Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Физико-технический факультет (ФТФ)
Кафедра прикладной газовой динамики и горения (ПГДиГ)

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
д-р физ.-мат. наук, профессор
_____ Г.Р. Шрагер
подпись
«16» мая 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА
(МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

МОДУЛЬ УПРАВЛЕНИЯ ЦИФРОВОЙ ТЕПЛИЦЕЙ

по направлению подготовки 15.04.06 - Мехатроника и робототехника, профиль “Моделирование робототехнических систем”.

Павлов Денис Александрович

Руководитель ВКР
д-р физ.-мат. наук, доцент
_____ Е. И. Борзенко
подпись
«19» мая 2023 г.

Автор работы
студент группы № 102113
_____ Д.А. Павлов
подпись
«19» мая 2023 г.

Томск – 2023

Министерство науки и высшего образования Российской Федерации.

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Физико-технический факультет (ФТФ)
Кафедра прикладной газовой динамики и горения (ПГДиГ)

УТВЕРЖДАЮ
Руководитель ООП
д-р физ.-мат. наук, профессор
_____ Г.Р. Шрагер
подпись
«20» сентября 2021 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы бакалавра / специалиста /магистра обучающемуся

Павлову Денису Александровичу
Фамилия Имя Отчество обучающегося

по направлению подготовки Код Наименование направления подготовки, направленность (профиль) «Наименование образовательной программы»

1 Тема выпускной квалификационной работы
Модуль управления цифровой теплицей

2 Срок сдачи обучающимся выполненной выпускной квалификационной работы:
а) в учебный офис / деканат – 19.05.2023 б) в ГЭК – 19.05.2023

3 Исходные данные к работе:
Объект исследования – Цифровая теплица
Предмет исследования – Система автоматизированного управления климатом в теплице.
Цель исследования – Разработка программно-аппаратного комплекса модуля управления цифровой теплицей.

Задачи:
Разработка материнской платы модуля управления цифровой теплицей, сборка и тестирование материнской платы, написание модуля протокола Modbus, написание управляющего программного обеспечения (ПО), тестирование и отладка ПО.

Методы исследования:
Метод анализа литературных данных и метод моделирования при разработке модуля

Организация или отрасль, по тематике которой выполняется работа, –
Сельское хозяйство

4 Краткое содержание работы
В работе разрабатываются аппаратная и программная части модуля управления цифровой теплицей.

Руководитель выпускной квалификационной работы
др физ.-мат. наук, профессор ФТФ ТГУ / Е.И. Борзенко
должность, место работы *подпись* *И.О. Фамилия*

Задание принял к исполнению
Студ. ФТФ ТГУ / Д.А. Павлов
должность, место работы *подпись* *И.О. Фамилия*

Аннотация

В выпускной квалификационной работе содержится 36 страниц, 22 рисунка, 18 источников литературы и одно приложение.

В работе рассматривается разработка модуля управления цифровой теплицей, которая содержит: основной управляющий микроконтроллер семейства STM32F7, интерфейсы для подключения внешних датчиков, интерфейсы подключения инкрементного энкодера, TFT-дисплея, USB, microSD, блоки дискретных входов, выходов, аналоговых входов, силовых выходов. Также рассматривается разработка программных модулей для работы с дискретными входами, выходами, связью с ПК по интерфейсу RS-485 с использованием протокола Modbus.

Были изучены следующие темы:

1. Написание управляющей программы для микроконтроллера STM32;
2. Протокол Modbus;
3. Проектирование печатных плат;
4. Разработка программ с использованием объектно-ориентированного подхода.

Результатом выпускной квалификационной работы является собранный прототип платы управления, на которой были проверены разрабатываемые в ходе данной работы модули.

Оглавление

Перечень условных обозначений, символов, сокращений, терминов

МК – микроконтроллер,

ARM – Advanced RISC Machine,

TFT – Thin-Film Transistor,

SPI – Serial peripheral interface,

I2C – Inter-Integrated Circuit,

ШИМ – Широтно Импульсная Модуляция,

USB – Universal Serial Bus,

USART – Universal Synchronous/Asynchronous Receiver/Transmitter,

ПЛК – Программируемый Логический Контроллер,

ОЗУ – Оперативное Запоминающее Устройство,

TVS – Transient Voltage Suppression,

ПК – Персональный Компьютер,

ПО – Программное Обеспечение,

CRC – Cyclic Redundancy Check,

DI – Discrete In,

DO – Discrete Out.

Введение

Система автоматизированной теплицы представляет собой набор датчиков различного назначения, исполнительных механизмов и основного блока, управляющего всей системой. Подобные системы активно внедряются в промышленность, так как они позволяют многократно повысить скорость реакции на такие события, как повышение температуры выше критической, недостаточность освещенности и другие. Все вышеперечисленное позволяет повысить урожайность в целом. Растущие требования к качеству и количеству сельскохозяйственной продукции, а также необходимость оптимизации трудозатрат и повышение эффективности производства, делают необходимым автоматизацию процессов сельскохозяйственной отрасли.

При создании устройств автоматизации необходимо поддерживать гибкость, проявляющуюся в способах подключения внешних устройств, а также интерфейсах взаимодействия с ними. Так, если создать систему, которая будет способна перестроиться под нужды исполнителя, то попытка применения такой системы увенчается успехом при расширении ПО.

В настоящее время технология интернета вещей (IoT) все чаще внедряется в сельское хозяйство. Примерами таких систем могут служить: устройства мониторинга жизненных показателей животных, системы мониторинга за ульями [1] и другие. Данная концепция позволяет внести любую «вещь» в интернет и взаимодействовать с ней из любой точки мира (при наличии интернета). В случае с автоматизированной теплицей примерами взаимодействия являются: наблюдение за такими показателями, как температура, влажность, в том числе и почвы, видеонаблюдение, системы сигнализации и другие.

Совет стран Персидского залива субсидирует создание автоматизированных теплиц. Такое решение было принято вследствие необходимости ограничения потребления водных ресурсов, а также увеличения производства сельскохозяйственной продукции. Ожидается, что в некоторой степени переход от производства в открытом грунте к защищенному сельскому хозяйству позволит сэкономить значительное количество воды, которую затем можно будет использовать для других целей. Этот подход значительно компенсирует ограничения региона, связанные с ограниченностью сельскохозяйственных угодий и нехваткой воды [2].

Увеличение вычислительных мощностей позволяет оперативно собирать огромные объемы данных (большие данные). Алгоритмы и ИИ открывают новые возможности для обработки больших массивов информации и получения новых знаний. При этом передача

данных оптимизирована за счет развития сети, через которые могут транслироваться открытые, стандартизированные форматы файлов. Использование цифровых систем в конечном счете способствует реализации эффективной аграрной политики и организации постоянного мониторинга, который в обычных условиях является очень сложным и требует много времени, особенно в области окружающей среды, климата и сельского хозяйства. Поэтому изучение состояния и перспектив цифровизации аграрной сферы представляет большой научный и практический интерес [3].

Цифровые технологии используются на 10% сельскохозяйственных угодий. Кроме того, одной из наиболее заметных проблем, с которыми сталкивается процесс внедрения цифровых технологий, является нехватка специализированных кадров и импорт иностранных технологи. Вместе с тем, как показывают модельные расчеты, наше сельское хозяйство располагает огромным потенциалом развития, который может быть реализован за счет многих факторов, включая оптимизацию размещения отраслей [4]. Эффект от этого можно существенно усилить за счет масштабного распространения цифровизации [3].

1 Теоретическая часть

Разрабатываемый модуль управление теплицей будет включать управляющий микроконтроллер STM32, управляющий модулями платы, Wi-Fi модуль на базе микроконтроллера ESP8266, обеспечивающий связь с сервером. Модуль будет также включать следующие блоки:

1. Блок дискретных входов для получения сигналов с внешних устройств;
2. Блок дискретных выходов для дискретного управления внешними устройствами;
3. Блок аналоговых входов с опорным напряжением 3,3 В для сбора показаний с аналоговых датчиков;
4. Блок силовых выходов, для управления мощными резистивными и индуктивными нагрузками (водяная помпа, приводы открытия форточек, освещение и другие);
5. Блок цифровых интерфейсов (I2C, SPI, RS-485) для обеспечения управления внешними устройствами или получения команд управления модулем, а также данных с внешних датчиков;
6. Блок отображения визуальной информации об устройстве (TFT-дисплей, светодиоды состояния).

Общая схема устройства представлена на рисунке 1.

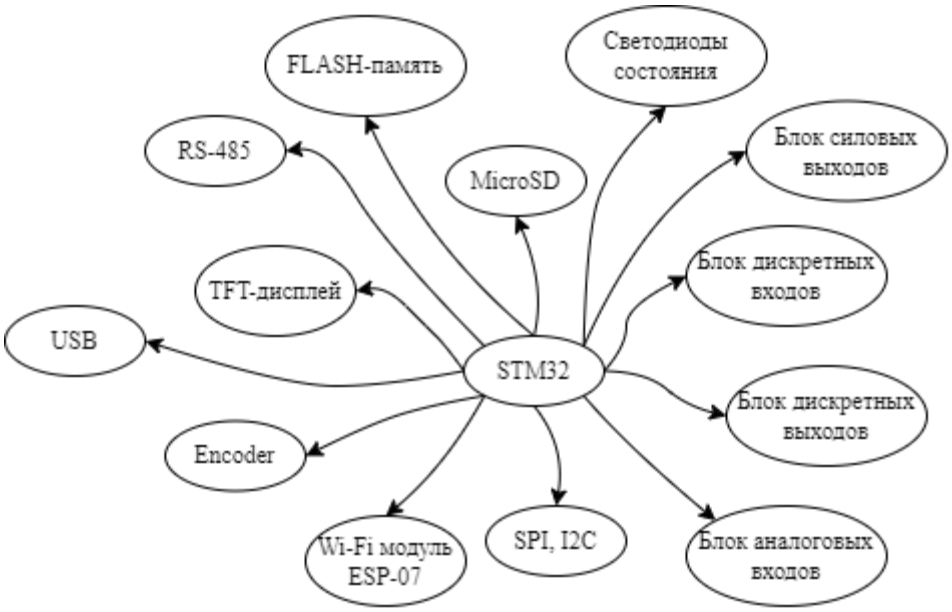


Рисунок 1 – Общая схема устройства

Алгоритм управления будет реализован с использованием объектно-ориентированного метода программирования. Для каждого модуля будут разработаны

соответствующие сущности, обладающими интерфейсами взаимодействия и получения состояния модуля.

Система прерываний будет реализована по необходимости для каждого модуля. Также будут инициированы периодические прерывания таймеров с различным приоритетом, для выполнения необходимых инструкций по обработке состояния аппаратных и программных сущностей (данный подход описан в главе 2.3.1).

1.1 Микроконтроллер STM32F7 серии

В качестве блока управления был выбран микроконтроллер STM32F746ZGT6 в корпусе LQFP-144 (144 ножки). Данный микроконтроллер обладает 1 Мбайт FLASH и 320 кбайт SRAM, имеет большое количество цифровых интерфейсов и основывается на ядре Cortex-m7 [5]. Данное ядро имеет высокую производительность при работе с целочисленными операциями.

1.2 Беспроводная передача данных

Для взаимодействия с сервером необходимо определить способ беспроводной передачи данных. Существует несколько вариантов: подключение по Ethernet, Wi-Fi. Также существуют варианты реализации обменам данными по таким протоколам, как LoRa [6], ZigBee [7], Bluetooth и др. Проблема последних – необходимо промежуточное звено, способное принимать данные по этим протоколам и сохранять их либо в глобальную сеть (тогда необходима передача данных в интернет), либо в локальную сеть (тогда самым оптимальным решением является добавление в систему приемного узла, который будет принимать и обрабатывать данные с основного блока управления, а также на приемном узле будет запущено серверное приложение). В данной работе разрабатывается устройство, взаимодействующее с глобальным сервером, поэтому было принято решение передавать данные по Wi-Fi.

1.3 Подключаемые устройства

Для отслеживания состояния теплицы, необходимо использовать большое количество датчиков, как аналоговых, так и цифровых. Для формирования правильной системы управления, минимальными показателями, за которыми необходим мониторинг, являются: температура, влажность, влажность почвы, освещенность. Блок управления должен быть способен считывать данные с каждого датчика, по которым формировать сигналы управления устройств взаимодействия с теплицей. Примерами таких устройств могут являться: помпа для полива, форточка для проветривания (возможно с активным

проветриванием), лампа. В конечном устройстве необходимо предусмотреть подключение различных датчиков и модулей по различным интерфейсам. Рассмотрим отслеживаемые показатели, информацию о которых передают популярные микросхемы по соответствующим интерфейсам:

1. Температура. В случае с температурой существует большое количество различных датчиков, таких как: термопара (аналоговый вход) и другие аналоговые датчики температуры, цифровой датчик DS18B20 [8] (1-Wire), BMP180 [9] (SPI или I2C).
2. Влажность воздуха и почвы. Для отслеживания данных показателей используются аналоговые датчики.
3. Освещенность. Для отслеживания можно использовать фоторезистор (требуется аналоговый вход).

Соответственно, необходимо поддержка подключения датчиков по протоколам I2C, SPI, 1-Wire, а также аналоговых датчиков.

Для внесения изменений в показатели теплицы (например, влажность почвы), используются устройства, основанные на электроприводах (в нашем случае постоянного тока). Соответственно для них необходимо предусмотреть способ подключения мощной нагрузки, управляемой с помощью ШИМ.

Для взаимодействия «на месте», необходимы устройства ввода вывода. Универсальным выбором будут следующие варианты: использование дискретных входов и выходов. Также для визуального контроля необходим дисплей (в нашем случае TFT), а также светодиоды, по которым можно будет отследить состояние основного блока.

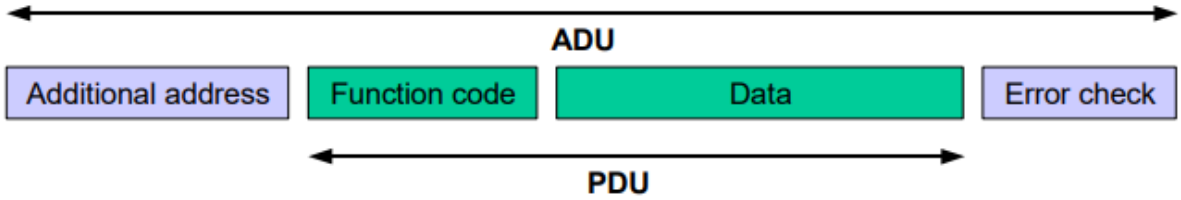
1.4 Протокол Modbus RTU

Передача данных по физическому интерфейсу без использования надстроек в виде протоколов может вызвать проблемы потери данных, например, при помехах. При этом использование протоколов передачи данных позволяет кратно повысить надежность систем.

В промышленности, одним из самых распространенных протоколов связи является Modbus [10]. Этот протокол может использоваться для связи между устройствами различных производителей, что делает его очень универсальным и широко применяемым в промышленной автоматизации. Так, большинство ПЛК, устройств управления и преобразователей частоты имеет возможность общения по данному протоколу. Поддержка протокола Modbus позволяет устройству гибко «вклиниваться» в промышленные предприятия.

1.4.1 Структура пакета

Modbus является полудуплексным протоколам, основанном на принципе «вопрос-ответ». Устройства обмениваются пакетами данных (рис 2).



Здесь Additional address – адрес устройства Modbus, Function code – функциональный код (команда), Data – данные, соответствующие функциональному коду, Error check – CRC пакета.

Рисунок 2 – Содержание пакета Modbus

Запрос инициализирует Master, отправляя соответствующий пакет данных. Slave должен обработать полученные данные и сформировать ответ, который соответствует структуре пакета Modbus.

Каждое устройство обладает своим уникальным адресом (максимальный адрес – 255). Master при этом должен обращаться к конкретному устройству, указывая его адрес.

1.4.2 Задержки между пакетами

Представим ситуацию, что Master отправил запрос и ждет ответа. Каким образом Slave должен понимать, что пакет закончен, если в структуре пакета отсутствуют поля синхронизации? В случае, если Slave не ответил, как Master должен понять, что возникла проблема? Для этого существуют задержки между пакетами и таймауты ожидания.

Для решения данной проблемы используются задержки между пакетами, которые определяются бодрейтом физического интерфейса. Вводятся 2 задержки: $t_{1.5}$ и $t_{3.5}$ [11]. Эти задержки определяются следующим образом: если бодрейт больше 19200, то $t_{1.5} = 750$ мкс и $t_{3.5} = 1750$ мкс. Если бодрейт меньше, то $t_{1.5}$ равен времени, соответствующему времени отправки 1,5 байта физического интерфейса, для $t_{3.5}$ – 3,5 времени отправки.

Задержки применяются следующим образом: $t_{1.5}$ это максимальная задержка между байтами, отправляемыми по физическому интерфейсу. Таким образом конец пакета определяется по истечению данного времени. Если по истечению данного промежутка времени возникнет еще одна посылка, это означает, что возникла ошибка (рисунок 3)

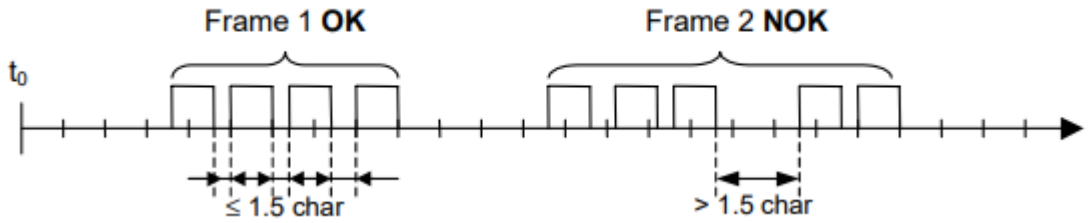


Рисунок 3 – Пример задержки $t_{1.5}$

Задержка $t_{3.5}$ используется как флаг окончания пакета. Таким образом устройства понимают, что пакет завершился и можно перейти к его обработке. Пример задержки $t_{3.5}$ представлен на рисунке 4.

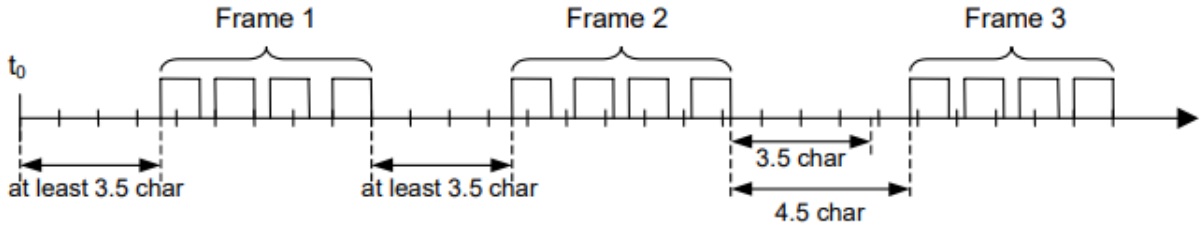


Рисунок 4 – Пример задержки $t_{3.5}$

1.4.3 Команды протокола Modbus

Для взаимодействия с устройством существуют различные команды: чтение/запись «катушек», чтение дискретных входов, чтение/запись регистров устройства и др. Все команды, определенные спецификацией протокола изображены на рисунке 5.

				Function Codes			
				code	*Sub code*	*(hex)*	*Section*
Data Access	**Bit access**	Physical Discrete Inputs	Read Discrete Inputs	**02**		**02**	6.2
		Internal Bits Or Physical coils	Read Coils	**01**		**01**	6.1
			Write Single Coil	**05**		**05**	6.5
			Write Multiple Coils	**15**		**0F**	6.11
	16 bits access	Physical Input Registers	Read Input Register	**04**		**04**	6.4
		Internal Registers Or Physical Output Registers	Read Holding Registers	**03**		**03**	6.3
			Write Single Register	**06**		**06**	6.6
			Write Multiple Registers	**16**		**10**	6.12
			Read/Write Multiple Registers	**23**		**17**	6.17
			Mask Write Register	**22**		**16**	6.16
			Read FIFO queue	**24**		**18**	6.18
	File record access		Read File record	**20**		**14**	6.14
			Write File record	**21**		**15**	6.15
Diagnostics			Read Exception status	**07**		**07**	6.7
			Diagnostic	**08**	**00-18,20**	**08**	6.8
			Get Com event counter	**11**		**OB**	6.9
			Get Com Event Log	**12**		**0C**	6.10
			Report Server ID	**17**		**11**	6.13
			Read device Identification	**43**	**14**	**2B**	6.21
Other			Encapsulated Interface Transport	**43**	**13,14**	**2B**	6.19
			CANopen General Reference	**43**	**13**	**2B**	6.20

Рисунок 5 – Команды Modbus

Пример фреймов запроса и ответа на команду 03 представлен на рисунке 6

Request

Function code	1 Byte	**0x03**
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 125 (0x7D)

Response

Function code	1 Byte	**0x03**
Byte count	1 Byte	2 x **N***
Register value	**N*** x 2 Bytes	

***N** = Quantity of Registers

Рисунок 6 – Пример запроса и ответа на команду 03

1.4.4 Коды ошибок

При запросе к Slave'у могут возникнуть ошибки, связанные с наличием различных свойств устройства. Например, Slave не может обработать запрос (не имеет в своей реализации поддержку команды) или Master пытается обратиться к несуществующим данным. Рассмотрим 4 кода ошибки: exception code 1, exception code 2, exception code 3, exception code 4. Exception code 1 возникает, если запрашиваемая команда не реализована в Slave. Exception code 2 говорит о недопустимом адресе данных. Exception code 3 говорит о недопустимых данных (например, выход за возможные границы чтения данных).

Exception code 4 определяет внутреннюю ошибку Slave’a при попытке выполнения команды.

Пакет ошибки должен содержать команду, логически сложенную со значением 0x80 и код ошибки. Пример такого пакета для команды 03 представлен на рисунке 7.

Error

Error code	1 Byte	**0x83**
Exception code	1 Byte	01 or 02 or 03 or 04

Рисунок 7 – Пример пакета-ошибки для команды 03

1.5 Интерфейс RS-485

RS-485 - это стандарт последовательного интерфейса, который используется для связи между устройствами через двухпроводную линию передачи данных. Он широко используется в промышленности и автоматизации для связи между различными устройствами, такими как датчики, контроллеры и другие устройства.

Интерфейс RS-485 обеспечивает полнодуплексную передачу данных. Он также позволяет передавать данные на большие расстояния (до 1200 м) и подключать несколько устройств к одной линии.

Преимущества интерфейса RS-485:

- Высокая скорость передачи данных;
- Возможность подключения нескольких устройств к одной линии;
- Дуплексная передача данных;
- Дальность передачи данных;
- Устойчивость к помехам и шумам на линии передачи.

Передача данных идёт по двум линиям, A и B, представляющим собой витую пару (два скрученных провода). Используется принцип дифференциальной передачи одного сигнала. По проводу A идет исходный сигнал, по проводу B противофазный. Когда на одном проводе логическая 1, на другом логический 0 и наоборот. Этим достигается высокая устойчивость к синфазной помехе, действующей на оба провода одинаково. Электромагнитная помеха, проходя через участок линии связи, наводит в каждом проводе одинаковый потенциал, при этом информативная разность потенциалов остается без изменений [12].

2 Практическая часть

2.1 Материнская плата

Материнская плата проектировалась на базе микроконтроллера STM32F7 в корпусе LQPF-144, который управляет периферией, и подключаемыми внешними устройствами. Общая схема материнской платы представлена на рисунке соответствует общей схеме устройства, описанной в главе 1.

2.1.1 Блок питания

Напряжение питания платы составляет 24 В. Данное напряжение подается на понижающий DC-DC преобразователь MP1584 [13], который понижает напряжение до 5В. Для питания микроконтроллера STM32 и датчиков используется линейный стабилизатор AMS1117-3.3, на выходе которого имеется 3.3 В. Так как микроконтроллер ESP8266 при передачи данных по Wi-Fi может иметь высокое пиковое потребление тока, было принято решение разместить отдельный стабилизатор напряжения, питающий данный микроконтроллер. Такое решение позволит избежать проблем с просадками питания при нагрузке, что может вызвать неконтролируемое поведение системы и перезагрузку управляющего микроконтроллера. Принципиальная схема блока питания представлена на рисунке 8.

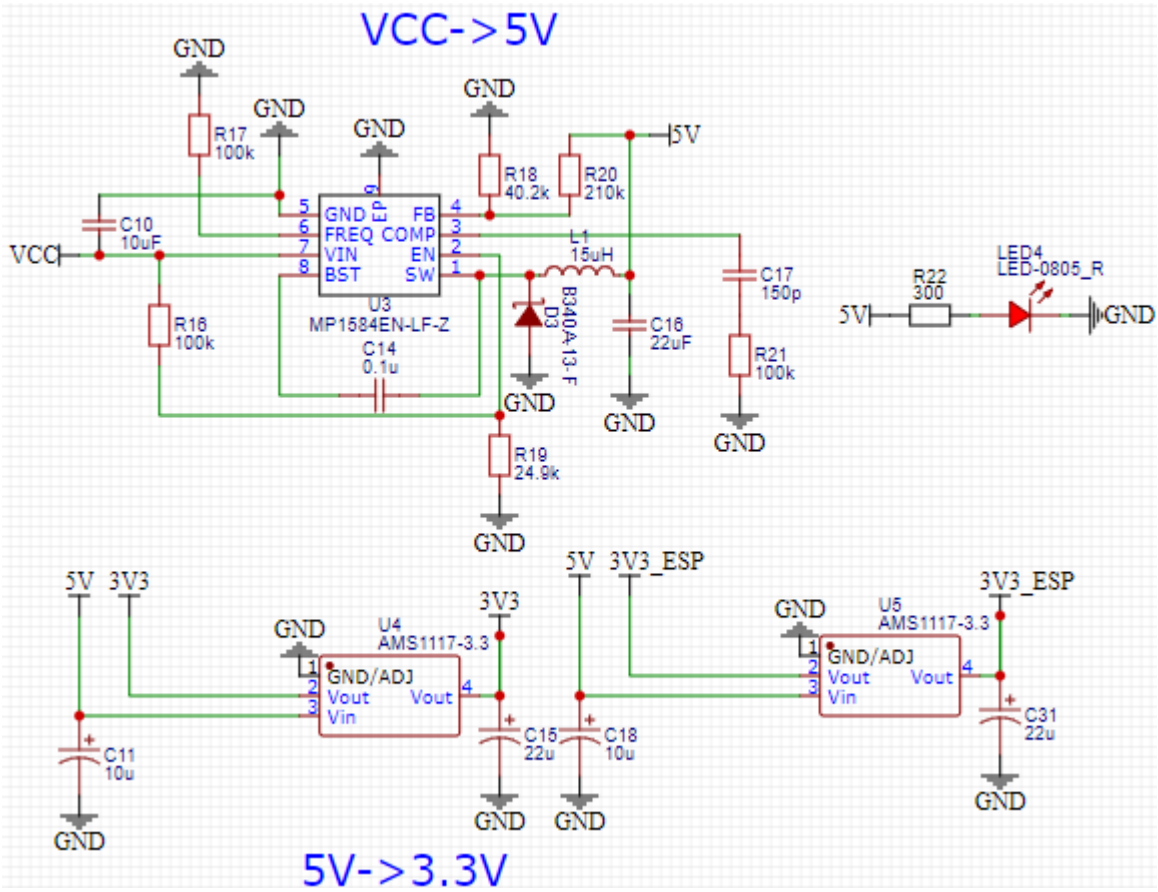


Рисунок 8 – Схема блока питания

2.1.2 Основной блок управления

В качестве управляющего микроконтроллера выбран STM32F7 серии. Данный микроконтроллер обладает высокой вычислительной мощностью (до 462 DMIPS). В первой версии платы была использована внешняя оперативная память, подключаемая по параллельной шине. Такое решение было необходимо, так как планировалось использование камеры, для фиксации фотографий текущего состояния теплицы. Тестирование камеры дало отрицательный результат качества изображения и было принято решение исключить последнюю из устройства. В соответствии с этим пропала необходимость использования внешней ОЗУ. Исключения камеры и оперативной памяти освободило выходы микроконтроллера, к которым были подключены другие устройства ввода/вывода.

Для визуального контроля за работой устройства было принято решение добавление трех светодиодов состояния. Для хранения настроек устройства используется внешняя FLASH память W25Q32JVSSIQ [14], подключаемая по интерфейсу SPI.

2.1.3 Модуль аналоговых входов

Для работы с аналоговыми датчиками был разработан блок аналоговых входов. Аналоговые входы обладают RC-фильтром для фильтрации поступающего сигнала. Эти фильтры вносят небольшую задержку, что в рамках данного устройства не имеют критического значения. Таким образом было спроектировано 12 аналоговых входов по схеме, представленной на рисунке 9.

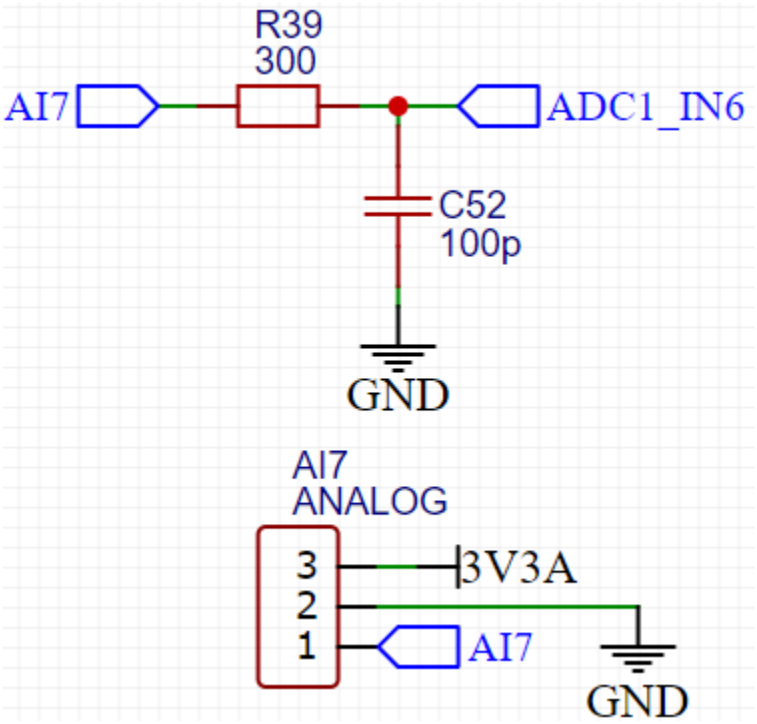


Рисунок 9 – Схема подключения аналоговых входов

Транзистор Q7 (рисунок 10) необходим для коммутации питания аналоговых датчиков. Это необходимо для предотвращения окисления при включенном питании таких датчиков, как, например, датчик влажности почвы.

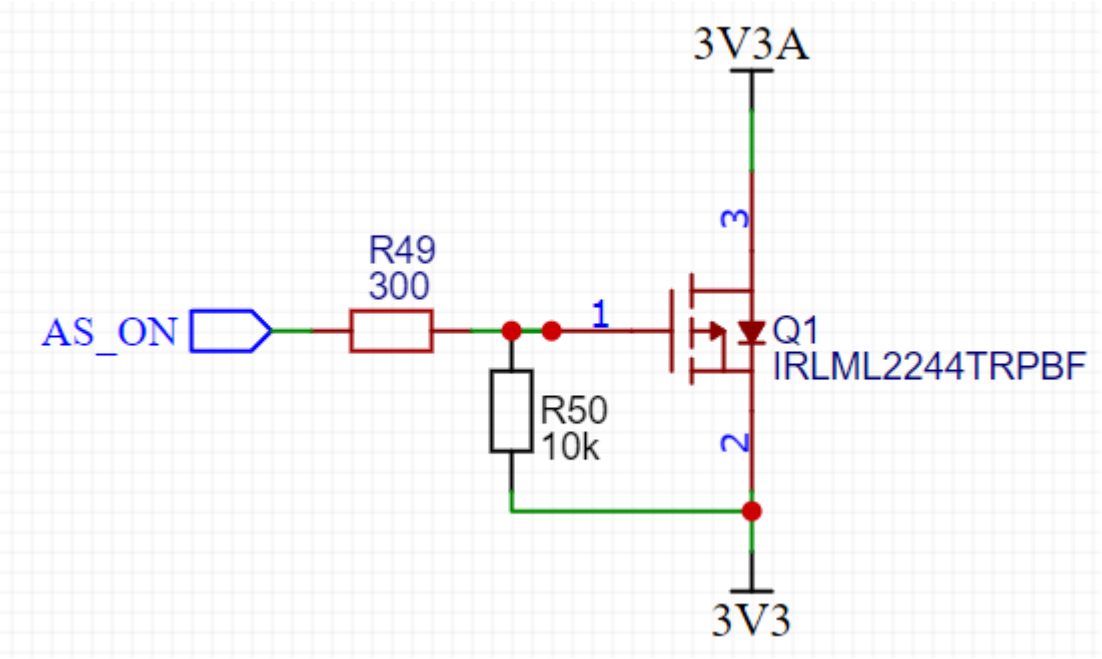


Рисунок 10 – Принципиальная схема коммутации аналоговых входов

2.1.4 Модуль дискретных входов

Дискретные входа необходимы для ввода сигнала с внешних устройств. Универсальным значением напряжения является 24В. Защитным механизмом выступает гальваническая развязка. Принципиальная схема дискретного входа представлена на рисунке 11.

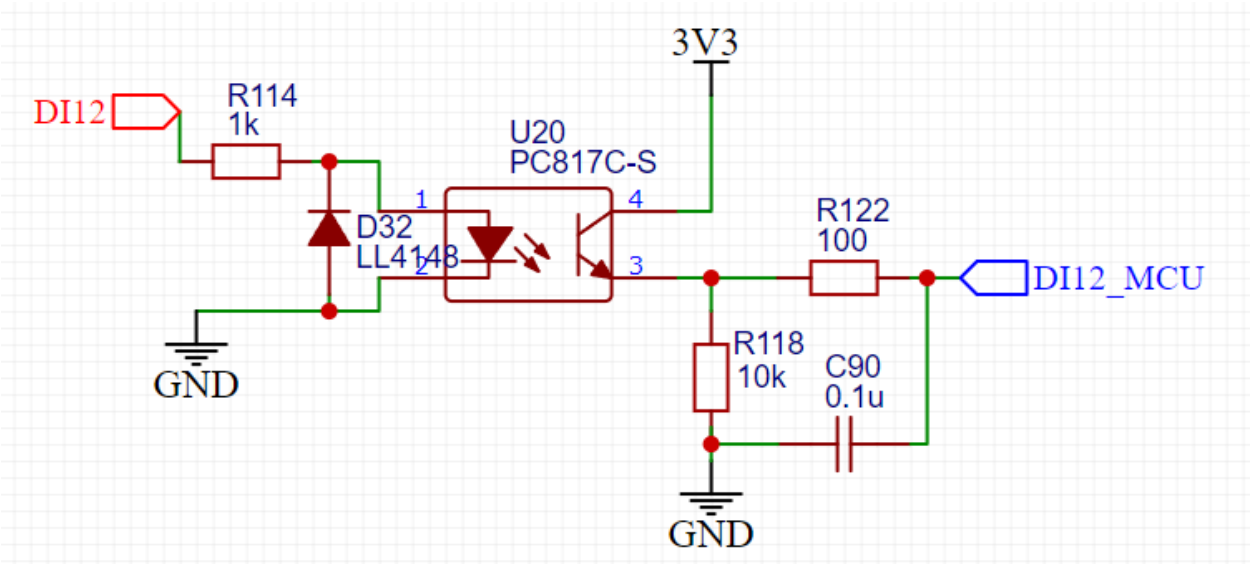


Рисунок 11 – Принципиальная схема дискретных входов

2.1.5 Модуль дискретных выходов

Для передачи сигналов от материнской платы к внешним устройствам используется блок дискретных выходов, напряжение коммутации которых составляет 24В. Также блок дискретных выходов содержит светодиоды индикации,

сигнализирующие о наличие сигнала на дискретном выходе. Принципиальная схема дискретных выходов представлена на рисунке 12. Как видно из схемы, дискретные выхода коммутируют нагрузку на общий провод.

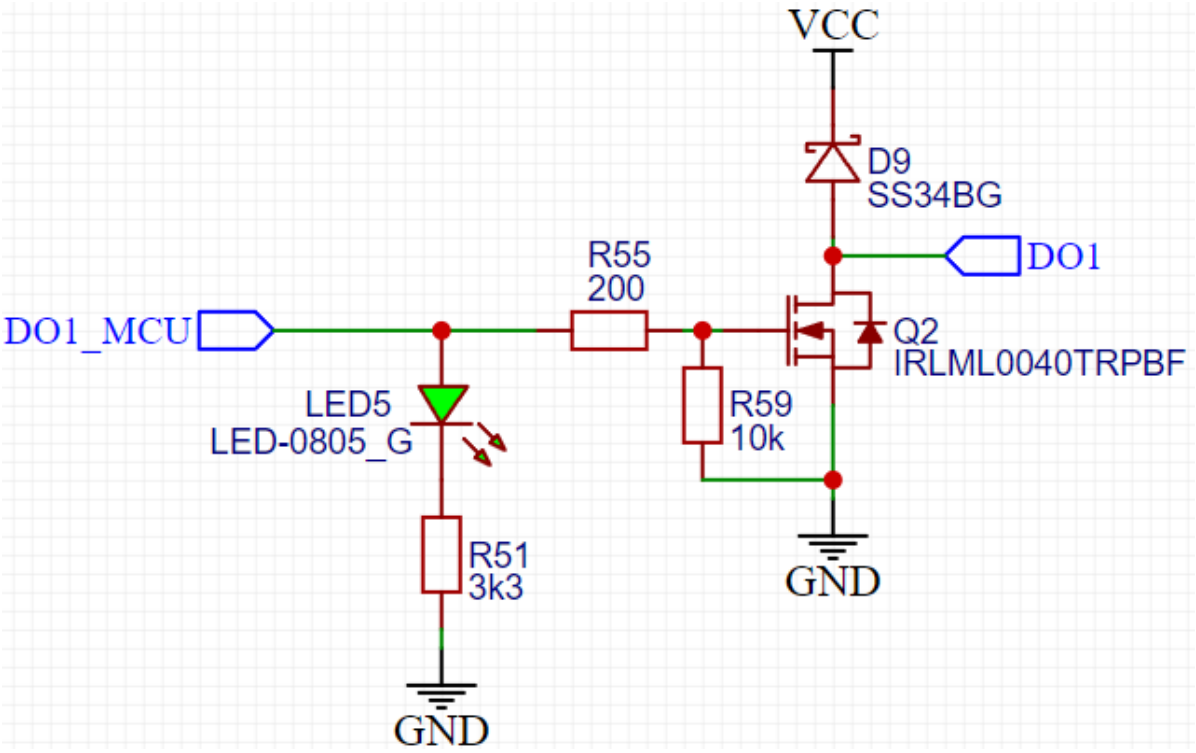


Рисунок 12 – Принципиальная схема дискретных выходов

2.1.6 Подключение внешних датчиков и других устройств

Так как на рынке существует большое количество внешних датчиков с разными интерфейсами взаимодействия было принято решение реализовать поддержку подключения по следующим интерфейсам: I2C, SPI. Также для устройств, подключения к котором требует напряжения питания не больше 5В были реализованы следующие интерфейсы подключения: TFT, подключение инкрементного энкодера, USB Type-C, MicroSD card. Для вышеперечисленных подключений была реализована защита от перенапряжения (может возникать при искровом воздействии) с использованием TVS-диодов (в данной работе используется сборка из 2х диодов NUP2105LT1G [15]. Данная защита была также реализована для подключения программаторов для микроконтроллеров STM32 и ESP8266.

Большая популярность выбранных цифровых интерфейсов добавляет возможность подключение не только датчиков, но и других устройств. Например, на плате отсутствует интерфейс 1-Wire, но спроектировав небольшое устройство, которое конвертирует 1-Wire в SPI мы можем беспрепятственно использовать последнее в системе управления теплицей.

Принципиальная схема SPI и I2C представлена на рисунке 13.

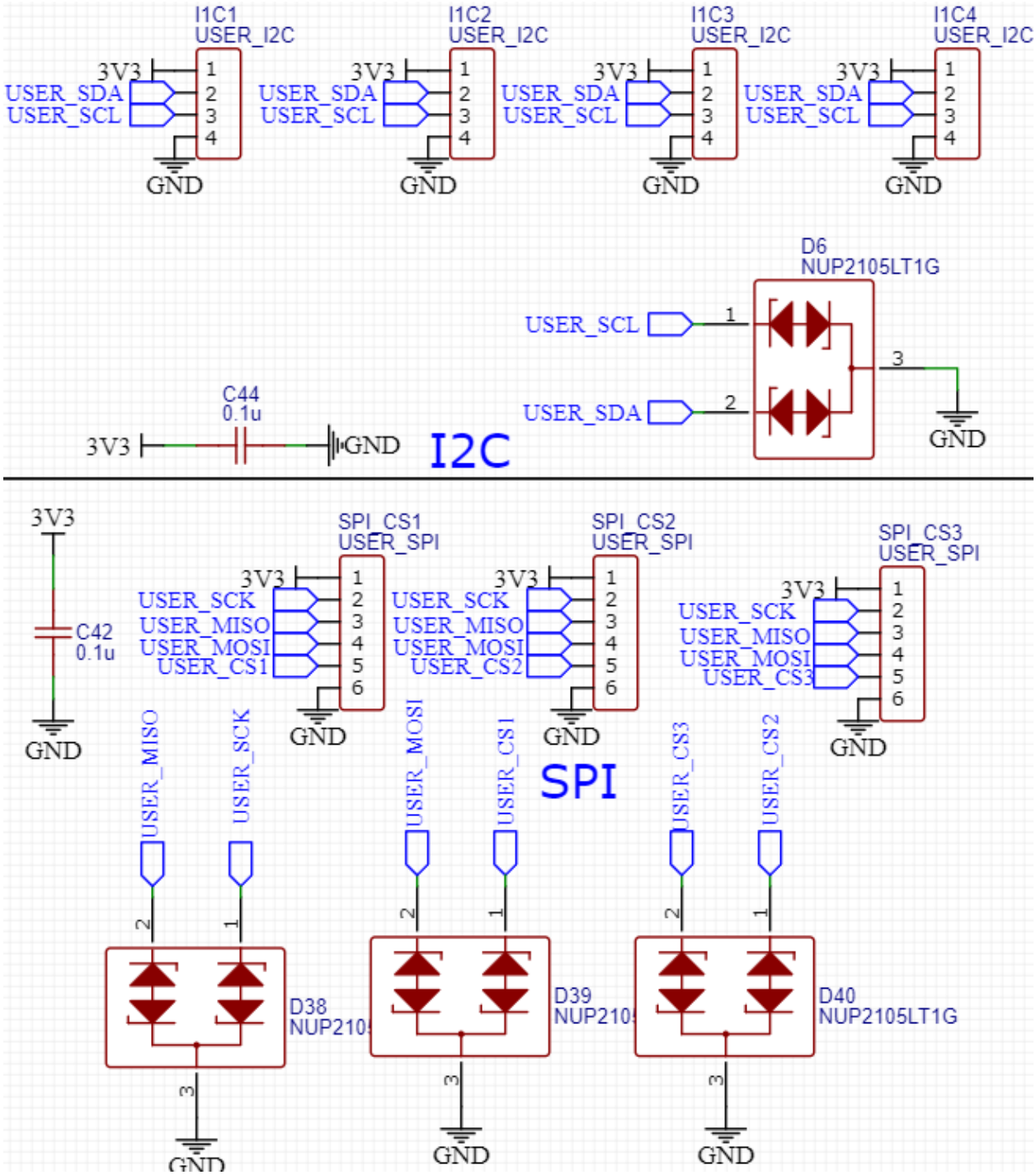


Рисунок 13 – Принципиальные схемы I2C и SPI

2.1.7 Блок силовых выходов

Предусмотрена возможность подключения устройств, питающихся от напряжения питания всей платы (24В), которыми мы можем управлять с помощью ШИМ. Сюда могут подключаться резистивные и индуктивные нагрузки.

Схема подключения реализована на мощных полевых транзисторах и представлена на рисунке 14. Также была реализована обратная связь по току с помощью шунта PA2512FKF7W0R002E [16] и операционного усилителя LMC7101BIM5X [17].

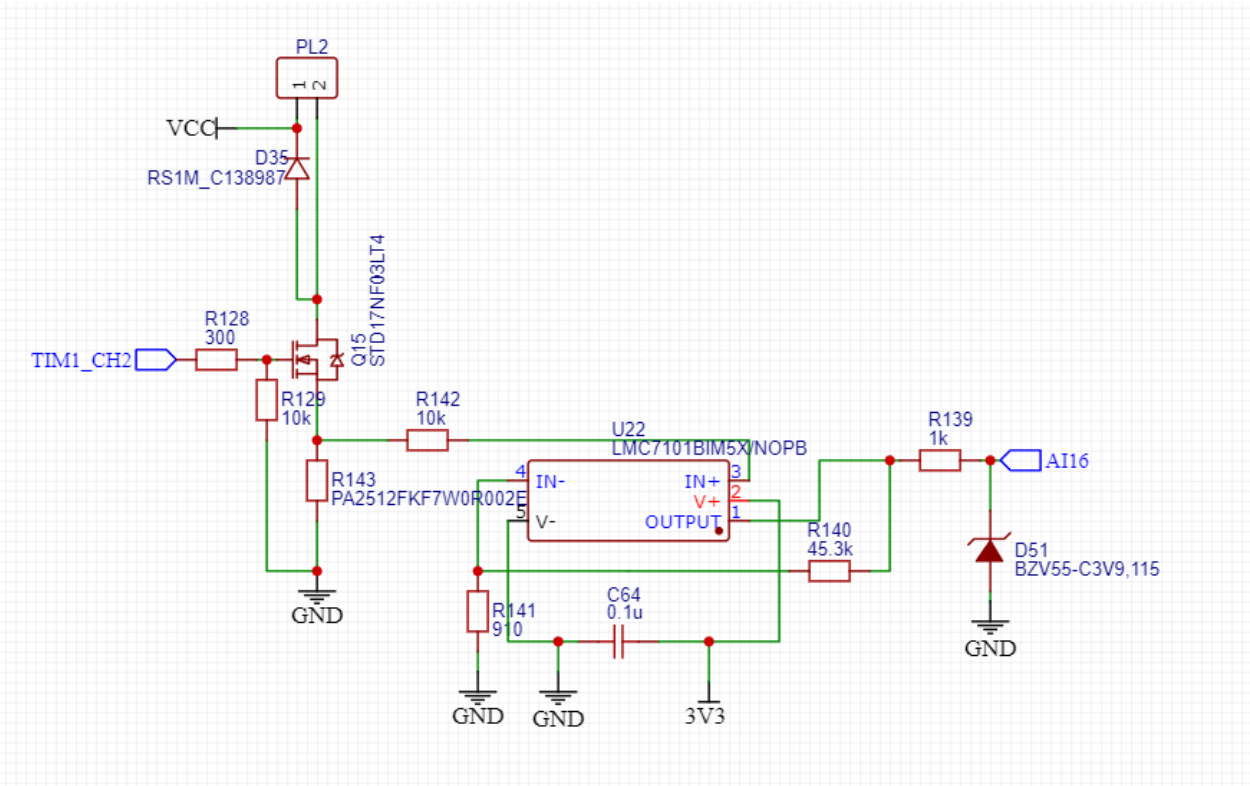


Рисунок 14 – Схема подключения силовых выходов с обратной связью

2.1.8 Микроконтроллер ESP8266

Микроконтроллер ESP8266 [18] подключен к основному микроконтроллеру по интерфейсу UART, и необходим для связи устройства с сервером. Это единственная задача для которой используется данный микроконтроллер. Также был выведен отладочный интерфейс JTAG для программирования и отладки. Схема подключения ESP-07 представлена на рисунке 15.

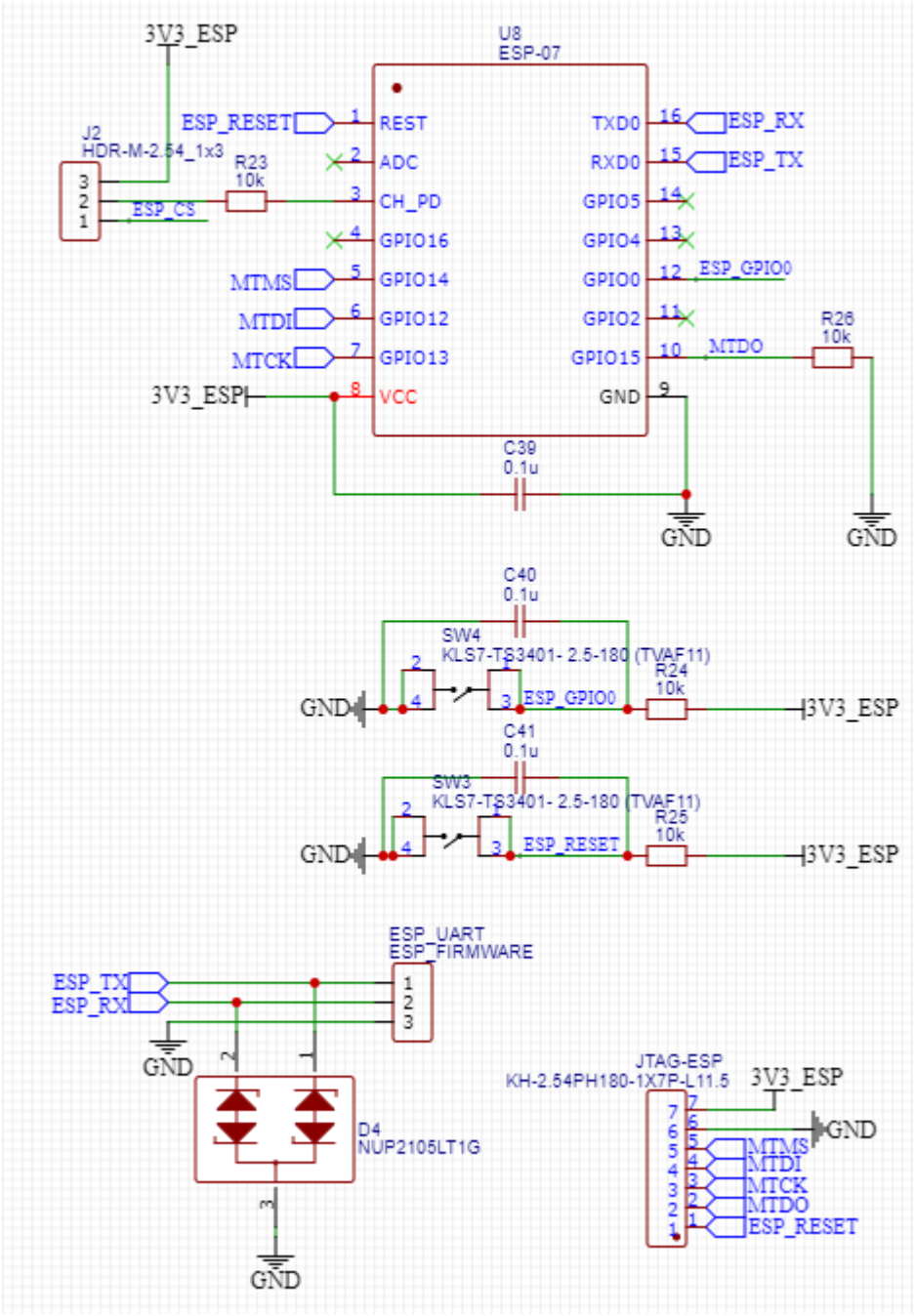


Рисунок 15 – Схема подключение ESP-07

2.1.9 Цифровые протоколы связи

Устройство способно связываться с другими внешними устройствами посредством двух интерфейсов: RS-485 и USB. USB необходим для подключения к ПК и может использоваться для различных нужд: загрузка управляющего ПО в основной микроконтроллер или обмен данными с использование виртуального COM-порта. RS-485 имеет большую распространенность, что способствует интеграции разрабатываемой платы управления в большинство промышленных систем управления.

2.2 Печатная плата

Было разработано 2 итерации материнской платы. Размер первой версии материнской платы составлял 100x100 мм. Первая версия обладала рядом ошибок и недостатков, описанных выше. 2D модель первой версии материнской платы представлена на рисунке 16.

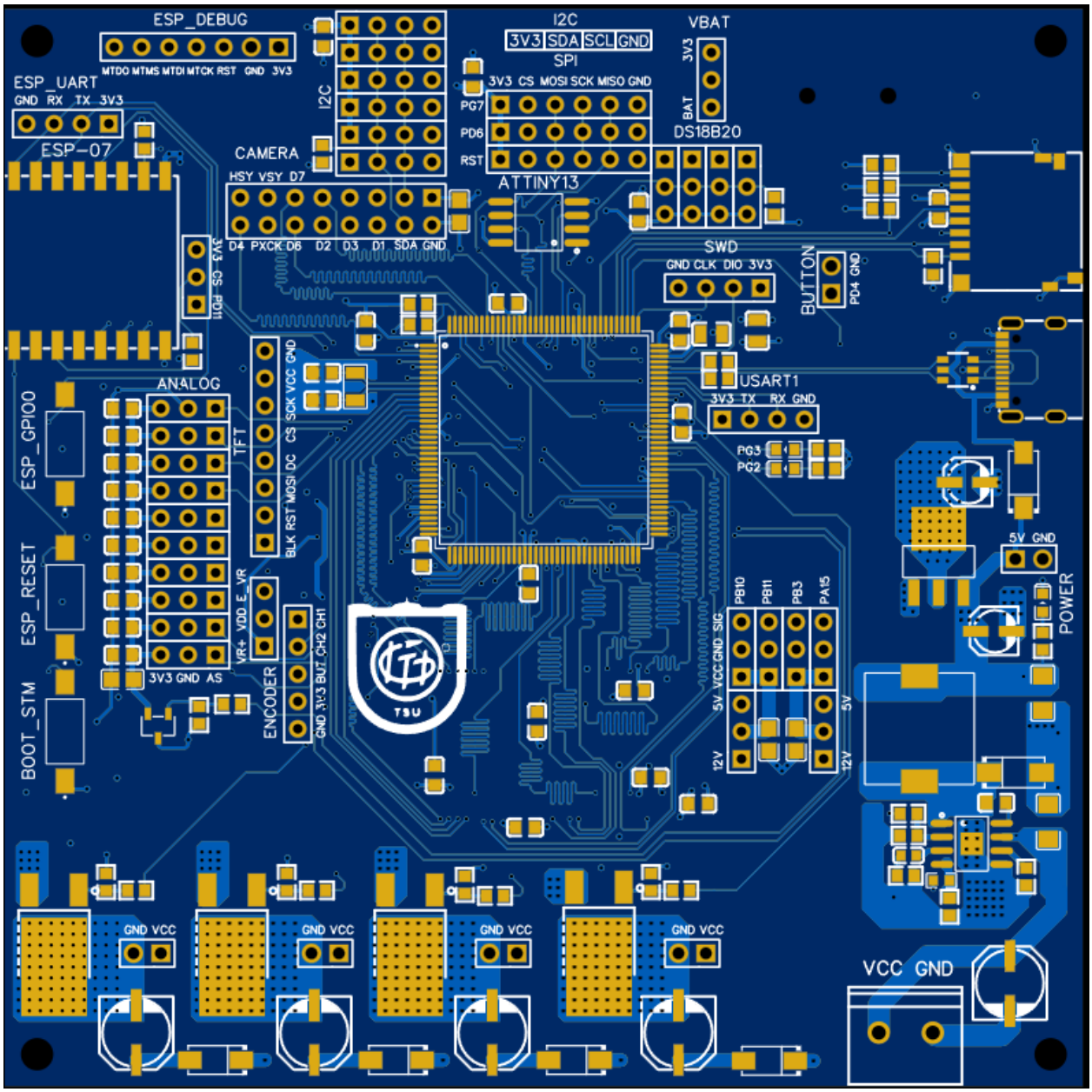


Рисунок 16(а) – Верхняя сторона материнской платы первой версии

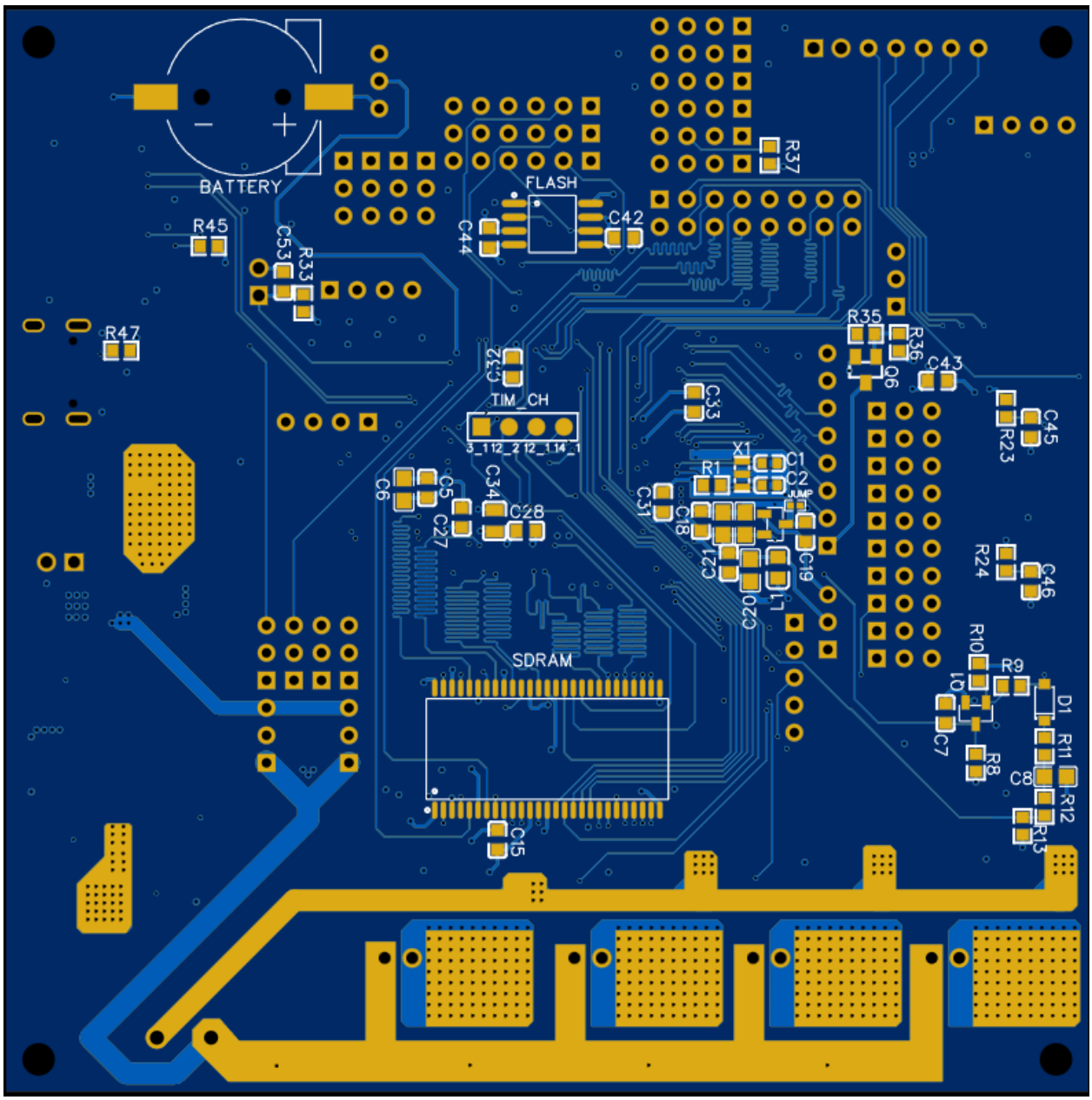


Рисунок 16(б) – Нижняя сторона материнской платы первой версии

Добавление блока дискретных входов и выходов, а также расширение ранее реализованных блоков увеличило размеры платы до 142x142 мм. Все компоненты были размещены на верхней стороне. 2D-модель верхней стороны второй версии материнской платы представлена на рисунке 17.

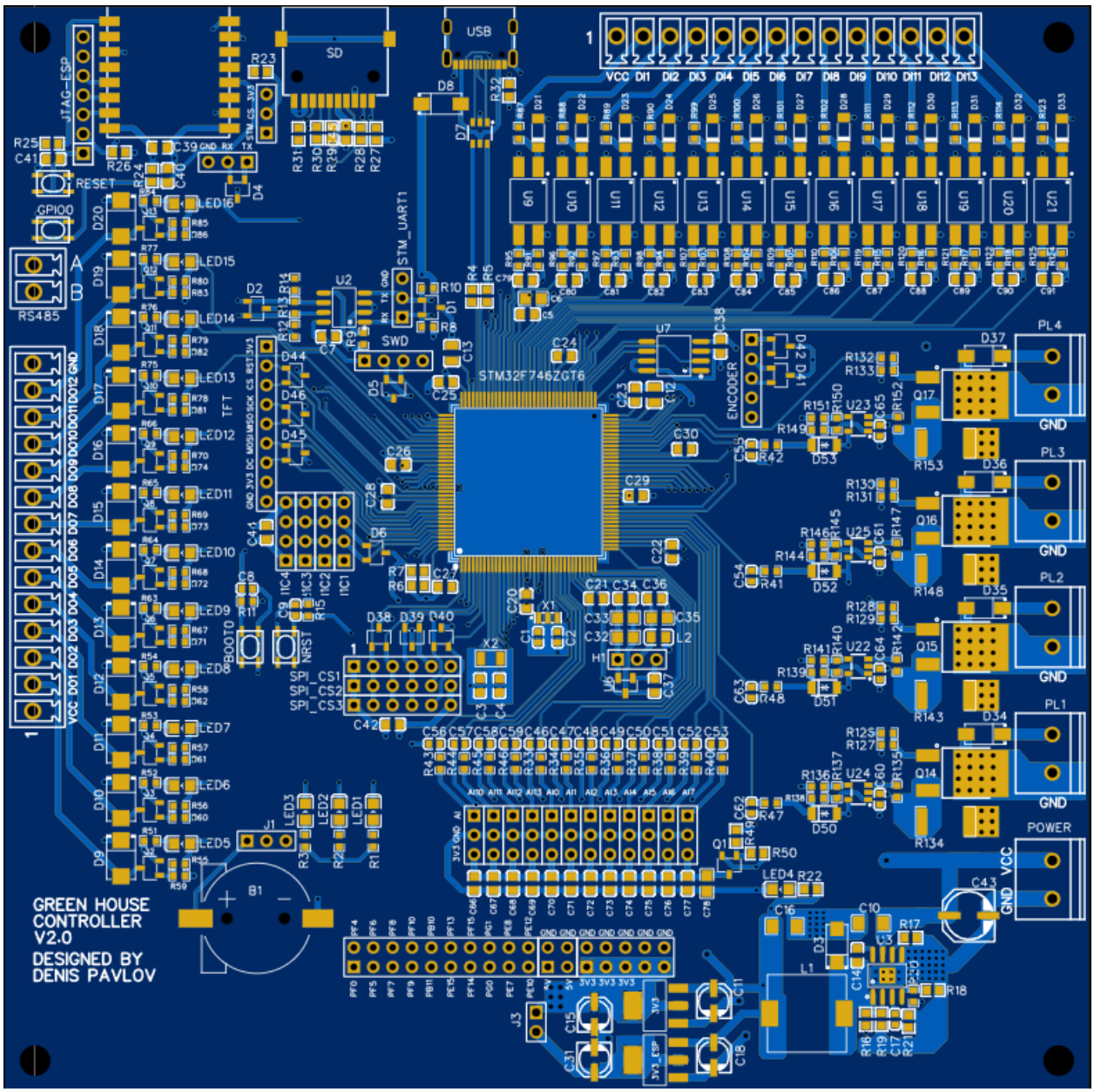


Рисунок 17 – Верхняя сторона материнской платы второй версии

2.3 Управляющее ПО

В рамках магистерской диссертации были разработаны следующие программные модули (далее драйвер): программный таймер, Modbus, CRC, UART, DI, DO.

2.3.1 Общая схема работы модулей

Программные модули (далее драйвер) работают следующим образом: существует функция *Update()*, которая периодически вызывается для внутренней обработки состояния драйвера. Такой поход позволяет декомпозировать работу драйвера на подзадачи, которые занимают мало процессорного времени, что важно при разработке микропроцессорных систем.

2.3.2 Драйвер дискретных входов/выходов

Для начала определим, какими свойствами и методами должны обладать драйверы дискретных входов и выходов. Методы для инициализации низкоуровневой части и метод обновления состояния драйвера. Также для корректной работы и интеграции в другие системы необходимы методы получения ссылки на состояние дискретов и получения их количества. Единственным необходимым свойством является побитовое состояние дискретов (бит 0 – вход/выход 1 и т.д.). Таким образом мы можем сформировать абстрактный класс, представленный в листинге А.1 приложения А. *DiscreteIoUnion* – объединение, позволяющее обращаться как полностью к состоянию дискретов, так и побитово (листинг А.2 приложения А)

Используя абстрактный класс, представленный в листинге 1, мы создадим драйверы дискретных входов/выходов. Для дискретных входов свойство *state_* будет изменяться самим драйвером, а пользователь будет считывать это значение. Для дискретных выходов свойство *state_* изменяет пользователь, а драйвер в соответствие со значением задает сигналы на выходы микроконтроллера.

2.3.3 Modbus

2.3.3.1 Инициализация

Для начала определим самые часто используемые команды Modbus. Исходя из практического опыта, такими командами являются: 0x01, 0x02, 0x03, 0x05, 0x0F, 0x10, 0x11. Соответственно, для корректной работы драйвера в режиме Slave нам необходимо передать в драйвер следующие параметры, представленные в листинге А.3 приложения А.

Также драйвер должен иметь свойство *id*, определяющее текущий идентификатор устройства в рамках протокола Modbus.

Для поддержки таймингов необходимо добавить таймер. В конструктор класса Modbus мы будем передавать ссылку на экземпляр таймера, который наследуется от абстрактного класса *TimerInterface* (листинг А.4 приложения А). Для работы с физическим интерфейсом конструктор принимает ссылку на объект, который наследует абстрактный класс *HardwareInterface* (листинг А.5 приложения А). Также конструктор принимает режим работы драйвера (Master/Slave).

Драйвер Modbus содержит следующие публичные методы.

Метод *RxCallback()*, который помещается в прерывание по приему одного байта с физического интерфейса. Данная функция запускает проверку таймингов и корректности

размера пакета. Обязательным условием является вызов данного метода только после записи принятого байта во внутренний буфер *HardwareInterface.*

Метод *SetTransmitCompleteFlag()* должен вызываться пользователь после окончания отправки данных по физическому интерфейсу. Данный метод сбрасывает состояние драйвера в ожидания новых данных, полученных от Master'a.

Метод *Update()* помещается в прерывание таймера и осуществляет действие в соответствии с текущем состоянием драйвера (будут описаны ниже).

Методы получения текущего состояния драйвера, наличия ошибок, счетчиков ошибочных пакетов. Данные методы необходимы для отладки.

2.3.3.2 Функция Update() и логика работы драйвера в режиме Slave

Драйвер имеет несколько состояний, в зависимости от которых мы производим соответствующие действия. Список состояний реализован в виде перечисления, представленного в листинге А.6 приложения А. Такое разбиение необходимо для уменьшения процессорного времени, занятого драйвером, что позволяет реализовать несколько «параллельно» работающих процессов. Блок схема работы драйвера представлена на рисунке 18.

Рисунок 18 – Блок-схема работы драйвера Modbus в режиме Slave

2.3.3.3 Тайминги

Исходя из спецификации протокола Modbus, нам необходимо реализовать следующие задержки: $t_{1.5}$, $t_{3.5}$, $t_{3.5} - t_{1.5}$. Задержка $t_{1.5}$ свидетельствует о том, что пакет закончился, в коллбеке по прошествию данной задержки необходимо переконфигурировать метод *RxCallback()* вызов которого покажет, что пакет сломан и драйвер перейдет в обработку ошибки. Также необходимо выставить задержку $t_{3.5} - t_{1.5}$, по достижению которой (если не пришли новые данные) начинается обработка данных.

2.3.3.4 Сброс состояния драйвера

Вызов метода *SetTransmitCompleteFlag()* сбрасывает состояние драйвера в *kZeroState*, в котором происходит сброс до значения по-умолчанию необходимых свойств.

2.3.3.5 Парсинг принятого пакета

При вызове функции парсинга данных производятся проверки содержимого пакета: id, команда, параметры команды, CRC. При несоответствии CRC или id, драйвер формирует ошибку. При неправильном запросе данных (например, при количестве дискретных входов, равном 10, запрашивается состояние 11-го дискретного входа) драйвер формирует пакет ошибки.

2.3.3.6 Функция Update() и логика работы драйвера в режиме Master

Драйвер имеет поддержку работы в режиме Master. В данном режиме осуществляется формирование запроса и ожидание принятия ответа. Запрос осуществляется созданием экземпляра соответствующей структуры и вызова метода *Request(void* request, Commands command, uint32_t timeout)*, который принимает ссылку на заполненную структуру запроса, комманду, определяемую в перечислении *Commands* (листинг А.7 приложения А) и таймаут ожидания, который определяется количеством вызовов метода *Update()* и возвращает ошибку при неправильном формировании запроса.

Получение ответа осуществляется методом *ReturnSlaveResponce(void* response)*, который принимает ссылку на структуру, которая будет заполнена в соответствии с запросом. Также данный метод возвращает ошибку (при наличии).

Блок-схема работы драйвера в режиме Master представленна на рисунке 19.

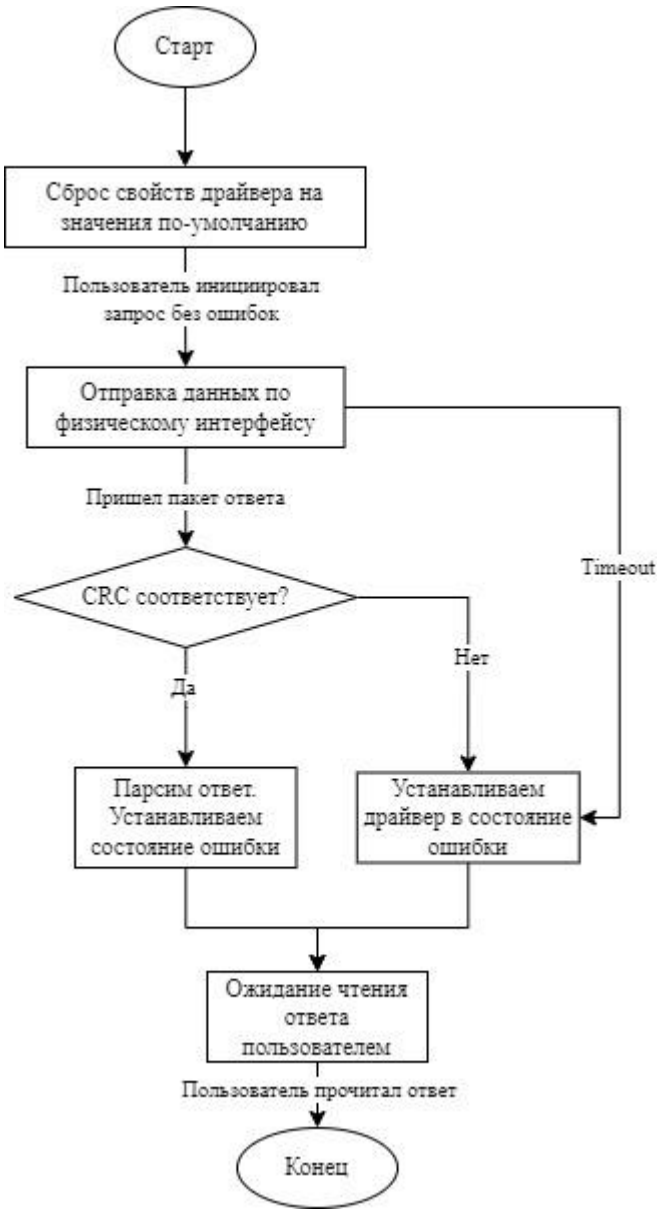


Рисунок 19 – Блок-схема работы драйвера Modbus в режиме Master

2.3.4 UART

В качестве физического интерфейса, который будет реализовывать протокол Modbus, был выбран UART. Для работы с низким уровнем была использована библиотека HAL. Приемопередача осуществляется с использованием прерываний. Библиотека HAL позволяет вызывать *callback* по приему одного байта (в него помещается метод *RxCallback()* драйвера Modbus) и *callback* по окончанию передачи данных (в него помещается метод *SetTransmitCompleteFlag()* драйвера Modbus). Драйвер наследуется от абстрактного класса *HardwareInterface* и реализует необходимый функционал.

2.4 Программный таймер

Для работы Modbus необходимо реализовать драйвер таймера, который наследуется от *TimerInterface*. У нас существует два варианта реализации таймера: программный таймер и аппаратный таймер. В рамках данной работы реализуется программный таймер.

Программный таймер реализован следующим образом: существует метод *Update()*, который вызывается периодически в детерменированные промежутки времени. Программный таймер вызывает функцию-callback (ссылка на данную функцию передается пользователем) с заданной частотой. Блок-схема работы драйвера представлена на рисунке 20.

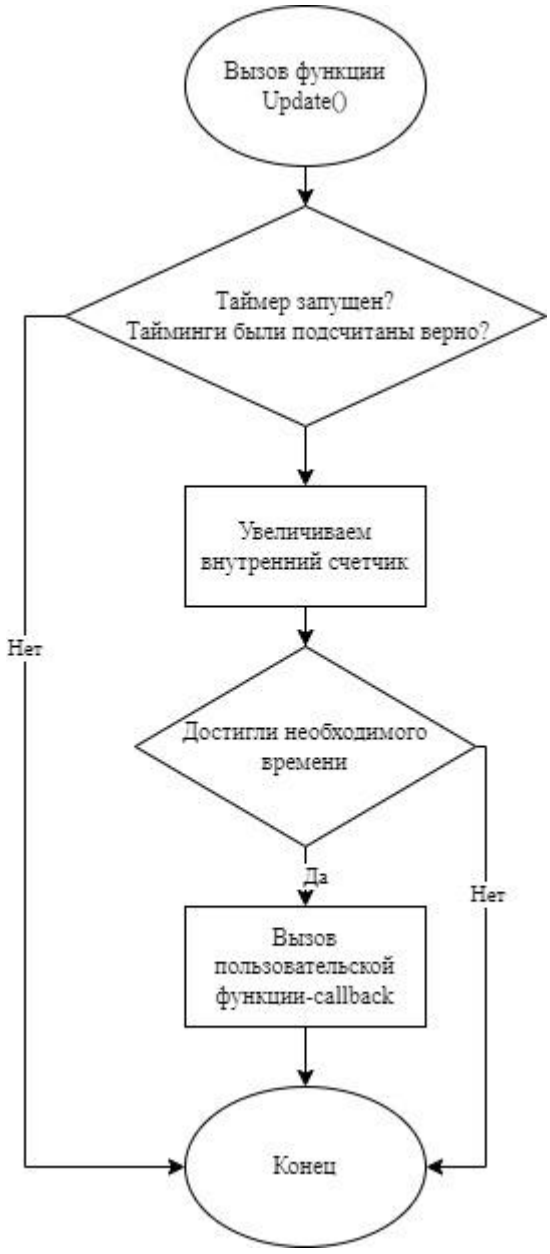


Рисунок 20 – Блок-схема работы программного таймера

2.5 Тестирование

Тестирование проводилось в ручном режиме. Были протестированы все разработанные модули.

Для проверки дискретных выходов вручную выставлялось состояние драйвера, который подавал сигналы на соответствующие выходы микроконтроллера. Установленные в схеме светодиоды позволили производить визуальную отладку. Таким образом все дискретные выходы показали правильную работоспособность в соответствие с ожиданиями.

Для проверки RS-485 был реализован драйвер UART, который проверялся с помощью отладки управляющего ПО. Таким образом приемопередача данных была успешно проверена.

Для проверки Modbus использовалась программа Modbus Poll. Были проверены все описанные ранее команды в режиме Slave. Также с помощью команды 0x01 была реализована проверка совместимости драйверов дискретных выходов и Modbus. Пример пакета пакетов запроса и ответа представлен на рисунке 21. Как видно и рисунка, обработка команды происходит корректно.

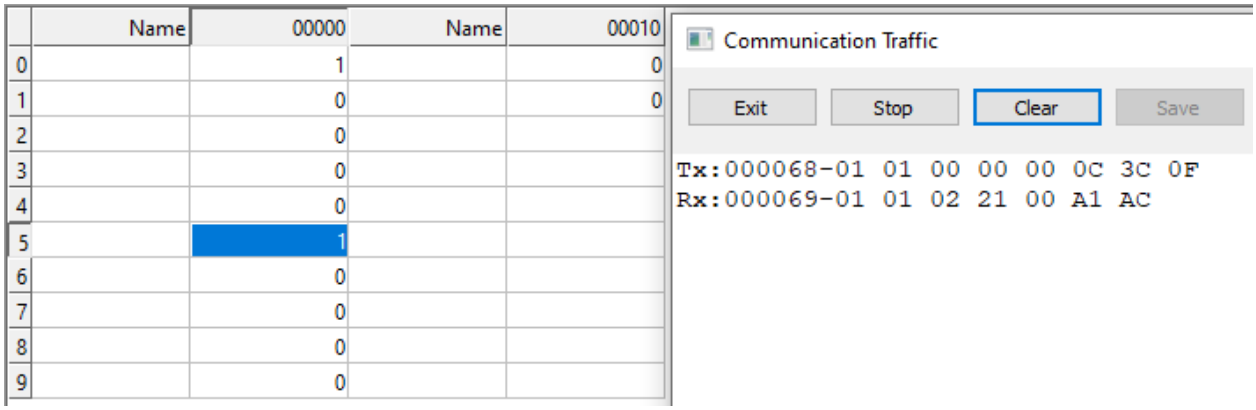


Рисунок 21 – Пример запроса и ответа команды 0x01

Дискретные входы были проверены также с помощью Modbus. На отдельный вход подавался сигнал 24В, который отображался при запросе команды 0x02 (рисунок 22). Таким образом были проверены все входы.

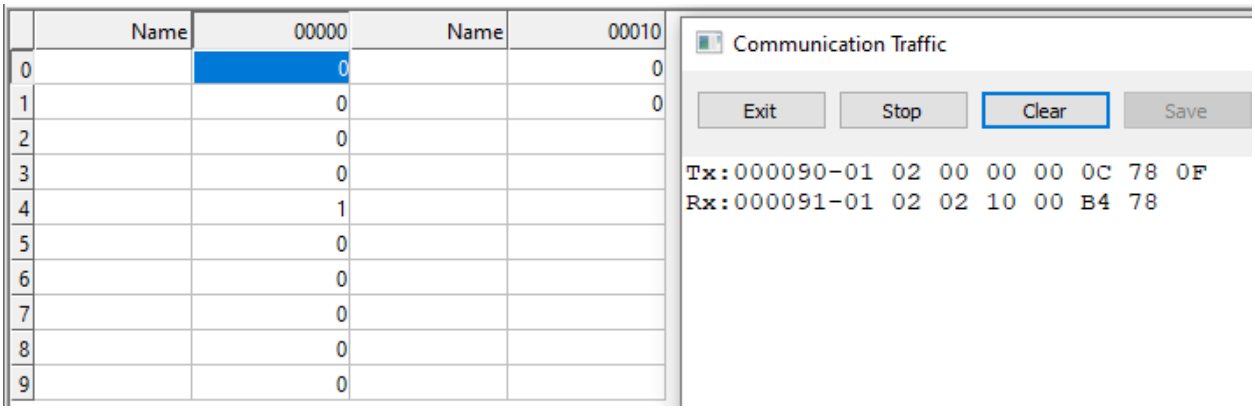


Рисунок 22 – Пример запроса и ответа команды 0x02

В результате был успешно протестирован весь разрабатываемый в рамках данной работы функционал.

Ссылка на репозиторий разрабатываемого ПО: https://github.com/akcel123/Green-House.

Заключение

По результатам работы:

1. Получены навыки программирования МК STM32 на языке «C++» с использованием объектно-ориентированного подхода;
2. Спроектирована и изготовлена печатная плата;
3. Разработана и отлажена библиотека Modbus;
4. Разработаны и отлажены программные модули для управления дискретными входами, дискретными выходами.

Трассировка материнской платы управления автоматизированной теплицы – нетривиальная задача. При проектировании необходимо учитывать множество факторов, а также предусматривать подключения большого количество датчиков, так как для больших теплиц требуется комплексный анализ состояния всех показателей, таких так температура, влажность и так далее.

Написание программы управления в рамках объектно-ориентированного подхода позволяет написать понятный, хорошо читаемый и поддерживаемый код, что важно для больших проектов, над которыми работает команда разработчиков. Также такой подход позволяет грамотно построить архитектуру приложения и разделить программные сущности.

Список использованных источников и литературы

1. GIL-LEBRERO S., GÁMIZ-LÓPEZ V., QUILES-LATORRE F.J., ORTIZ-LÓPEZ M., SÁNCHEZ-RUIZ V., LUNA-RODRÍGUEZ, J.J. HONEY BEE COLONIES REMOTE MONITORING SYSTEM / J.J. GIL-LEBRERO S., GÁMIZ-LÓPEZ V., QUILES-LATORRE F.J., ORTIZ-LÓPEZ M., SÁNCHEZ-RUIZ V., LUNA-RODRÍGUEZ // SENSORS. – 2017. – Т. 17, № 1. – С. 55. – ISSN 1424-8220.

2. Salem, Al-Naemi, Smart sustainable greenhouses utilizing microcontroller and IOT in the GCC countries; energy requirements & economical analyses study for a concept model in the state of Qatar / Al-Naemi Salem, Al-Otoom Awni. // Results in Engineering. — 2023. — № 17. — С. 100.

3. ДАЮБ Н. РАЗВИТИЕ ЦИФРОВИЗАЦИИ СЕЛЬСКОГО ХОЗЯЙСТВА В РОССИИ И ЗАРУБЕЖНЫХ СТРАНАХ // ВЕСТНИК КУРСКОЙ ГОСУДАРСТВЕННОЙ СЕЛЬСКОХОЗЯЙСТВЕННОЙ АКАДЕМИИ. - 2020. - №5. - С. 199-206.

4. Светлов Н.М., Сиптиц С.О., Романенко И.А. Как улучшить размещение отраслей сельского хозяйства России // АПК: экономика, управление. – 2018. - № 3. – С. 13-19.

5. ARM®-based Cortex®-M7 32b MCU+FPU, 462DMIPS, up to 1MB Flash/320+16+ 4KB RAM, USB OTG HS/FS, ethernet, 18 TIMs, 3 ADCs, 25 com itf, cam & LCD [Электронный ресурс] // URL https://www.st.com/resource/en/datasheet/stm32f745ie.pdf (дата обращения: 29.05.2022).

6. Обзор технологии LoRa [Электронный ресурс]// Технологии связи – Электрон. дан. – URL: https://itechinfo.ru/content/обзор-технологии-lora (дата обращения: 17.04.2023)

7. Сети ZigBee. Зачем и почему? [Электронный ресурс]// Хабр – Электрон. дан. – URL: https://habr.com/ru/articles/155037/ (дата обращения: 17.04.2023)

8. DS18B20 Programmable Resolution 1-Wire®Digital Thermometer [Электронный ресурс] // URL https://cdn.sparkfun.com/datasheets/Sensors/Temp/DS18B20.pdf (дата обращения: 29.05.2022).

9. BMP180 Digital pressure sensor [Электронный ресурс] // URL https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf (дата обращения: 29.05.2022).

10. MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b [Электронный ресурс] // URL https://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf (дата обращения: 17.04.2023).

11. MODBUS over Serial Line Specification and Implementation Guide V1.02 [Электронный ресурс] // URL https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf (дата обращения: 17.04.2023).

12. RS-485 ДЛЯ ЧАЙНИКОВ [Электронный ресурс] // ТЕХНОСФЕРА – Электрон. дан. – URL: https://www.ivtechno.ru/articles-one?id=19 (дата обращения: 17.04.2023)

13. MP1584 3A, 1.5MHz, 28V Step-Down [Электронный ресурс] // URL https://static.chipdip.ru/lib/807/DOC005807061.pdf (дата обращения: 29.05.2022).

14. 3V 32M-BIT SERIAL FLASH MEMORY WITH DUAL, QUAD SPI [Электронный ресурс] // URL https://docs.rs-online.com/dede/0900766b81622f8f.pdf (дата обращения: 29.05.2022).

15. NUP2105L. Dual Line CAN Bus Protector [Электронный ресурс] // URL https://static.chipdip.ru/lib/031/DOC001031979.pdf (дата обращения: 17.04.2023).

16. CURRENT SENSOR -LOW TCR AUTOMOTIVE GRADE [Электронный ресурс] // URL https://static.chipdip.ru/lib/270/DOC014270010.pdf (дата обращения: 17.04.2023).

17. LMC7101, LMC7101Q-Q1 Tiny Low-Power Operational Amplifier With Rail-to-Rail Input and Output [Электронный ресурс] // URL https://static.chipdip.ru/lib/993/DOC012993751.pdf (дата обращения: 17.04.2023).

18. ESP8266EX Datasheet [Электронный ресурс] // URL https://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf (дата обращения: 29.05.2022).

ПРИЛОЖЕНИЕ А. Листинги программ

```
class DiscreteIOInterface {
     protected:
          DiscreteIoUnion state_;
     public:
          virtual uint8_t GetNumOfDiscreteIO() = 0;
          virtual void Init() = 0;
          virtual void Update() = 0;
          uint16_t*    GetDiscreteOutStateLink()    {return
&(this->state_.all_value);}
     };
```

Листинг А.1 – Абстрактный класс дискретных входов/выходов

```
union DiscreteIoUnion {
          uint16_t all_value;
          struct {
               uint32_t bit0:1;
               uint32_t bit1:1;
               uint32_t bit2:1;
               uint32_t bit3:1;
               uint32_t bit4:1;
               uint32_t bit5:1;
               uint32_t bit6:1;
               uint32_t bit7:1;
               uint32_t bit8:1;
               uint32_t bit9:1;
               uint32_t bit10:1;
               uint32_t bit11:1;
               uint32_t bit12:1;
               uint32_t bit13:1;
               uint32_t bit14:1;
               uint32_t bit15:1;
          } bit;
     };
```

Листинг А.2 – Объединение, определяющее состояние дискретных входов/выходов

Продолжение приложения А.

```
typedef struct {
          uint16_t* coils_state;
          uint8_t num_of_coils;
          uint16_t* discrete_in_state;
          uint8_t num_of_discrete_in;
          uint32_t** holding_registers;
          uint16_t num_of_holding_registers;
          uint8_t *server_id;
          uint8_t size_of_server_id;
     } DeviceParameters;
```

Здесь

- *coils_state* – ссылка на переменную, хранящую состояние койлов;
- num_of_coils – количество койлов;
- discrete_in_state – ссылка на переменную, хранящую состояние дискретных входов;
- num_of_discrete_in – количество дискретных входов;
- holding_registers – ссылка на массив ссылок на созданные пользователем регистры (регистры должны иметь размера 4 байт);
- num_of_holding_registers – количество регистров;
- server_id – массив данных, который возвращается устройством в ответ на команду 0x11;
- size_of_server_id – размер вышеописанного массива.

Листинг А.3 - Структура параметров устройства, необходимых для инициализации

Продолжение приложения А.

```
class TimerInterface {
public:
     virtual void Update() = 0;        //Данную функию необходимо помещать в основной колбек, она высчитывает необходимую задержку и вызывает функцию колбека
     virtual void Start() = 0;
     virtual void Stop() = 0;
     virtual void Reset() = 0;         //данная функция перезагружает таймер, чтобы он начал работу сначала
     virtual void CalculateNewTimings(uint32_t timer_freq) = 0;
     virtual void CalculateNewTimings(float timer_freq) = 0;
     void SetCallback(TimerCallback callback) {this->callback_ = callback;}
     virtual void SetCallBackWithArg(
          TimerCallbackWithArgument callback_with_arg,
          void* arg)
          {    this->callback_with_arg_ = callback_with_arg;
               this->arg_ = arg;}

     CalcTimingState GetTimingState()
          {return this->calc_timing_state_;}
     ProgrammTimerState GetTimerState()
          {return this->state_;}
protected:
     TimerCallback callback_;
     TimerCallbackWithArgument callback_with_arg_;
     CalcTimingState calc_timing_state_;
     ProgrammTimerState state_;
     void* arg_;
};
```

Листинг А.4 – Абстрактный класс TimerInterface

Продолжение приложения А.

```
class HardwareInterface {
    public:
        virtual void SendData() = 0; //  Данный  метод ОБЯЗАТЕЛЬНО должен отправлять из буффера tx_buf, количеством tx_len
        uint32_t GetCurrentBaudrate()
            {return current_baudrate_;}
        uint8_t tx_buf[255];
        uint8_t rx_buf[255];
        uint8_t tx_len;
        uint8_t rx_len;
        InterfaceState  state_;  //  данное  свойство определяет, занят ли интерфейс (например, отправляет данные) или нет, например, если началась отправка, необходимо установить переменную в kBusy, когда закончится - kSucsecc
    protected:
        uint32_t current_baudrate_;
        uint8_t tx_counter_;
};
```

Листинг А.5 – Абстрактный класс HardwareInterface

Продолжение приложения А.

```
typedef enum {
     kNotInit = 0,
     kZeroState,
     kWaitData,
     kNeedResp,
     kResReq,
     kNeedSendResp,
     kError,
     //ниже прописаны состояния для мастера
     kReseivingResp,
     kResResp,
     kNeedSendReq,
     kNeedReadResp
}State;
```

Здесь

- kNotInit – драйвер не проинициализирован
- kZeroState – начальное состояние драйвера после инициализации
- kWaitData – состояние ожидания данных по физическому интерфейсу
- kNeedResp – необходимо сформировать ответ Master’у
- kNeedSendResp – необходимо отправить ответ Master’у
- kError – ошибка
- kReseivingResp – состояние приема ответа от Slave’а
- kResResp – состояние по окончанию приема ответа от Slave’а
- kNeedSendReq – состояние отправки запроса
- kNeedReadResp – состояние окончания обработки ответа Slave’а

Листинг А.6 – Состояния драйвера Modbus

Продолжение приложения А.

```
typedef enum {
     kReadCoild0x01 = 0,
     kReadDiscreteInputs0x02 = 1,
     kReadHoldingRegisters0x03 = 2,
     kWriteSingleCoil0x05 = 3,
     kWriteMultipleCoils0x0F = 4,
     kWriteMultipleRegisters0x10 = 5,
     kReportServerID0x11 = 6,
     kAllFunctions = 7
} Commands;
```

Листинг А.7 – Перечисление Commands

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

Отчет о проверке на заимствования №1

Автор: Павлов Денис
Проверяющий: Павлов Денис

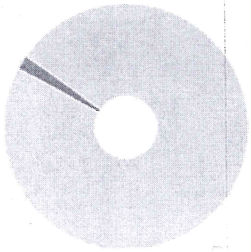
Отчет предоставлен сервисом «Антиплагиат» - http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 9
Начало загрузки: 19.05.2023 05:51:30
Длительность загрузки: 00:00:02
Имя исходного файла: Магистерская диссертация.pdf
Название документа: Магистерская диссертация
Размер текста: 45 кБ
Символов в тексте: 46140
Слов в тексте: 5181
Число предложений: 315

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 19.05.2023 05:51:32
Длительность проверки: 00:00:01
Комментарии: не указано
Модули поиска: Интернет Free



СОВПАДЕНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
1,83%	0%	0%	98,17%

Совпадения — фрагменты проверяемого текста, полностью или частично сходные с найденными источниками, за исключением фрагментов, которые система отнесла к цитированию или самоцитированию. Показатель «Совпадения» – это доля фрагментов проверяемого текста, отнесенных к совпадениям, в общем объеме текста.

Самоцитирования — фрагменты проверяемого текста, совпадающие или почти совпадающие с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа. Показатель «Самоцитирования» – это доля фрагментов текста, отнесенных к самоцитированию, в общем объеме текста.

Цитирования — фрагменты проверяемого текста, которые не являются авторскими, но которые система отнесла к корректно оформленным. К цитированиям относятся также шаблонные фразы; библиография; фрагменты текста, найденные модулем поиска «СПС Гарант: нормативно-правовая документация». Показатель «Цитирования» – это доля фрагментов проверяемого текста, отнесенных к цитированию, в общем объеме текста.

Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.

Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.

Оригинальный текст — фрагменты проверяемого текста, не обнаруженные ни в одном источнике и не отмеченные ни одним из модулей поиска. Показатель «Оригинальность» – это доля фрагментов проверяемого текста, отнесенных к оригинальному тексту, в общем объеме текста.

«Совпадения», «Цитирования», «Самоцитирования», «Оригинальность» являются отдельными показателями, отображаются в процентах и в сумме дают 100%, что соответствует полному тексту проверяемого документа.

Обращаем Ваше внимание, что система находит текстовые совпадения проверяемого документа с проиндексированными в системе источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности совпадений или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в тексте	Источник	Актуален на	Модуль поиска	Комментарии
[01]	1,23%	Программное управление потоком. http://mydocx.ru	08 Июл 2016	Интернет Free	
[02]	1,23%	Управление потоком http://mydocx.ru	11 Июл 2016	Интернет Free	
[03]	1,2%	2.3 Скорость передачи данных. http://studfiles.ru	14 Июл 2016	Интернет Free	

Еще источников: 4
Еще совпадений: 0,59%

Автор работы / Павлов Д.А.

Руководитель ООП / Шарапов Г.Р./

Науч. руководитель / Буркина Е.Ю.