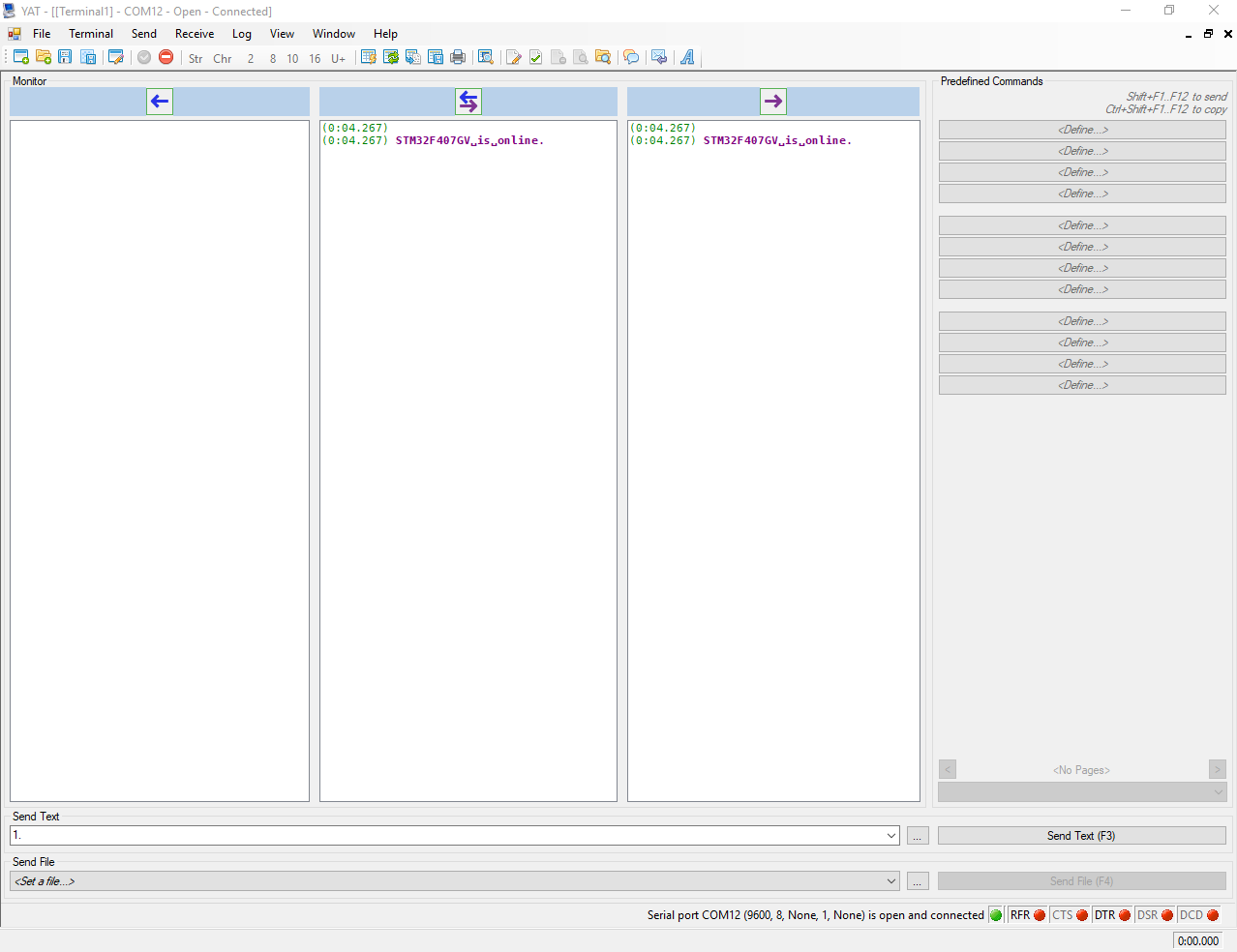


Рисунок 5.9 – Подключение к устройству

## 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.

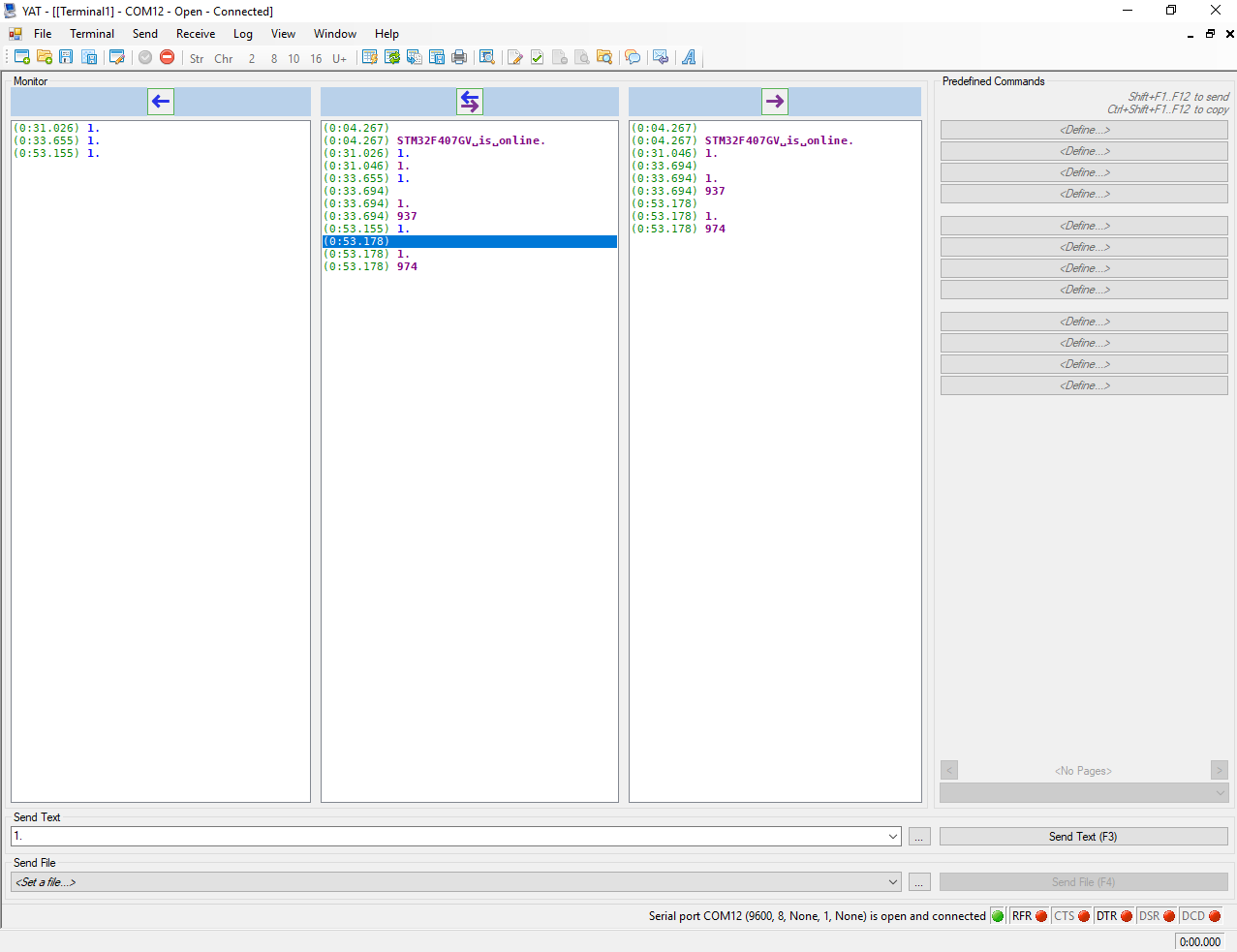


Рисунок 5.10 – Получение информации об освещенности

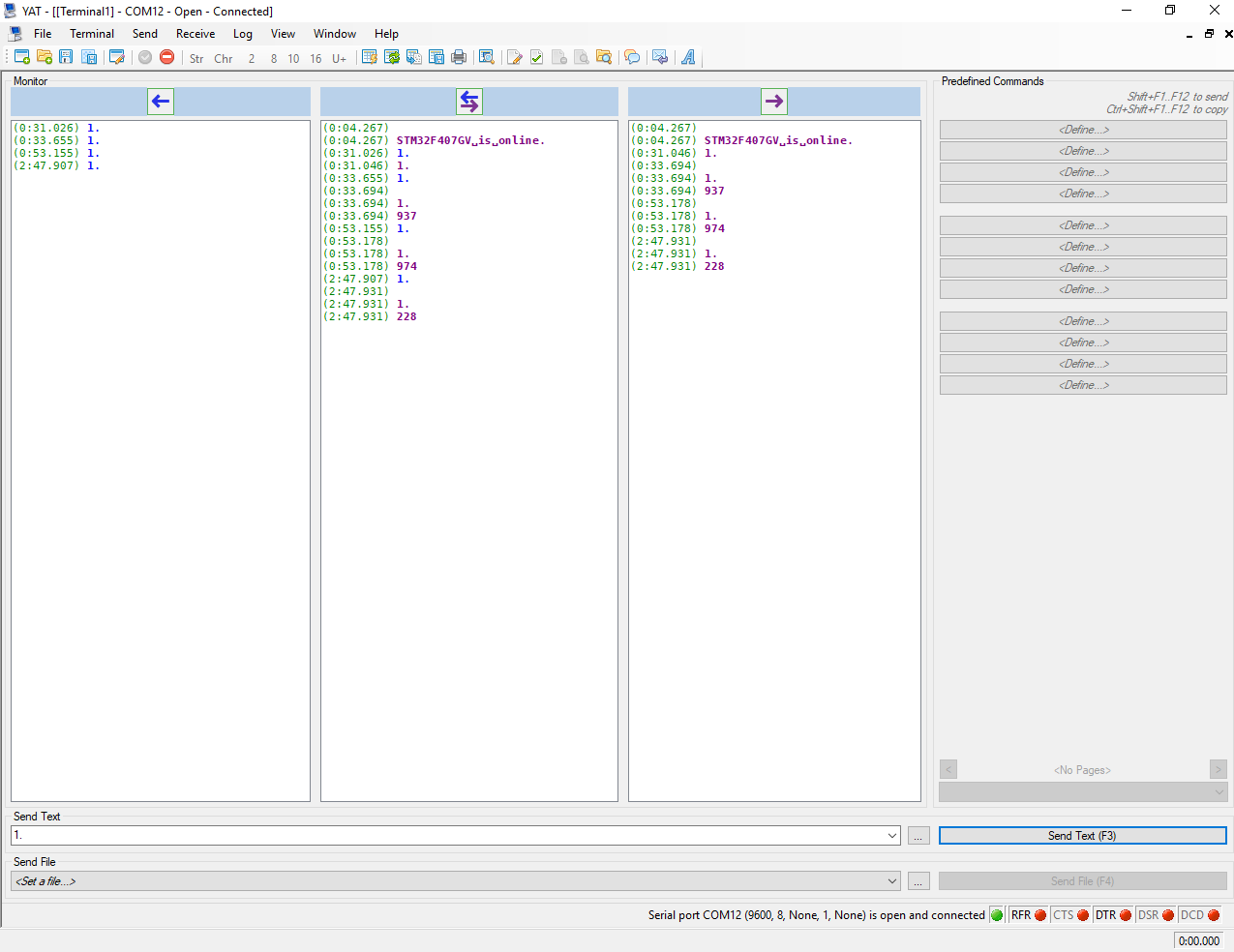


Рисунок 5.11 – Получение информации об освещенности – 2

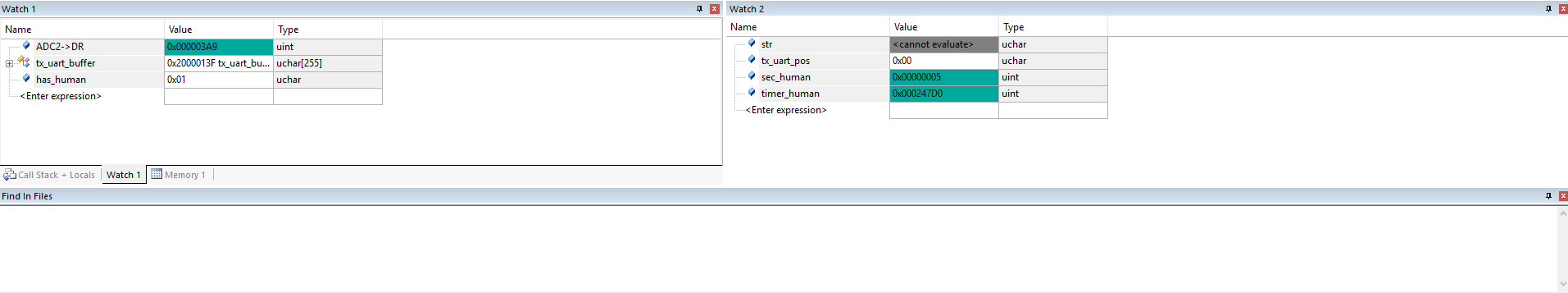


Рисунок 5.12 – Получение информации об присутствии

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.

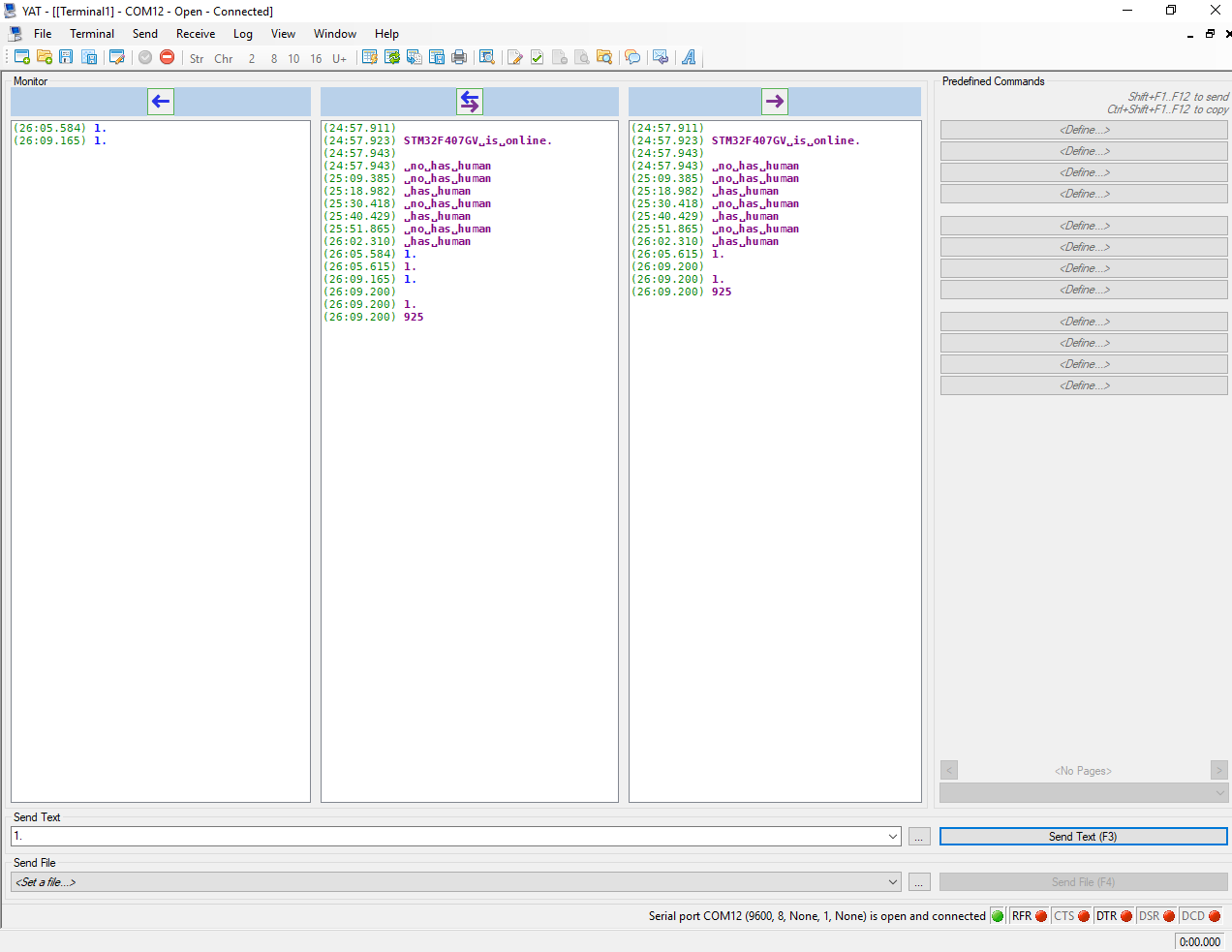


Рисунок 5.13 – Получение информации об присутствии - 2

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.

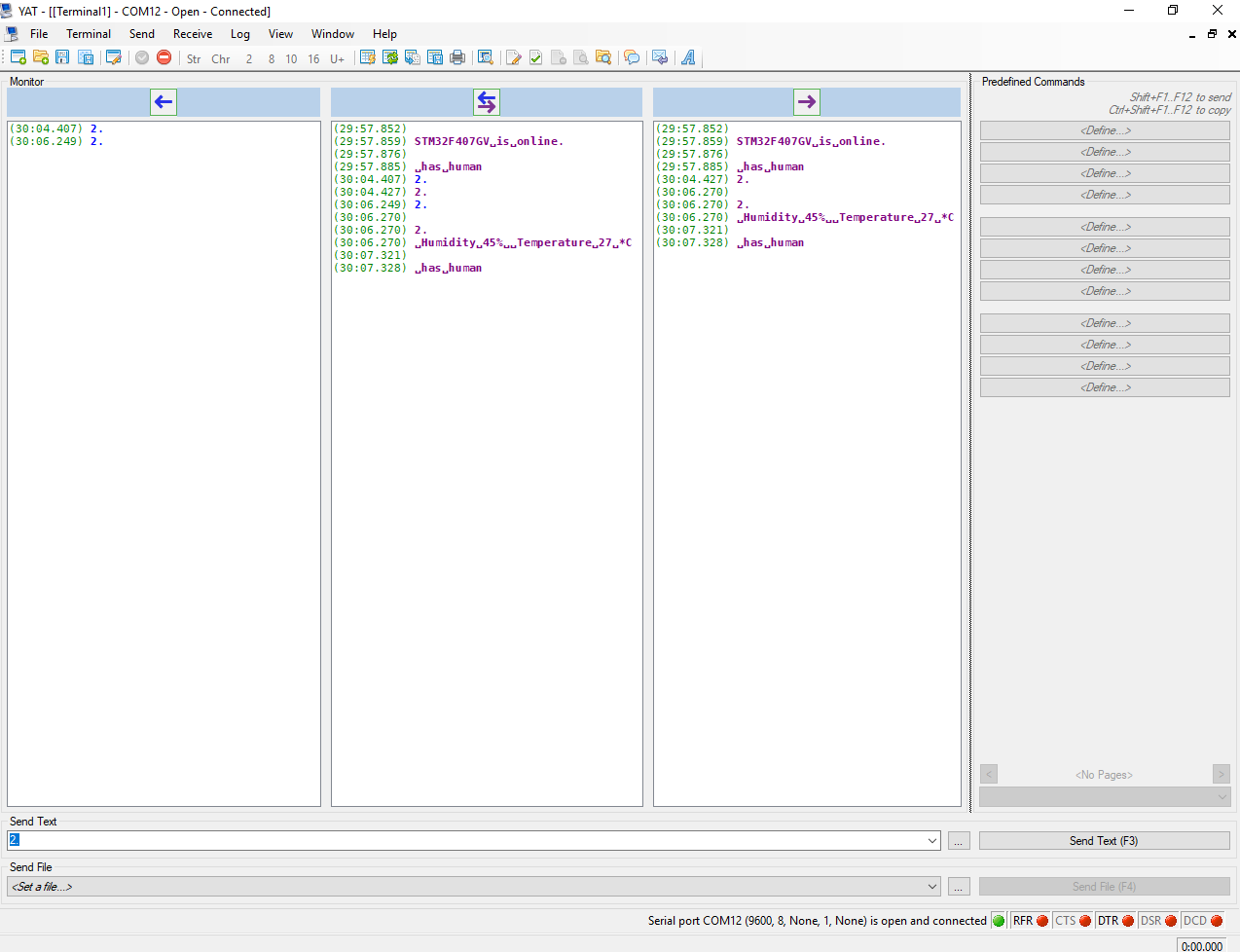


Рисунок 5.14 – Получение информации с датчика влажности и температуры

## 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, Рм |  | | | 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 CO2 | 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 | 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):

где – коэффициент премий ( в нашем Расчете)

n – категории исполнителей, занятых разработкой;

– часовая заработная плата исполнителя i-й категории, р.;

– 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)

где Нд – 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)

где Нсоц – 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 | Табл.6.3 | 210,15 |
| 2. Затраты на разработку программной части | Табл.6.5 | 5649,84 |
| 3. Сумма затрат на производство программно-аппаратного комплекса | Зпр =210,15+5649,84 | 6159,99 |
| 4. Накладные расходы | 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.

, (6.4)

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

– прибыль, включаемая в цену, р.;

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

= 2433,19∙50 (1-0,18) = 99760,79 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 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 Зо | 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:

(6.5)

где – затраты на материалы на единицу изделия, р. (см. таблицу 6.1),

Nп – прогнозируемый годовой объем.

Пм = 17,97 ∙50 = 898,5 р.

2) Определяется годовая потребность в комплектующих изделиях по формуле:

(6.6)

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

Пк = 192,18 ∙ 50 = 9609 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:

. (6.7)

=2101,50 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

(6.8)

где ‒ прирост чистой прибыли, руб.;

‒ инвестиции в производство (Исок + Ир), руб.

Ри = (99760,79 /63017,70) ∙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 -](https://beltelecom.by/private/smart-home%20-) 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](https://components101.com/hcsr501pir-sensor) Дата доступа: 13.04.2020.

[10] TCRT5000 [Электронный ресурс]. – Электронные данные. – Режим доступа: [https://pdf1.alldatasheet.com/datasheet‑pdf/view/252411/VISHAY/TCRT5000.hеml](https://pdf1.alldatasheet.com/datasheet-pdf/view/252411/VISHAY/TCRT5000.html) – 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](https://training.ti.com/tiprecisionlabsisolationwhatgalvanicisolation?utm_source=google&utm_medium=cpc&utm_campaign=ascintisogalvanic_isolation-cpc-tr-google-wwe&utm_content=galvanic_isolation&ds_k=galvanic+isolation&DCM=yescontext=1139747113501511392691135013&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 СO2 [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