

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет (институт)
информационных технологий
и электроники
(наименование)

Кафедра
«Нано- и микроэлектроника»
(наименование)

Направление подготовки

11.04.04. «Электроника и
наноэлектроника»
(код и наименование)

Магистерская программа

Нано- и микроэлектроника
(наименование)

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

Разработка устройства управления автономными источниками питания

Студент

23ПДм1

номер группы

Козин 07.06.2025
подпись, дата

Козин Алексей Сергеевич

ФИО полностью

Научный руководитель

к.т.н., доцент, кафедра «НиМЭ», ПГУ

должность, наименование организации

А. Н. Головяшкин
07.06.2025
подпись, дата инициалы, фамилия

Рецензент

Директор-главный конструктор,

ООО НПК «Крона»

должность, наименование организации

В. Г. Ханбеков
06.06.2025
подпись, дата инициалы, фамилия

Нормоконтролер

должность

Т. Н. Рыжова
16.06.25
подпись, дата инициалы, фамилия

Работа допущена к защите в ГЭК (протокол заседания кафедры от 09.06.25 № 10)

Заведующий кафедрой НиМЭ

заведующий кафедрой, ПГУ

должность, наименование организации

И.А.Пронин
09.06.25
подпись, дата инициалы, фамилия

Работа защищена с оценкой отл. (протокол ГЭК от 18.06.25 № 2)

Секретарь ГЭК

доцент

должность, наименование организации

А.А. Вольсков
18.06.25
подпись, дата инициалы, фамилия

РЕФЕРАТ

Пояснительная записка состоит из 78-истраниц, 2-хрисунков, 1-йтаблицы, списка использованных источников из 30-и наименований и 8-и приложений.

Ключевые слова: плата *BMS*, интерфейс *RS-485*, интерфейс *RS-232* интерфейс *UART*, система на кристалле, операционная система *FreeRTOS*, источники питания.

Перечень сокращений:

- АКБ – аккумуляторная батарея;
- *BMS – Battery Management System*;
- КЗ – короткое замыкание;
- ЗУ – зарядное устройство;
- *UART – Universal Asynchronous Receiver-Transmitter*;
- *DMA– Direct Memory Access*;
- ЦП – центральный процессор.

Цель: разработать устройство управления автономными источниками питания.

Задачи:

1. Выполнить аналитический обзор научно-технической литературы.
2. Разработать схему электрическую принципиальную на отечественных компонентах.
3. Разработать алгоритм работы для системы на кристалле.
4. Разработать описание встроенного приложения для устройства управления автономными источниками питания.

В настоящее время устройства управления автономными источниками питания, выполненные на основе отечественных компонентах, не

наблюдается. В связи с этим, работа по разработке устройства управления на отечественной компонентной базесегодняшний день актуальная.

Полученные результаты: схема электрическая принципиальная, схема электрическая функциональная, перечень компонентов, встроенное приложение для системы на кристалле.

Плата управления портативными источниками энергии предназначена для защиты аккумулятора и увеличения её срока эксплуатации. Более дорогие модели имеют расширенный функционал, оснащены дисплеями и делают доступной настройку рабочих параметров для оптимизации процесса.

Понятие платы управления включает в себя любые микросхемы, цель которых – защитить и выстроить корректную работу АКБ. Сюда относятся как самые упрощенные платы балансировки или защиты, так и сложные микроконтроллеры с дисплеем для отображения данных и возможностью тонкой настройки параметров.

СОДЕРЖАНИЕ

Введение	6
1 Обзор современной литературы про устройства управления автономными источниками питания	7
1.1 Общие сведения про устройства управления автономными источниками питания	7
1.2 Обзор элементов, входящих в разработанную плату управления портативными источниками энергии	9
2 Обзор технологий, использованных в ходе разработке устройства управления автономными источниками питания	19
2.1 Обзор протокола <i>Modbus RTU</i> и последовательных интерфейсов	19
2.2 Обзор системы реального времени <i>FreeRTOS</i> и набора стандартных библиотек <i>CMSIS</i>	21
3 Разработка устройства управления автономными источниками питания	25
3.1 Разработка электрических схем для устройства управления автономными источниками питания	25
3.2 Алгоритм работы встроенного приложения	26
3.3 Разработка встроенного приложения для устройства управления автономными источниками питания	27
4 Расчет общего искусственного освещения в помещении с ПЭВМ	73
Заключение	75
Список использованных источников.	76

Приложение А. Техническое задание на ВКР	79
Приложение Б. Схема электрическая структурная	83
Приложение В. Схема электрическая функциональная	85
Приложение Г. Схема электрическая принципиальная	87
Приложение Д. Перечень компонентов	89
Приложение Е. Список публикаций по теме ВКР	93
Приложение Ж. Заявление и протокол проверки ВКР на оригинальность в системе «Антиплагиат.ВУЗ»	95
Приложение З. Презентация ВКР	98

ВВЕДЕНИЕ

Плата управления портативными источниками энергии предназначена для защиты аккумулятора и увеличения её срока эксплуатации. Более дорогие модели имеют расширенный функционал, оснащены дисплеями и делают доступной настройку рабочих параметров для оптимизации процесса.

Необходимость применения контроллера вытекает из того, что любая АКБ рассчитана на вполне конкретный рабочий диапазон напряжения. Если хотя бы одна параллель в батарее в ходе эксплуатации превысит предел нагрузки, то запустится необратимая деградация. В самом лёгком случае это приводит к заметной потере ёмкости и длительности срока службы, если не повезёт – может произойти поломка и даже возгорание элемента.

Понятие платы управления включает в себя любые микросхемы, цель которых – защитить и выстроить корректную работу АКБ. Сюда относятся как самые упрощенные платы балансировки или защиты, так и сложные микроконтроллеры с дисплеем для отображения данных и возможностью тонкой настройки параметров[1].

Цель: разработать устройство управления автономными источниками питания.

Задачи:

1. Выполнить аналитический обзор научно-технической литературы.
2. Разработать схему электрическую принципиальную на отечественных компонентах.
3. Разработать алгоритм работы для системы на кристалле.
4. Разработать описание встроенного приложения для устройства управления автономными источниками питания.

Объект исследования: вопрос реализации устройства управления на основе отечественной компонентной базе.

Предмет исследования: разработка устройства управления автономными источниками питания.

1 Обзор современной литературы про устройства управления автономными источниками питания

1.1 Общие сведения про устройства управления автономными источниками питания

1.1.1 Применение устройств управления автономными источниками питания. Плата управления портативными источниками энергии— это электронная система, управляющая зарядом, разрядом, защитой, мониторингом и балансировкой АКБ.

Литий-ионные элементы чувствительны к перезаряду и глубоким разрядам. Одной из важнейших функций платы управления является ограничение заряда выше допустимого уровня, равно как и прекращение разряда ниже допустимого уровня заряда.

АКБ набраны из множества аккумуляторных элементов, рассчитанных на работу в определенном диапазоне напряжений. В процессе эксплуатации уровень напряжения на элементах одной батареи неизбежно начинает различаться: одни элементы остаются недозаряженными, а другие перезаряжаются.

Избежать опасных процессов в аккумуляторной батарее позволяет система балансировки. Система управления батареей (*BMS*) следит за тем, чтобы все элементы в конце заряда получали равное напряжение. При подходе зарядного процесса к концу *BMS* шунтирует зарядившиеся ячейки (пассивная балансировка) либо же перераспределяет энергию элементов с большим напряжением к элементам с меньшим напряжением (активная балансировка). Важно отметить, что активная балансировка работает как при заряде, так и при разряде. В отличие от активной, при пассивной балансировке практически восполнившие заряд ячейки получают меньший ток или совсем исключаются из зарядного процесса до момента, пока все элементы АКБ не будут иметь равный уровень напряжения. При активной

балансировке система не дает элементам разбалансироваться ни при заряде, ни при разряде, а также не теряет энергию на обогреве шунтового резистора, перераспределяя ее внутри АКБ [2-4].

1.1.2 Основные функции устройств управления автономными источниками питания. Система управления батареей просматривает отдельные ячейки внутри АКБ и подсчитывает, какое количество тока может безопасно входить (при зарядке) и выходить (при разрядке), не повреждая АКБ. Это защищает АКБ от перегрузки и выравнивает заряд, что благотворно влияет на состояние АКБ и продлевает срок ее службы.

1. Модуль отслеживает общее напряжение АКБ, напряжение отдельных элементов, не допускает их выхода за рамки допустимых значений.

2. Контроль токов при заряде и разряде: при превышении допустимых значений тока контроллер отключает зарядное устройство или потребляющее оборудование.

3. Контроль исправности: выполняется путем анализа данных, хранящихся в памяти модуля управления: количества циклов перезарядки, граничных значений напряжения и тока.

4. Защитная функция: поддержание оптимальных условий для безопасной работы батареи. Корректное подключение и отключение нагрузки, управление ею, защита АКБ от токовых перегрузок, излишнего заряда, критического разряда, нагрева, утечки тока, КЗ. Плата управления не допускает опасного состояния, непосредственно влияя на батарею (отключая ее), или подает соответствующий сигнал на управляющий контроллер. Отключение АКБ от нагрузки или ЗУ происходит при несоответствии требованиям хотя бы 1 рабочего параметра.

5. Функция связи: плата может передавать сведения о рабочих параметрах на внешние управляющие устройства по проводной или беспроводной связи.

6. Балансировка: распределение энергии между элементами АКБ – для обеспечения корректной зарядки и продления срока службы АКБ.

Кроме отслеживания работы АКБ, плата управления может участвовать в контроле рабочих параметров АКБ, управлять его электрической мощностью и выполнять другие функции [5-6].

1.2 Обзор элементов, входящих в разработанную плату управления портативными источниками энергии

1.2.1 Общие сведения про плату *BMSHKIVI-A06A24S-6P*. Система управления, разработанная специально для литий-ионных АКБ большой ёмкости, представляет собой интеллектуальную плату защиты литий-ионных АКБ.

Среди функций – сбор резервного напряжения, активная балансировка больших токов, защита от перезарядки, переразряда и перегрева. Также имеются счётчики кулонов, *Bluetooth*-коммуникация, дистанционное управление *GPS* и другие функции. Подходит для тройных литий-ионных, литий-железо-фосфатных и других типов АКБ.

Плата защиты может обеспечить это, используя технологию активной балансировки передачи энергии с независимыми правами на интеллектуальную собственность.

2 А – максимальный непрерывный ток выравнивания. Использование технологии активного выравнивания с большим током может гарантировать постоянную максимальную мощность, увеличить срок службы АКБ и замедлить его старение.

Для платы защиты доступно мобильное приложение, поддерживающее *Android* и *iOS*. С помощью приложения можно подключиться к плате защиты через *Bluetooth* на телефоне, чтобы отслеживать рабочее состояние АКБ, изменять рабочие параметры платы защиты, управлять переключателем заряда и разряда и многое другое. Плата защиты компактна, удобна в использовании и функциональна. Она часто применяется в аккумуляторных

блоках для самокатов, автомобилей общего пользования, небольших экскурсионных автомобилей, солнечных электростанций, мощных накопителей энергии и резервных источников питания для базовых станций[7].

Таблица 1.1 – Основные технические характеристики *BMSHKIVI-A06A24S-6P*

Литий-ионная АКБ	Значения
Способ балансировки	Активный Баланс
Текущий активный баланс	0,6А
Непрерывный ток разряда	60А
Пиковый ток разряда	100А
Защита от перегрузки по току	10А-60А регулируется
Шкала напряжения с одной ячейкой	1-5 В
Точность напряжения	±3 мВ
Напряжение защиты от перегрузки	1,2 В-4,35 В регулируется
Снятие защиты от перезаряда	1,2 В-4,55 В регулируется
Время срабатывания защиты от перегрузки по току	Регулируемый 2S-120S
Напряжение защиты от перегрузки	1,2 В-4,35 В регулируется
Защита от чрезмерного разряда	1,2 В-4,35 В регулируется
Количество датчиков температуры	3
Защита от короткого замыкания	ДА
Функция беспроводного управления	Поддержка <i>Android</i> и <i>IOS</i>
Другой коммуникационный порт	<i>RS-485/ CAN</i>
Размер	136*83*17 мм

1.2.2 Общие сведения про инвертор *TIANMOTPU1000W*. Инвертор *TIANMOTPU1000W*– это компактный автомобильный инвертор, преобразующий постоянное напряжение 24 В от АКБ в переменное 220 В для питания маломощных устройств.

Основные характеристики:

- номинальная мощность: 1000 Вт (пиковая до 2000 Вт);
- входное напряжение: 24 В (DC);
- выходное напряжение: 220–240 В (AC), чистая синусоида;
- выходная частота: 50 Гц;
- разъемы: 1–2 розетки (евро/*USB*, зависит от модификации);

- защита: от перегрузки, перегрева, короткого замыкания, низкого/высокого напряжения;
- охлаждение: вентилятор;
- габариты и вес: $20 \times 15 \times 5$ см, вес $\sim 1-1,5$ кг [8].

1.2.3 Общие сведения про систему на кристалле, входящую в состав разработанной платы управления. В состав разработанного устройства управления входит отечественная система на кристалле K1977BE43M.

Сегодня микроконтроллеры с ядром *ARM Cortex M4* широко применяются в различных встраиваемых системах, где нужна большая вычислительная мощность наряду с относительно небольшим энергопотреблением.

Эти устройства отличаются повышенной скоростью обработки чисел с плавающей запятой, простотой интеграции аналоговых цепей и хорошими показателями потребляемой мощности. Микроконтроллеры на основе данного ядра сочетают в себе сложную интеграцию компонентов обработки смешанных сигналов с возможностями многозадачности в реальном времени. Одним из мировых лидеров по производству микроконтроллеров с ядром *CortexM4* является компания *STMicroelectronics*, но в связи глобальным дефицитом микросхем и недавними внешнеполитическими событиями достать продукцию данного производителя стало довольно проблематично. В связи с этим у отечественных разработчиков встает вопрос поиска аналогов микроконтроллеров серии *STM32F4xx*, такой является отечественная система на кристалле K1977BE43M.

На рисунке 1.1 приведены технические характеристики системы на кристалле K1977BE43M[9].

Технические характеристики

Ядро	ARM Cortex-M4F
Частота	240 МГц
FLASH	512 Кб
SRAM	96+128 Кб
Вх/Вых	80
Диапазон напряжений	2,6...3,6 В
Диапазон рабочих температур	-40...105 °C
SPI	4
I2S	2
I2C	3
U(S)ART	4
CAN	2
SDIO	2
Разрядность АЦП	12 бит
Количество АЦП	3
Кол-во каналов АЦП	16
Количество ЦАП	2
Кол-во каналов ЦАП	2

Рисунок 1.1 – Технические характеристики системы на кристалле K1977BE43M

Универсальный асинхронный приёмник/передатчик, обычно называемый *UART*, представляет собой последовательный асинхронный полнодуплексный протокол связи, который широко используется во встраиваемых системах.

Канал *UART* имеет две линии передачи данных. На каждом устройстве есть контакты *RX* и *TX* (*RX*– для приёма, *TX*– для передачи). Контакт *RX* каждого устройства соединён с контактом *TX* другого устройства. Это «асинхронный» аспект универсального асинхронного приёмника-передатчика.

UART, представляющий собой разновидность протокола асинхронной последовательной связи, работает по принципу передачи каждого двоичного бита передаваемых данных побитно. В протоколе связи *UART*, когда состояние сигнальной линии высокое, оно соответствует «1», а когда состояние сигнальной линии низкое, оно соответствует «0».

Последовательная связь подразумевает использование линии передачи для последовательной передачи данных побитно, либо для формирования полнодуплексной связи, могут использоваться две сигнальные линии, например, RS-232. Характерной особенностью является простота линии связи, возможность реализации связи с помощью простого кабеля, снижение затрат и пригодность для связи на большие расстояния, но низкая скорость передачи.

При асинхронной передаче в качестве единицы передачи используется один символ. Временной интервал между двумя символами в процессе передачи не фиксирован, но временной интервал между двумя соседними битами в одном символе фиксирован. Вообще говоря, для связи между двумя устройствами *UART* тактовая линия не требуется. В этом случае на двух устройствах *UART* необходимо указать одинаковую скорость передачи, а также биты ожидания, начальные биты, биты чётности и конечные биты, то есть следовать одному и тому же протоколу.

Скорость передачи данных выражается в бодах, то есть в количестве бит, передаваемых в секунду. Например, если скорость передачи данных составляет 120 символов в секунду, а каждый символ состоит из 10 бит (1 стартовый бит, 7 бит данных, 1 контрольный бит, 1 стоповый бит), то скорость передачи в бодах составляет $10 \times 120 = 1200$ символов в секунду = 1200 бод.

Формат передачи данных.

1. Начальный бит: каждый раз, когда начинается передача данных, отправитель посылает логический сигнал «0», указывающий на начало передачи символа. Поскольку в режиме ожидания шина находится в

состоянии высокого уровня, при начале передачи данных сначала посылается сигнал, явно отличающийся от состояния ожидания.

2. Бит данных: после начального бита идут данные, которые мы хотим передать. Биты данных могут быть 5, 6, 7, 8, 9 бит и т.д. для формирования символа (обычно 8 бит). Младший бит отправляется первым, а старший – последним. Используйте низкий уровень для обозначения "0" и высокий уровень для обозначения «1», чтобы завершить передачу битов данных.

3. Бит четности: после добавления этого бита к биту данных количество «единиц» должно быть четным (четная четность) или нечетным (нечетная четность), чтобы проверить правильность передачи данных. Контрольная цифра на самом деле является корректирующим числом, а проверка последовательного порта выполняется несколькими способами:

- никакого паритета;
- нечётная чётность: если количество «1» в бите данных чётное, то бит чётности равен «1», а если количество «1» нечётное, то бит чётности равен «0».
- четная четность: если количество «1» в данных четное, то бит четности равен «0», а если нечетное, то бит четности равен «1».

4. Стопорный бит: это конечный маркер для символьных данных. Он может быть 1-битным, 1,5-битным или 2-битным. Поскольку данные передаются по линии связи с синхронизацией, а у каждого устройства есть свои часы, вполне вероятно, что между двумя устройствами в процессе передачи будет небольшая рассинхронизация. Таким образом, стоповый бит не только указывает на конец передачи, но и даёт компьютеру возможность скорректировать часы. Чем больше стоповых битов, тем стабильнее передача данных, но тем ниже скорость передачи данных.

Во время работы с *UART* могут возникать ошибки.

1. Ошибки кадрирования. Регистры состояния и управления *UART* будут отображать различные состояния, в том числе состояния ошибок, связанных с передачей. Понимание этого протокола упрощает понимание

состояний ошибок. К ним относятся ошибки кадрирования, которые возникают, когда приёмник не видит стоповый бит в ожидаемое время. Если линия данных не находится в ожидаемом высоком состоянии (в зависимости от количества данных и битов чётности, которые настраивает *UART*) в момент ожидания стопового бита, *UART* выдаст сигнал об ошибке кадрирования.

2. Ошибка четности. Ошибки чётности возникают только в том случае, если *UART* работает в режиме чётности. В режиме чётности отправляется дополнительный бит для определения чётности (чётной или нечётной) всех передаваемых битов данных. Если бит данных ошибочен (1 вместо 0 или 0 вместо 1), бит чётности будет неправильным, и возникнет ошибка с указанием типа ошибки.

3. Ошибка переполнения. Ошибка переполнения возникает, когда получатель не обрабатывает (не удаляет символ из входного буфера) до тех пор, пока не поступит следующий символ.

4. Условие разрыва. Состояние прерывания не обязательно является ошибкой. Это происходит, когда входное устройство приёмника находится в логическом низком состоянии более определённого времени (обычно более одного символа). Для приёмника это выглядит как нулевой символ с ошибкой кадрирования. Это взлом схемы программного сброса [10-14].

Прямой доступ к памяти (*DMA*) – это стандартная аппаратная функция, которая позволяет периферийному устройству напрямую взаимодействовать с основной памятью хоста, не требуя от ЦП управления каждым этапом процесса передачи данных.

Прямой доступ к памяти использует специальный аппаратный компонент, называемый контроллером прямого доступа к памяти (*DMAC*), для управления задачами передачи данных. Это позволяет процессору хоста выполнять другие задачи, пока данные передаются в фоновом режиме.

Архитектура контроллеров прямого доступа к памяти (*DMA*) разработана для обеспечения эффективной и автономной передачи данных

между блоками памяти и периферийными устройствами в обход ЦП для повышения скорости обработки. В основе этой архитектуры лежат несколько ключевых компонентов, каждый из которых выполняет важную функцию в системе.

В основе контроллера прямого доступа к памяти лежит блок управления, который координирует весь процесс передачи данных. Блок управления содержит набор инструкций или сигналов, определяющих, когда и как должна происходить передача данных. С функцией управления связаны регистры данных и адресные регистры, которые играют важную роль в хранении данных и адресации соответственно. Регистры данных временно хранят данные, которые передаются между памятью и периферийными устройствами. Адресные регистры, в свою очередь, хранят адреса данных в памяти, обеспечивая точность операций чтения и записи.

Контроллеры *DMA* могут быть классифицированы на различные типы в зависимости от их архитектуры и функциональности. Две распространенные формы – одноканальные и многоканальные контроллеры *DMA*. Одноканальные контроллеры управляют операциями передачи данных между одним периферийным устройством и памятью одновременно, что делает их подходящими для простых задач с низкой пропускной способностью. Они часто используются в ситуациях, когда эффективность использования ресурсов имеет первостепенное значение и когда потребности в передаче данных не являются очень сложными.

Многоканальные контроллеры прямого доступа к памяти, напротив, могут одновременно обрабатывать несколько каналов передачи данных. Эта возможность важна в системах, где несколько периферийных устройств должны одновременно взаимодействовать с памятью. Многоканальные контроллеры часто используются в высокопроизводительных вычислительных средах, таких как серверы и сложные встраиваемые системы, где параллельная обработка данных имеет решающее значение для оптимизации общей производительности системы.

Процесс работы *DMA* начинается с запроса на передачу данных, который может исходить как от ЦП, так и от периферийного устройства, требующего передачи данных. Получив этот запрос, контроллер *DMA* берёт на себя управление и настраивает необходимые параметры, такие как адреса источника и назначения, направление передачи и размер блока данных. Эта конфигурация может быть запрограммирована ЦП через управляющие регистры контроллера *DMA*.

После завершения настройки контроллер *DMA* генерирует управляющие сигналы для координации передачи данных. Он отправляет запрос контроллеру памяти, чтобы получить доступ к шине памяти. Когда доступ к памяти получен, контроллер *DMA* напрямую считывает данные из исходного местоположения в памяти и записывает их в место назначения, минуя ЦП. Этот цикл продолжается до тех пор, пока не будет передан весь блок данных. После завершения контроллер *DMA* отправляет сигнал прерывания на ЦП, сигнализируя об успешной передаче.

DMA обладают рядом ключевых преимуществ, которые значительно повышают производительность и эффективность вычислительных систем. Благодаря возможности прямой передачи данных между памятью и периферийными устройствами контроллеры *DMA* значительно снижают нагрузку на ЦП. Разгружая ЦП от задач по передаче данных, контроллеры *DMA* высвобождают его для выполнения других критически важных задач, тем самым оптимизируя производительность системы.

Одним из основных преимуществ использования контроллеров *DMA* является повышенная скорость передачи данных. В отличие от традиционной передачи данных через ЦП, контроллеры *DMA* могут передавать данные на более высокой скорости, поскольку они работают независимо от тактовых циклов ЦП. Такая независимая работа позволяет эффективно перемещать пакеты данных большого размера, минимизируя узкие места и повышая общую пропускную способность как в однопользовательских, так и в многозадачных средах.

Повышение эффективности системы – ещё одно ключевое преимущество. Поскольку контроллеры *DMA* автономно обрабатывают передачу данных, снижается задержка системы, а периферийные устройства работают более плавно. Например, в приложениях с интенсивным использованием графики, таких как рендеринг видео или игры, контроллеры *DMA* могут напрямую передавать большие объёмы графических данных из памяти в графический процессор (*GPU*) без постоянного вмешательства ЦП. Это не только повышает визуальную производительность, но и обеспечивает бесперебойную и быструю работу пользователя.

Реальное повышение производительности с помощью контроллеров *DMA* можно наблюдать в различных приложениях. В обработке звука контроллеры *DMA* позволяют передавать звуковые данные непосредственно на звуковую карту, тем самым обеспечивая высококачественный звук без значительной нагрузки на процессор. Аналогичным образом в устройствах хранения данных, таких как твердотельные накопители (*SSD*), контроллеры *DMA* помогают повысить скорость записи и чтения данных, что приводит к сокращению времени загрузки системы и улучшению операций передачи файлов[15-18].

2 Обзор технологий, использованных в ходе разработке устройства управления автономными источниками питания

В ходе разработке устройства управления автономными источниками питания были использованы следующие методы и технологии:

- инвертор управляется по протоколу *ModbusRTU* с использованием интерфейса *RS-232*;
- плата *BMS* выдает информацию раз в секунду с использованием интерфейса *RS-485*;
- встроенное приложение разработано с использованием операционной системы реального времени *FreeRTOS* и использованием набора стандартных библиотек *CMSIS*.

2.1 Обзор протокола *Modbus RTU* и последовательных интерфейсов

ModbusRTU (*RemoteTerminalUnit* – удаленный терминальный блок) – один из наиболее распространенных протоколов связи в автоматизации. Разработан для обмена данными между устройствами.

ModbusRTU использует последовательный интерфейс (*RS-232* или *RS-485*) для связи. Он основан на принципе «ведущий-ведомый»: главное устройство (ведущий) первым начинает обмен данными, а подчиненное устройство (ведомый) отвечает только после окончания обмена главного устройства.

На рисунке 2.1 представлена структура кадра сообщения в протоколе *ModbusRTU* [19].

Начало	Адрес устройства	Код функции	Данные	Контрольная сумма CRC	Конец
Интервал, равный времени передачи 4 символов	8 бит	8 бит	n×8 бит	16 бит	Интервал, равный времени передачи 4 символов

Рисунок 2.1 – Структура кадра сообщения в протоколе *ModbusRTU*

Интерфейс *RS-232*— это интерфейс, предназначенный для обмена данными между компьютерами, модемами, маршрутизаторами и т. д. Этот процесс осуществляется по прямой связи, то есть между двумя устройствами, без участия каких-либо дополнительных узлов.

Одной из главных особенностей *RS-232* является его способность передавать информацию на короткие расстояния по двум проводам — один для передачи данных, второй для приема. Это обеспечивает параллельный прием и передачу, простоту и надежность соединения между устройствами.

Скорость работы интерфейса *RS-232* может значительно варьироваться в зависимости от расстояния между соединяемыми устройствами. Обычно, при расстоянии до 15 метров, скорость передачи данных может составлять до 9600 бит в секунду. На минимальных же дистанциях скорость может достигать 115,2 килобит в секунду.

С развитием технологий *RS-232* постепенно уступает место более новым и быстрым интерфейсам, однако его продолжают использовать в некоторых областях, где требуется надежное и простое соединение между устройствами. Это особенно актуально для промышленных систем и систем автоматизации, где *RS-232* обеспечивает необходимую надежность и стабильность работы.

Еще одной областью применения *RS-232* являются системы связи и передачи данных. Благодаря своим характеристикам, данный интерфейс идеально подходит для создания простых и надежных каналов связи между различными устройствами, включая модемы, маршрутизаторы и другое телекоммуникационное оборудование.

Интерфейс *RS-485* является одним из наиболее популярных и надежных протоколов для организации промышленной связи между различными устройствами и системами. Он был разработан в 1983 году и получил свое название благодаря использованию кабеля с волновым сопротивлением 485 Ом.

Интерфейс RS-485 имеет многоточечный режим работы и позволяет подключать до 32 устройств к одной линии связи (с повторителями больше, обычно до 256). Это позволяет создавать системы с большим количеством устройств без необходимости использования дополнительных кабелей и оборудования.

Благодаря использованию дифференциального сигнала, RS-485 обеспечивает передачу данных на большие расстояния. Скорость передачи данных составляет до 9600 бит/с на расстоянии до 1200м. Чем больше скорость, тем меньше дальность стабильной связи.

Устройства, подключенные к интерфейсу RS-485, могут питаться от одного источника питания с напряжением от 5 до 24 В. Это делает его идеальным решением для систем с большим количеством компонентов, не требуя отдельного питания для каждого устройства.

RS-485 используется в различных областях, включая автоматизацию зданий, системы управления освещением, системы безопасности, системы контроля доступа, системы видеонаблюдения и многие другие. Благодаря своей надежности и гибкости, RS-485 стал одним из самых популярных интерфейсов для организации взаимодействия между устройствами в промышленных системах.

Интерфейс RS-485 представляет собой надежный и гибкий способ организации связи между устройствами и системами, обеспечивая высокую скорость передачи данных, многоточечный режим работы и устойчивость к помехам[20-23].

2.2 Обзор системы реального времени *FreeRTOS* и набора стандартных библиотек *CMSIS*

FreeRTOS – это минималистичная операционная система реального времени с открытым исходным кодом, написанная на языке C. Она разработана специально для микроконтроллеров и встраиваемых устройств.

Главная цель *FreeRTOS* – обеспечить планирование задач и синхронизацию между ними в средах с ограниченными ресурсами.

Задачи, выполняемые *FreeRTOS*:

- упрощает многозадачность – можно писать код для каждой задачи отдельно;
- позволяет задавать приоритеты задачам;
- обеспечивает синхронизацию между задачами и защищённый доступ к ресурсам;
- позволяет масштабировать проект без «адского» копияста и состояний;
- повышает читаемость и поддержку кода.

Система довольно компактна – основное ядро *FreeRTOS* занимает от 4 до 10 КБ памяти и легко встраивается даже в ограниченные микроконтроллеры. Архитектура построена вокруг задач (*tasks*), которые исполняются в соответствии с планировщиком.

Основные механизмы:

- задачи: аналог потоков;
- планировщик: определяет, какая задача сейчас должна работать.

Задачи – это основные исполняемые единицы в *FreeRTOS*. Они представляют собой функции, которые выполняются "параллельно" (на самом деле, планировщик быстро переключает их, создавая иллюзию параллелизма).

Для создания задачи используется функция *xTaskCreate()* или *xTaskCreateStatic()* (если нужен статический стек). Задача может находиться в одном из 4 состояний:

- *running*: задача выполняется на ядре (в данный момент активна);
- *ready*: задача готова к выполнению, но ожидает, т.к. есть задача с высшим приоритетом;
- *blocked*: задача приостановлена (например, из-за *vTaskDelay()*, ожидания семафора);

- *suspended*: задача явно приостановлена (*vTaskSuspend()*), пока не возобновлена (*vTaskResume()*).

Также в *FreeRTOS* есть возможность управления задачами:

- задержка (*vTaskDelay()*): задача приостанавливается на время, передаваемое в *vTaskDelay()*, в мс. В это время выполняются другие задачи;
- приостановка и возобновление (*vTaskSuspend*, *vTaskResume*): задача приостанавливается пользователем до момента, пока не он не возобновит ее;
- удаление задачи (*vTaskDelete*): пользователь удаляет задачу.

Также у задач есть приоритеты: чем выше приоритет, тем чаще планировщик будет вызывать данную задачу для выполнения. Если задачи будут иметь одинаковый приоритет, то задачи выполняются по очереди.

Каждая задача имеет свой стек: память, выделяемая на данную задачу. Если будет недостаточно памяти для выполнения задачи, то будет переполнение стека и разрабатываемая программа попадет в *HardFault*. *HardFault* – это специальное системное исключение в процессорах *ARM Cortex-Mx*. Оно указывает на серьёзную ошибку в системе и возникает из-за различных условий, которые процессор не может обработать через другие механизмы исключений.

FreeRTOS можно конфигурировать через файл *FreeRTOSConfig.h* – включать или отключать нужные модули, задавать размеры стеков и прочие параметры[24-27].

CMSIS (стандарт общего программного интерфейса микроконтроллеров) – это набор программных компонентов, инструментов и рабочих процессов, которые помогают упростить повторное использование программного обеспечения, сократить время обучения разработчиков микроконтроллеров, ускорить сборку и отладку проектов и тем самым сократить время выхода на рынок новых приложений.

CMSIS был создан для того, чтобы помочь отрасли в стандартизации. Он обеспечивает согласованность программных уровней и поддержку устройств в широком спектре инструментов разработки и

микроконтроллеров. *CMSIS* – это не огромный программный уровень, который требует дополнительных ресурсов и не определяет стандартные периферийные устройства. Таким образом, с помощью этого общего стандарта полупроводниковая промышленность может поддерживать широкий спектр устройств на базе процессоров *ArmCortex*.

Преимущества *CMSIS* заключаются в следующем:

- *CMSIS* сокращает время обучения, затраты на разработку и время выхода на рынок. Разработчики могут быстрее создавать программное обеспечение с помощью различных простых в использовании стандартизированных программных интерфейсов;

- согласованные программные интерфейсы повышают переносимость и возможность повторного использования программного обеспечения. Универсальные программные библиотеки и интерфейсы обеспечивают согласованную программную среду;

- он предоставляет интерфейсы для подключения к отладке, периферийные устройства для отладки, доставку программного обеспечения и поддержку устройств, чтобы сократить время выхода на рынок для внедрения новых микроконтроллеров;

- он позволяет использовать компилятор по вашему выбору, так как не зависит от компилятора и поддерживается основными компиляторами[28-29].

3 Разработка устройства управления автономными источниками питания

3.1 Разработка электрических схем для устройства управления автономными источниками питания

В приложении Б приведена схема электрическая структурная. По ней выбирались готовые блоки, использующиеся в разработанном устройстве: плата *BMSHKIVI-A06A24S-6P*, инвертор *TIANMOTPU1000W*.

В приложении В представлена схема электрическая функциональная для устройства управления автономными источниками питания. По ней видно, что необходимо связываться с инвертором по RS-232, с платой *BMS* по RS-485, питание платы составляет 24 В.

По данным требованиям были выбраны следующие микросхемы:

- 5559ИН28У: интерфейс RS-485;
- 5559ИН1Т: интерфейс RS-232;
- K1977BE43М: система на кристалле для управления устройством;
- МДМ10-1Б3.3СН: модуль питания, преобразующий 24В в 3,3В.

В приложении Г представлена схема электрическая принципиальная.

Резисторы *R1-R4*, *R8-R16*, *R31* считаются по формуле 3.1:

$$R = U/I, \quad (3.1)$$

где: $I = 12 \text{ мА}$, $U = 3,3 \text{ В}$.

Тогда сопротивление резисторов равно:

$$R = 3,3/0,012 = 275 \text{ Ом}.$$

По ряду *E24* выбираем ближайший подходящий номинал. Тогда, сопротивление равно 270 Ом.

Резисторы *R5-R7* выбираются исходя из технической документации на микросхему 5559ИН28У.

Резисторы *R24-R30* рассчитываются по формуле 3.1, но $I = 27,5 \text{ мА}$. Тогда сопротивление резисторов равно:

$$R = 3,3/0,0275 = 120 \text{ Ом}.$$

Рассчитанное сопротивление соответствует ряду *E24*.

Резисторы *R20*, *R22* и *R23* рассчитываются по формуле 3.1, но $I = 6$ мА – ток, необходимый для открытия биполярного транзистора 2Т665А9.

Тогда сопротивление резисторов равно:

$$R = 3,3/0,006 = 550 \text{ Ом.}$$

По ряду *E24* выбираем ближайший подходящий номинал. Тогда, сопротивление равно 560 Ом.

Резисторы *R18*, *R19* и *R21* необходимы для того, чтобы ключи на биполярном транзисторе не переходили в открытое состояние во время включения устройства.

Оптореле *KA1 PRAB34S* необходимо для включения/выключения инвертора.

В приложении Д представлен перечень компонентов.

3.2 Алгоритм работы встроенного приложения

Система на кристалле получает данные с платы *BMS*. После этого происходит извлечение необходимой информации: напряжение на АКБ, уровень оставшегося заряда АКБ, выходной ток с АКБ. На основе этих данных формируются сигналы для отображения полученной информации.

1. Контроль параметров. Если уровень оставшегося заряда АКБ равен 0%, то в плату инвертора отправляется сигнал об его выключении.

2. Отображение оставшегося заряда АКБ. Данные про оставшейся заряда АКБ приходят от 0% до 100%. Всего на отображение задействовано 7 светодиодов: 4 зеленых, 1 желтый и 2 красных. Если уровень заряда от 87,5% до 100% горят все светодиоды, от 75% до 87,5% не горит один зеленый светодиод, от 62,5% до 75% не горят два зеленых светодиода и т. д.

3. Отображение заряда/разряда устройства. Если выходной ток с АКБ положительный, то происходит заряд АКБ. Если отрицательный, то происходит разряд АКБ.

4. Отображение режима работы устройства. Если полученная мощность больше 0, то устройство выдает напряжение. Если идет заряд АКБ, то в устройство подключено внешнее напряжение. Эти режимы могут работать одновременно. Если это происходит, то устройство работает транзитом, то есть не расходует заряд с АКБ, а передает его с внешнего источника.

Также включение/выключение платы инвертора может производиться вручную: тогда происходит проверка, включена/выключена плата инвертора, уровень заряда АКБ. Во время включения/выключения звучит звуковой сигнал в течении 10 мс, который сигнализирует об изменении режима работы платы инвертора.

3.3 Разработка встроенного приложения для устройства управления автономными источниками питания

Данное встроенное приложение разработано в среде программирования *keil*. Использован *CMSIS* и *FreeRTOS*.

Встроенное приложение представлено в виде двух блоков: *Hardware* и *StateMachines*.

В блоке *Hardware* описывается так называемая «привязка» к конкретной аппаратной платформе. В нашем случае к системе на кристалле K1977BE43M.

В блоке *StateMachines* описываются основные алгоритмы работы устройства: связь с другими платами, отображение полученной информации, управление запуском.

Главный файл: *DIPLOM.c* – в нем инициализируются все выводы, интерфейсы, тактирование системы на кристалле, а также происходит запуск задач *FreeRTOS*. Ниже представлено описание файла *DIPLOM.c*:

```
/*-----  
* Name:   PB_C.c  
* Purpose: AT32F407VE (100-pin) main project file Power Bank China  
* Version: V1.0
```

* Date: 30.01.2024

* Note(s):

-----/

```
#include "at32f403a_407_conf.h"
```

```
#include "FreeRTOS.h"
```

```
#include "task.h"
```

```
#include "stdio.h"
```

```
#include "../Hardware/GPIO.h"    /* Ports init          */
```

```
#include "../Hardware/UART.h"    /* USARTs init         */
```

```
#include "../Hardware/DMA.h"     /* DMA init            */
```

```
#include "KB_LED.h"              /* KeyBoard and LED    */
```

```
#include "INVERTER.h"           /* Inverter connection */
```

```
#include "BMS.h"                /* BMS connection     */
```

```
#include "CONTROL.h"            /* Main control function */
```

```
#define MAX_TASK 6
```

```
uint32_t heap_size[MAX_TASK];
```

```
void wait_delay (int16_t ms)
```

```
{uint32_t i, j;
```

```
  j = (uint32_t)ms;
```

```
  j = j << 11;
```

```
  for (i = 0; i < j; i++)
```

```
  {
```

```

__ASM("nop");
}
}

```

```

void initial_jtag_defend_delay (int16_t ms){wait_delay (ms);}

```

```

#define ENABLE_CLOCK(X)    crm_clock_source_enable(X, TRUE);
#define WAIT_CLOCK_STABLE(X) while(crm_flag_get(X) != SET) {}

```

```

const crm_periph_clock_type PBC_BLOCK_CLOCK_INIT [] = {
    CRM_ACC_PERIPH_CLOCK,
    CRM_IOMUX_PERIPH_CLOCK,
    CRM_GPIOA_PERIPH_CLOCK,
    CRM_GPIOB_PERIPH_CLOCK,
    CRM_GPIOC_PERIPH_CLOCK,
    CRM_GPIOD_PERIPH_CLOCK,
    CRM_GPIOE_PERIPH_CLOCK,
    CRM_DMA1_PERIPH_CLOCK,
    CRM_USART1_PERIPH_CLOCK,
    CRM_UART7_PERIPH_CLOCK
};

```

```

#define PBC_BLOCK_CLOCK_SIZE
(sizeof(PBC_BLOCK_CLOCK_INIT)/(sizeof(crm_periph_clock_type)))

```

```

void pbc_system_clock_config(void)

```

```

{uint8_t i;

```

```

    crm_reset();

```

```

ENABLE_CLOCK   (CRM_CLOCK_SOURCE_LICK)
WAIT_CLOCK_STABLE(CRM_LICK_STABLE_FLAG)
ENABLE_CLOCK   (CRM_CLOCK_SOURCE_HICK)
WAIT_CLOCK_STABLE(CRM_HICK_STABLE_FLAG)

crm_pll_config(CRM_PLL_SOURCE_HEX,      CRM_PLL_MULT_25,
CRM_PLL_OUTPUT_RANGE_GT72MHZ);

ENABLE_CLOCK   (CRM_CLOCK_SOURCE_PLL)
WAIT_CLOCK_STABLE(CRM_PLL_STABLE_FLAG)

crm_ahb_div_set (CRM_AHB_DIV_1);
crm_apb2_div_set(CRM_APB2_DIV_2);
crm_apb1_div_set(CRM_APB1_DIV_2);

crm_auto_step_mode_enable(TRUE);
crm_sysclk_switch(CRM_SCLK_PLL);
while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL) {}

crm_auto_step_mode_enable(FALSE);
system_core_clock_update();

for    (i    =    0;    i<    PBC_BLOCK_CLOCK_SIZE;    i++)
crm_periph_clock_enable(PBC_BLOCK_CLOCK_INIT[i], TRUE);
}

void pbc_init (void) {
SystemInit ();
pbc_system_clock_config();

```

```

GPIO_init ();
UART_init ();
DMA_init ();

KB_LED_init ();
INVERTER_init ();
BMS_init ();
CONTROL_init ();

nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);

DMA_irq_init();
UART_irq_init();
}

void vApplicationIdleHook(void){}
void vApplicationStackOverflowHook(TaskHandle_t xTask, char *
pcTaskName){}

int main(void) {
    __ASM("nop");
#ifdef SIMULATION_MODE
    initial_jtag_defend_delay (1000);
#endif

    pbc_init ();

    heap_size[0] = xPortGetFreeHeapSize();
    xTaskCreate(&KB_LED_main, "KB_LED",
3*configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY+0, NULL);

```

```

    heap_size[1] = xPortGetFreeHeapSize();
    xTaskCreate(&INVERTER_main, "INVERTER",
3*configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY+0, NULL);
    heap_size[2] = xPortGetFreeHeapSize();
    xTaskCreate(&BMS_main, "BMS",
3*configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY+0, NULL);
    heap_size[3] = xPortGetFreeHeapSize();
    xTaskCreate(&CONTROL_main, "CONTROL",
3*configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY+0, NULL);
    heap_size[4] = xPortGetFreeHeapSize();
    vTaskStartScheduler();
}

```

В файле *GPIO.c* описываются функции для работы с выводами, а именно: *reset_all_ports*, *init_used_ports*, *GPIO_init*.

В функции *reset_all_ports* все выводы системы на кристалле сбрасываются в исходное состояние.

В функции *init_used_ports* все выводы системы на кристалле, используемые в схеме, назначаются теми функциями, которые они должны выполнять в разработанной схеме.

При вызове функции *GPIO_init* выполняются функции, описанные выше.

Ниже представлено описание файла *GPIO.c*:

```

/*-----
* Name:   GPIO.c
* Purpose: AT32F407VE (100-pin) GPIO init
* Version: V1.0
* Date:   25.01.2024
* Note(s):
*-----*/

```

```

#include "GPIO.h"

#define SET_1 1
#define SET_0 0

typedef struct {
    uint16_t    pin;
    gpio_type*   port;
    gpio_mode_typefunc;
    uint8_t      initial;
} set_port_t;

#define SET_PIN(PIN, PORT, FUNC, INITIAL) {PIN, PORT, FUNC,
INITIAL}

const set_port_t PINS_CODE [] = {

    SET_PIN(PIN_USART1_RXD,          PORT_USART1_RXD,
    GPIO_MODE_INPUT, SET_0),
    SET_PIN(PIN_USART1_TXD,          PORT_USART1_TXD,
    GPIO_MODE_MUX,   SET_0),

    SET_PIN(PIN_UART7_RXD,           PORT_UART7_RXD,
    GPIO_MODE_INPUT, SET_0),
    SET_PIN(PIN_UART7_TXD,           PORT_UART7_TXD,
    GPIO_MODE_MUX,   SET_0),
    SET_PIN(PIN_UART7_DE,            PORT_UART7_DE,
    GPIO_MODE_OUTPUT, SET_0),

```

SET_PIN(PIN_HL1,	PORT_HL1,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL2,	PORT_HL2,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL3,	PORT_HL3,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL4,	PORT_HL4,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL3,	PORT_HL5,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL4,	PORT_HL6,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL5,	PORT_HL7,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL6,	PORT_HL8,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL9,	PORT_HL9,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL10,	PORT_HL10,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL11,	PORT_HL11,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_HL12,	PORT_HL12,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_A,	PORT_A,	GPIO_MODE_OUTPUT,
SET_0),		
SET_PIN(PIN_B,	PORT_B,	GPIO_MODE_OUTPUT, SET_0),

```

        SET_PIN(PIN_C,      PORT_C,      GPIO_MODE_OUTPUT, SET_0),
        SET_PIN(PIN_D,      PORT_D,      GPIO_MODE_OUTPUT,
SET_0),
        SET_PIN(PIN_E,      PORT_E,      GPIO_MODE_OUTPUT, SET_0),
        SET_PIN(PIN_F,      PORT_F,      GPIO_MODE_OUTPUT, SET_0),
        SET_PIN(PIN_G,      PORT_G,      GPIO_MODE_OUTPUT,
SET_0),
        SET_PIN(PIN_P,      PORT_P,      GPIO_MODE_OUTPUT, SET_0),
        SET_PIN(PIN_VT1,    PORT_VT1,    GPIO_MODE_OUTPUT,
SET_0),
        SET_PIN(PIN_VT2,    PORT_VT2,    GPIO_MODE_OUTPUT,
SET_0),
        SET_PIN(PIN_VT3,    PORT_VT3,    GPIO_MODE_OUTPUT,
SET_0),

        SET_PIN(PIN_INV_VKL,                                PORT_INV_VKL,
GPIO_MODE_OUTPUT, SET_0),
        SET_PIN(PIN_INV_BUT,  PORT_INV_BUT,  GPIO_MODE_INPUT,
SET_0),
        SET_PIN(PIN_BUZZER,                                PORT_BUZZER,
GPIO_MODE_OUTPUT, SET_0)
    };

```

```

#define PINS_CODE_SIZE (sizeof(PINS_CODE)/sizeof(set_port_t))

```

```

void reset_all_ports(void)
{
    gpio_bits_reset(GPIOA, GPIO_PINS_ALL);
    gpio_bits_reset(GPIOB, GPIO_PINS_ALL);
    gpio_bits_reset(GPIOC, GPIO_PINS_ALL);
}

```

```

gpio_bits_reset(GPIOD, GPIO_PINS_ALL);
gpio_bits_reset(GPIOE, GPIO_PINS_ALL);
}

void init_used_ports(void)
{ gpio_init_type p;
  constset_port_t* c;
  uint8_t i;

  gpio_pin_remap_config(SWJTAG_MUX_010, TRUE);
  gpio_pin_remap_config(USART3_GMUX_0001, TRUE);

  p.gpio_pull = GPIO_PULL_NONE;
  p.gpio_out_type = GPIO_OUTPUT_PUSH_PULL;
  p.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  for (i = 0; i < PINS_CODE_SIZE; i++) {
    c = &PINS_CODE[i];
    p.gpio_pins = c->pin;
    p.gpio_mode = c->func;
    if (GPIO_MODE_INPUT == c->func) {
      p.gpio_pull = GPIO_PULL_UP;
    }
    gpio_init(c->port, &p);
    if (GPIO_MODE_OUTPUT == c->func) {
      if (c->initial) gpio_bits_set (c->port, c->pin); else gpio_bits_reset (c-
>port, c->pin);
    }
  }
}

```

```
void GPIO_set_pin(gpio_type* port, uint16_t pin) {
    gpio_bits_set(port, pin);
}
```

```
void GPIO_reset_pin(gpio_type* port, uint16_t pin) {
    gpio_bits_reset(port, pin);
}
```

```
uint8_t GPIO_read_output_pin(gpio_type* port, uint16_t pin) {
    return gpio_output_data_bit_read(port, pin);
}
```

```
uint8_t GPIO_read_input_pin(gpio_type* port, uint16_t pin) {
    return gpio_input_data_bit_read(port, pin);
}
```

```
void GPIO_init(void) {
    reset_all_ports();
    init_used_ports();
}
```

В файле *UART* описаны функции инициализации интерфейса *uart*, а также основные функции, которые понадобятся для работы с этим интерфейсом.

В функции *UART_irq_init* происходит разрешение прерывания интерфейса *UART*.

В функции *uart_receive_reinit* происходит переинициализация канала DMA приемника интерфейса *UART*.

В функции *uart_switch_on_transmit* происходит переключение интерфейса *UART* на передачу для интерфейса RS-485.

В функции *uart_switch_on_receive* происходит переключение интерфейса *UART* на прием.

В функции *uart_receive_stop* происходит остановка приемника интерфейса *UART*.

В функции *uart_receive_start* происходит включение приемника интерфейса *UART*.

В функции *uart_receive_set_queue* происходит передача принятых данных интерфейса *UART* в другую задачу.

В функции *uart_receive_complite* происходит подсчет количества принятых байт данных с помощью интерфейса *UART* и передача этих данных в другую задачу.

В функции *uart_rxd_message_waiting* происходит ожидание приема количества байт данных из другой задачи.

В функции *UART_init* происходит инициализация интерфейса *UART*: режим работы, скорость и т. д.

В функции *uart_put_pkt* происходит передача данных с помощью интерфейса *UART* с использованием канала *DMA*.

В функции *uart_init_wait_rxd_pkt* происходит ожидание приема данных.

В функции *uart_get_pkt* происходит ожидание приема данных из другой задачи.

В функции *USART1_IRQHandler* происходит обработка прерывания, возникающее при приеме данных первого интерфейса *UART*.

В функции *USART7_IRQHandler* происходит обработка прерывания, возникающее при приеме данных седьмого интерфейса *UART*.

Ниже представлено описание файла *UART.c*:

```
/*-----
```

```
* Name:   UART.c
```

```
* Purpose: AT32F407VE (100-pin) UART communication for host control unit
```

```
* Version: V1.0
```

* Date: 25.01.2024

* Note(s):

-----/

#include "UART.h"

#include "GPIO.h"

#include "at32f403a_407_conf.h"

#include "../Hardware/DMA.h" /* DMA init */

#include "string.h"

#include "FreeRTOS.h"

#include "task.h"

#include "message_buffer.h"

#define UART_BEFORE_SWITCH_ON_TRANSMIT_IN_MS 1

#define UART_AFTER_SWITCH_ON_TRANSMIT_IN_MS 1

#define UART_BEFORE_SWITCH_OFF_TRANSMIT_IN_MS 1

#define UART_AFTER_SWITCH_OFF_TRANSMIT_IN_MS 1

#define UART_DMA_RXD_FRAME_BLOCK_USART1 16

#define UART_DMA_RXD_FRAME_BLOCK_USART7 64

#define USART7_SPEED 2400

#define USART1_SPEED 115200

typedef struct{

usart_type* usart_x;

```

uint32_t          baud_rate;
IRQn_Type irq;
gpio_type*        port;
uint32_t          pin;
DMA_LINK_NUMBER_t dma_tx;
DMA_LINK_NUMBER_t dma_rx;
uint16_t          rx_size;
}uart_init_t;

const uart_init_t UART_INIT[] = {
    {USART1, USART1_SPEED, USART3_IRQn,    NULL,    NULL,
DMA_UART1_TXD_0,                                DMA_UART1_RXD_1,
UART_DMA_RXD_FRAME_BLOCK_USART1},
    {UART7,    USART7_SPEED,  UART7_IRQn,    PORT_UART7_DE,
PIN_UART7_DE,    DMA_UART7_TXD_2,    DMA_UART7_RXD_3,
UART_DMA_RXD_FRAME_BLOCK_USART7}
};

#define UART_MAX_NUMBER (sizeof(UART_INIT)/sizeof(uart_init_t))

uint8_t          uart_rxd_data [UART_MAX_NUMBER];

uint32_t          uart_counter_txd [UART_MAX_NUMBER];
uint32_t          uart_counter_rxd [UART_MAX_NUMBER];

MessageBufferHandle_t uart_handle [UART_MAX_NUMBER];

void UART_irq_init(void)
{
    const uart_init_t* u;
    uint8_t          i;

```

```

    for (i = 0; i < UART_MAX_NUMBER; i++) {
        u = &UART_INIT[i];
        nvic_irq_enable(u->irq, 0, 0);
    }
}

void uart_receive_reinit (UART_LINK_NUMBER_t link)
{
    const uart_init_t* u;

    u = &UART_INIT[link];
    usart_interrupt_enable (u->usart_x, USART_IDLE_INT, TRUE);
    usart_dma_receiver_enable (u->usart_x, TRUE);
    if (DMA_NULL != u->dma_rx) {
        DMA_init_transaction (u->dma_rx, (uint32_t)&(u->usart_x->dt),
            (uint32_t)uart_rxd_data[link], u->rx_size);
        DMA_start (u->dma_rx);
    }
}

void uart_switch_on_transmit (UART_LINK_NUMBER_t link) {
    vTaskDelay(UART_BEFORE_SWITCH_ON_TRANSMIT_IN_MS);
    if (NULL != UART_INIT[link].port) {
        gpio_bits_set (UART_INIT[link].port, UART_INIT[link].pin);
        uart_receive_stop (link);
    }
}

void uart_switch_on_receive (UART_LINK_NUMBER_t link) {
    vTaskDelay(UART_BEFORE_SWITCH_OFF_TRANSMIT_IN_MS);

```

```

        if (NULL != UART_INIT[link].port) gpio_bits_reset
(UART_INIT[link].port, UART_INIT[link].pin);
    uart_init_wait_rxd_pkt (link, &uart_rxd_data[link]);
    uart_receive_start (link);
    vTaskDelay(UART_AFTER_SWITCH_OFF_TRANSMIT_IN_MS);
}

```

```

void    uart_receive_stop (UART_LINK_NUMBER_t link)
{constuart_init_t* u;

```

```

    u = &UART_INIT[link];
    DMA_stop (u->dma_rx);

```

```

    usart_interrupt_enable (u->usart_x, USART_IDLE_INT, FALSE);
    usart_dma_receiver_enable (u->usart_x, FALSE);
}

```

```

void    uart_receive_start (UART_LINK_NUMBER_t link)
{constuart_init_t* u;

```

```

    u = &UART_INIT[link];
    DMA_start (u->dma_rx);
    usart_interrupt_enable (u->usart_x, USART_IDLE_INT, TRUE);
    usart_dma_receiver_enable (u->usart_x, TRUE);
}

```

```

void    uart_receive_reinit_dma (UART_LINK_NUMBER_t link, uint8_t*
dst)
{constuart_init_t* u;

```

```

    u = &UART_INIT[link];
    DMA_init_transaction    (u->dma_rx,    (uint32_t)&(u->usart_x->dt),
(uint32_t)dst, u->rx_size);
}

```

```

void uart_receive_set_queue(UART_LINK_NUMBER_t link, uint8_t* buf,
uint16_t size)
{
    xMessageBufferSendFromISR(uart_handle[link],    buf,    size,
(BaseType_t*)1);
}

```

```

void    uart_receive_complite (UART_LINK_NUMBER_t link)
{ usart_type* u;
  uint16_t  i;
  uint16_t  s;

```

```

    u = UART_INIT[link].usart_x;
    if(SET == usart_interrupt_flag_get(u, USART_IDLEF_FLAG)) {
        uart_receive_stop (link);
        usart_flag_clear  (u, USART_IDLEF_FLAG);
        s = UART_INIT[link].rx_size;
        i = s - DMA_receive_size(UART_INIT[link].dma_rx);
        if (i) uart_receive_set_queue  (link, &uart_rxd_data[link], i);
        uart_receive_reinit_dma  (link, &uart_rxd_data[link]);
    }
    uart_counter_rxd [link]++;
}

```

```

uint16_t uart_rxd_message_waiting (UART_LINK_NUMBER_t link)

```

```

{constuart_init_t* u;

    u = &UART_INIT[link];
    return xMessageBufferSpacesAvailable(uart_handle[link]);
}

void UART_init (void)
{uint32_t i;
constuart_init_t* u;
MessageBufferHandle_t h;

    for (i = 0; i< UART_MAX_NUMBER; i++) {
        u = &UART_INIT[i];
        usart_init(u->usart_x,          u->baud_rate,          USART_DATA_8BITS,
USART_STOP_1_BIT);
        usart_transmitter_enable (u->usart_x, TRUE);
        usart_receiver_enable (u->usart_x, TRUE);
        usart_parity_selection_config (u->usart_x, USART_PARITY_NONE);
        usart_hardware_flow_control_set(u->usart_x,
USART_HARDWARE_FLOW_NONE);
        usart_interrupt_enable (u->usart_x, USART_IDLE_INT, TRUE);

        usart_dma_transmitter_enable (u->usart_x, TRUE);
        usart_dma_receiver_enable (u->usart_x, TRUE);

        usart_enable(u->usart_x, TRUE);

        uart_handle[i] = xMessageBufferCreate(u->rx_size);

        uart_counter_txd [i] = 0;

```

```

uart_counter_rxd [i] = 0;
    }
}

uint8_t uart_put_pkt (UART_LINK_NUMBER_t link, uint8_t* src)
{uint16_t t;
  uint32_t addr, size;
  constuart_init_t* u;

  u = &UART_INIT[link];
  vTaskDelay(UART_AFTER_SWITCH_ON_TRANSMIT_IN_MS);
  addr = (uint32_t)&src[1];
  size = (uint32_t)src[0];
  if (DMA_NULL != u->dma_tx) {
    DMA_stop (u->dma_tx);
    DMA_init_transaction (u->dma_tx, (uint32_t)&(u->usart_x->dt), addr,
size);
    DMA_start (u->dma_tx);
  }
  vTaskDelay(UART_BEFORE_SWITCH_OFF_TRANSMIT_IN_MS);
  uart_counter_txd[link]++;
  return t;
}

void uart_init_wait_rxd_pkt (UART_LINK_NUMBER_t link, uint8_t*
dst)
{constuart_init_t* u;

  u = &UART_INIT[link];
  if (DMA_NULL != u->dma_rx) {

```

```

DMA_init_transaction      (u->dma_rx,      (uint32_t)&(u->usart_x->dt),
(uint32_t)dst, u->rx_size);
DMA_start  (u->dma_rx);
}
}

```

```

uint8_t* uart_get_pkt  (UART_LINK_NUMBER_t link, uint16_t* size)
{uint16_t t;

```

```

    *size = uart_rxd_message_waiting(link);
    xMessageBufferReceive(uart_handle[link], uart_rxd_data, *size, 0xFFFF);
    return &uart_rxd_data [link];
}

```

```

void      USART1_IRQHandler      (void)      {uart_receive_complite
(UART_LINK_0);}
void      UART7_IRQHandler      (void)      {uart_receive_complite
(UART_LINK_1);}

```

В файле *DMA*.спредставлены функции, обеспечивающие работу каналов *DMA* для интерфейса *UART*.

В функции *DMA_irq_init* происходит инициализация прерываний каналов *DMA*.

В функции *DMA_channel_init* происходит инициализация каналов *DMA*: задается режим работы.

В функции *DMA_stop* происходит остановка канала *DMA*.

В функции *DMA_start* происходит включение канала *DMA*.

В функции *DMA_init_transaction* происходит инициализация передачи данных по каналу *DMA*.

В функции *DMA_receive_size* происходит вычисление количества принятых данных по каналу *DMA*.

Функция *DMA_init* инициализирует каналы *DMA*.

Вфункции*DMA_channel_irq_txd* происходит обработка прерываний канала *DMA*, работающего на передачу.

Вфункции*DMA_channel_irq_rxd* происходит обработка прерываний канала *DMA*, работающего на прием.

Ниже представлено описание файла *DMA.c*:

```
/*-----  
* Name:   DMA.c  
* Purpose: AT32F407VE (100-pin) Direct Memory Access init  
* Version: V1.0  
* Date:   30.01.2024  
* Note(s):  
*-----*/
```

```
#include "DMA.h"
```

```
#include "at32f403a_407_conf.h"
```

```
#include "stdio.h"
```

```
#include "string.h"
```

```
#include "../Hardware/UART.h"    /* USARTs init */
```

```
typedef struct {  
    dma_channel_type*    dma_x;  
    dma_dir_type dir;  
    dma_priority_level_type    priority;  
    confirm_state    loop;  
    dma_type*    type;  
    uint8_t    flex;  
    dma_flexible_request_type    request;
```

```
IRQn_Type irq;
```

```
UART_LINK_NUMBER_t uart;
```

```
} dma_init_t;
```

```
const dma_init_t DMA_INIT [] = {  
    {DMA1_CHANNEL1,      DMA_DIR_MEMORY_TO_PERIPHERAL,  
DMA_PRIORITY_LOW,      FALSE,      DMA1,      FLEX_CHANNEL1,  
DMA_FLEXIBLE_UART1_TX, DMA1_Channel1_IRQn, UART_LINK_0},  
    {DMA1_CHANNEL2,      DMA_DIR_PERIPHERAL_TO_MEMORY,  
DMA_PRIORITY_LOW,      FALSE,      DMA1,      FLEX_CHANNEL2,  
DMA_FLEXIBLE_UART1_RX, DMA1_Channel2_IRQn, UART_LINK_0},  
    {DMA1_CHANNEL3,      DMA_DIR_MEMORY_TO_PERIPHERAL,  
DMA_PRIORITY_LOW,      FALSE,      DMA1,      FLEX_CHANNEL3,  
DMA_FLEXIBLE_UART7_TX, DMA1_Channel3_IRQn, UART_LINK_1},  
    {DMA1_CHANNEL4,      DMA_DIR_PERIPHERAL_TO_MEMORY,  
DMA_PRIORITY_LOW,      FALSE,      DMA1,      FLEX_CHANNEL4,  
DMA_FLEXIBLE_UART7_RX, DMA1_Channel4_IRQn, UART_LINK_1}  
};
```

```
#define DMA_MAX_NUMBER (sizeof(DMA_INIT)/sizeof(dma_init_t))
```

```
uint32_t dma_counter_txd[DMA_MAX_NUMBER];
```

```
uint32_t dma_counter_rxd[DMA_MAX_NUMBER];
```

```
void DMA_irq_init(void)
```

```
{ const dma_init_t* d;
```

```
int8_t      i;
```

```
for (i = 0; i < DMA_MAX_NUMBER; i++) {
```

```
    d = &DMA_INIT[i];
```

```

nvic_irq_enable(d->irq, 0, 0);
}
}

```

```

void DMA_channel_init(DMA_LINK_NUMBER_t link)
{ constdma_init_t* d;
  dma_init_type      s;

```

```

    d = &DMA_INIT[link];
    dma_reset(d->dma_x);
    dma_default_para_init(&s);

```

```

    s.direction      = d->dir;
    s.memory_data_width = DMA_MEMORY_DATA_WIDTH_BYTE;
    s.memory_inc_enable = TRUE;
    s.peripheral_data_width = DMA_PERIPHERAL_DATA_WIDTH_BYTE;
    s.peripheral_inc_enable = FALSE;
    s.priority        = d->priority;
    s.loop_mode_enable = d->loop;
    dma_init(d->dma_x, &s);
    dma_flexible_config(d->type, d->flex, d->request);
    dma_interrupt_enable(d->dma_x, DMA_FDT_INT, TRUE);
}

```

```

void DMA_stop(DMA_LINK_NUMBER_t link)
{dma_channel_enable((&DMA_INIT[link])->dma_x, FALSE);}

void DMA_start(DMA_LINK_NUMBER_t link)
{dma_channel_enable((&DMA_INIT[link])->dma_x, TRUE);}

void DMA_init_transaction(DMA_LINK_NUMBER_t link, uint32_t
peripheral, uint32_t memory, uint16_t size)

```

```

{constdma_init_t* i;

i = &DMA_INIT[link];
i->dma_x->dtcnt = size;
i->dma_x->paddr = peripheral;
i->dma_x->maddr = memory;
}

uint16_t DMA_receive_size(DMA_LINK_NUMBER_t link) {return
dma_data_number_get(DMA_INIT[link].dma_x);}

void DMA_init(void)
{uint8_t i;

for (i = 0; i < DMA_MAX_NUMBER; i++)
DMA_channel_init((DMA_LINK_NUMBER_t)i);
memset(dma_counter_txd,0x00, sizeof(dma_counter_txd));
memset(dma_counter_rxd,0x00, sizeof(dma_counter_rxd));
}

void DMA_channel_irq_txd (DMA_LINK_NUMBER_t link, uint32_t irq)
{UART_LINK_NUMBER_tuart;

uart = DMA_INIT[link].uart;
if(dma_interrupt_flag_get(irq)) {
dma_flag_clear(irq);
dma_channel_enable(DMA_INIT[link].dma_x, FALSE);
}
dma_counter_txd[link]++;
}

```

```

void DMA_channel_irq_rxd (DMA_LINK_NUMBER_t link, uint32_t irq)
{
    const dma_init_t* i;
    uint32_t peripheral;
    uint32_t memory;
    uint16_t size;

    i = &DMA_INIT[link];
    if(dma_interrupt_flag_get(irq)) {
        dma_flag_clear(irq);
    }
    dma_counter_rxd[link]++;
}

```

```

void DMA1_Channel1_IRQHandler (void)
{DMA_channel_irq_txd(DMA_UART1_TXD_0, DMA1_FDT1_FLAG);}

void DMA1_Channel2_IRQHandler (void)
{DMA_channel_irq_rxd(DMA_UART1_RXD_1, DMA1_FDT2_FLAG);}

void DMA1_Channel3_IRQHandler (void)
{DMA_channel_irq_txd(DMA_UART7_TXD_2, DMA1_FDT3_FLAG);}

void DMA1_Channel4_IRQHandler (void)
{DMA_channel_irq_rxd(DMA_UART7_RXD_3, DMA1_FDT4_FLAG);}

```

В файле *KB_LED*.спредставлены функции для отображения времени зарядки/разрядки, режимов работы устройства.

В функции *kb_set_time* происходит установка времени (количества минут) зарядки/разрядки устройства.

В функции *kb_set_time_hour* происходит установка времени (количества часов) зарядки/разрядки устройства.

В функции *kb_set_charge* происходит отображения уровня оставшегося заряда устройства.

В функции *kb_set_mode* происходит отображение показания зарядки/разрядки устройства.

В функции *kb_set_io_220* происходит отображение режима работы устройства: работа от аккумулятора либо работа от внешнего источника питания.

В функции *KB_LED_set_code* происходит отображение времени зарядки/разрядки устройства.

В функции *KB_LED_set_charge* происходит установка уровня оставшегося заряда устройства.

В функции *KB_LED_set_mode* происходит установка показания зарядки/разрядки устройства.

В функции *KB_LED_set_io_220* происходит установка режима работы устройства: работа от аккумулятора либо работа от внешнего источника питания.

В функции *KB_LED_init* происходит инициализация начального состояния отображения устройства.

В функции *KB_LED_main* происходит описание алгоритма работы задачи по отображению данных.

Ниже представлено описание файла *KB_LED.c*:

```
/*-----
```

```
* Name: KB_LED.c
```

```
* Purpose: AT32F407VE (100-pin) Keyboard and Light-Emitting Diode functions
```

```
* Version: V1.0
```

```
* Date: 25.01.2024
```

```
* Note(s):
```

```
*-----*/
```

```
#include "KB_LED.h"
```

```
#include "CONTROL.h"
```

```

#include "../Hardware/GPIO.h"      /* Ports init          */

#include "FreeRTOS.h"
#include "task.h"
#include "stdio.h"
#include "string.h"

#define KB_LED_BLINK_DURATION_IN_MS    5

typedef struct {
    uint16_t pins;
    gpio_type* port;
} led_gpio_t;

typedef enum {
    CODE_A = 0x80,
    CODE_B = 0x40,
    CODE_C = 0x20,
    CODE_D = 0x10,
    CODE_E = 0x08,
    CODE_F = 0x04,
    CODE_G = 0x02
} PIN_CODE_t;

typedef enum {
    SEG_HOUR   = 2,
    SEG_MIN_10 = 1,
    SEG_MIN_1  = 0
} PIN_SEG_t;

```

```

#define PIN_CODE_SIZE 8
#define PIN_SEG_SIZE 3
#define PIN_LED_SIZE 8

#define PIN_A_DATA {PIN_A, PORT_A}
#define PIN_B_DATA {PIN_B, PORT_B}
#define PIN_C_DATA {PIN_C, PORT_C}
#define PIN_D_DATA {PIN_D, PORT_D}
#define PIN_E_DATA {PIN_E, PORT_E}
#define PIN_F_DATA {PIN_F, PORT_F}
#define PIN_G_DATA {PIN_G, PORT_G}

#define PIN_SEG_100 {PIN_VT1, PORT_VT1}
#define PIN_SEG_10 {PIN_VT2, PORT_VT2}
#define PIN_SEG_1 {PIN_VT3, PORT_VT3}

#define PIN_CHARGE_100 {PIN_HL12, PORT_HL12}
#define PIN_CHARGE_090 {PIN_HL6, PORT_HL6}
#define PIN_CHARGE_080 {PIN_HL11, PORT_HL11}
#define PIN_CHARGE_065 {PIN_HL10, PORT_HL10}
#define PIN_CHARGE_050 {PIN_HL4, PORT_HL4}
#define PIN_CHARGE_035 {PIN_HL9, PORT_HL9}
#define PIN_CHARGE_020 {PIN_HL5, PORT_HL5}
#define PIN_CHARGE_010 {PIN_HL3, PORT_HL3}

#define PIN_MODE_CHARGE {PIN_HL1, PORT_HL1}
#define PIN_MODE_DISCHARGE {PIN_HL2, PORT_HL2}

#define PIN_IO_220_IN {PIN_HL7, PORT_HL7}

```

```

#define PIN_IO_220_OUT    {PIN_HL8, PORT_HL8}

#define LED_0  (CODE_A | CODE_B | CODE_C | CODE_D | CODE_E |
CODE_F) // 0  (1111110)

#define LED_1  (      CODE_B | CODE_C
                                ) // 1
(0110000)

#define LED_2  (CODE_A | CODE_B |      CODE_D | CODE_E |
CODE_G      ) // 2  (1101101)

#define LED_3  (CODE_A | CODE_B | CODE_C | CODE_D |
CODE_G      ) // 3  (1111001)

#define LED_4  (      CODE_B | CODE_C |      CODE_G |
CODE_F) // 4  (0110011)

#define LED_5  (CODE_A      | CODE_C | CODE_D |      CODE_G |
CODE_F) // 5  (1011011)

#define LED_6  (CODE_A      | CODE_C | CODE_D | CODE_E |
CODE_G | CODE_F) // 6  (1011111)

#define LED_7  (CODE_A | CODE_B | CODE_C
                                ) //
7  (1110000)

#define LED_8  (CODE_A | CODE_B | CODE_C | CODE_D | CODE_E |
CODE_G | CODE_F) // 8  (1111111)

#define LED_9  (CODE_A | CODE_B | CODE_C | CODE_D |
CODE_G | CODE_F) // 9  (1111011)

#define LED_OFF 0x00
                                // off
(0000000)

uint8_t kb_led_time[PIN_SEG_SIZE];
uint8_t kb_led_time_point;
uint8_t kb_led_charge;
uint8_t kb_led_mode;
uint8_t kb_led_io_220;

```

```

uint8_t kb_led_cycle;

constled_gpio_t KB_LED_PIN_CODE [] =
{PIN_G_DATA, PIN_F_DATA, PIN_E_DATA, PIN_D_DATA,
PIN_C_DATA, PIN_B_DATA, PIN_A_DATA};

constled_gpio_t KB_LED_SEG_CODE [] =
{PIN_SEG_100, PIN_SEG_10, PIN_SEG_1};

constled_gpio_t KB_LED_CHARGE_CODE [] =
{PIN_CHARGE_100, PIN_CHARGE_090, PIN_CHARGE_080,
PIN_CHARGE_065, PIN_CHARGE_050, PIN_CHARGE_035,
PIN_CHARGE_020, PIN_CHARGE_010};

constled_gpio_t KB_LED_MODE_CODE [] =
{PIN_MODE_CHARGE, PIN_MODE_DISCHARGE};

constled_gpio_t KB_LED_IO_CODE [] =
{PIN_IO_220_IN, PIN_IO_220_OUT};

const uint8_t KB_LED_DIGIT_CODE [] =
{LED_0, LED_1, LED_2, LED_3, LED_4, LED_5, LED_6, LED_7,
LED_8, LED_9,
CODE_A, CODE_B, CODE_C, CODE_D, CODE_E, CODE_F,
CODE_G};

#define LED_ALL_OFF (sizeof(KB_LED_DIGIT_CODE))

void kb_clear (void) {KB_LED_init();}

void kb_set_time (uint16_t minutes)
{uint8_t h,m;

if (!minutes) {
memset(kb_led_time, 0x00, PIN_SEG_SIZE);

```

```

        return;
    }
    h = minutes / 60;
    kb_led_time[SEG_HOUR] = h;
    m = minutes - 60 * h;
    h = m / 10;
    kb_led_time[SEG_MIN_10] = h;
    m -= (10 * h);
    kb_led_time[SEG_MIN_1] = m;
}

void kb_set_time_hour (uint16_t hour)
{uint8_t h,m;

    if (!hour) {
memset(kb_led_time, 0x00, PIN_SEG_SIZE);
        return;
    }
    h = hour / 60;
    m = h / 100;
    kb_led_time[SEG_HOUR] = m;
    h -= (m * 100);
    m = h / 10;
    kb_led_time[SEG_MIN_10] = m;
    h -= (m * 10);
    m = h;
    kb_led_time[SEG_MIN_1] = m;
}

```

```

const uint8_t KB_SET_CHRAGE_MASK[] = {0x00, 0x80, 0x40, 0x20,
0x10, 0x18, 0x1C, 0x1E, 0x1F};

void kb_set_charge (KB_LED_CHARGE_LEVEL_t value) {kb_led_charge
= KB_SET_CHRAGE_MASK[value];}

void kb_set_mode (KB_LED_MODE_t value) { kb_led_mode =
value;}

void kb_set_io_220 (KB_LED_IO_220_t value) { kb_led_io_220 =
value;}

void KB_LED_set_code(uint8_t value, uint8_t position)
{uint8_t t;
uint8_t i;
uint8_t mask;

if (LED_ALL_OFF == value) {
for (i = 0; i< PIN_CODE_SIZE; i++) gpio_bits_reset
(KB_LED_PIN_CODE[i].port, KB_LED_PIN_CODE[i].pins);
return;
}

if (SEG_HOUR == position) {if (CONTROL_get_seg_flag())
gpio_bits_set (PORT_P, PIN_P);
else
gpio_bits_reset (PORT_P, PIN_P);}
else
gpio_bits_reset (PORT_P, PIN_P);
t = KB_LED_DIGIT_CODE [value];
mask = CODE_G;
for (i = 0; i< PIN_CODE_SIZE; i++) {
if (t & mask) gpio_bits_set (KB_LED_PIN_CODE[i].port,
KB_LED_PIN_CODE[i].pins);

```

```

        else        gpio_bits_reset        (KB_LED_PIN_CODE[i].port,
KB_LED_PIN_CODE[i].pins);
        mask <=<= 1;
    }
    for (i = 0; i< PIN_SEG_SIZE; i++) {
        if (i == position) gpio_bits_set        (KB_LED_SEG_CODE[i].port,
KB_LED_SEG_CODE[i].pins);
        else        gpio_bits_reset        (KB_LED_SEG_CODE[i].port,
KB_LED_SEG_CODE[i].pins);
    }
}

```

```

void KB_LED_set_charge (uint8_t value)

```

```

{ uint8_t i;

```

```

uint8_t mask;

```

```

    for (i = 0; i< PIN_LED_SIZE; i++)        gpio_bits_reset
(KB_LED_CHARGE_CODE[i].port, KB_LED_CHARGE_CODE[i].pins);

```

```

    mask = CL_010;

```

```

    for (i = 0; i< PIN_LED_SIZE; i++) {
        if        (mask        &kb_led_charge)        gpio_bits_set
(KB_LED_CHARGE_CODE[i].port, KB_LED_CHARGE_CODE[i].pins);
        mask <=<= 1;
    }
}

```

```

void KB_LED_set_mode (uint8_t value) {

```

```

    gpio_bits_reset        (KB_LED_MODE_CODE[0].port,
KB_LED_MODE_CODE[0].pins);

```

```

        gpio_bits_reset                                (KB_LED_MODE_CODE[1].port,
KB_LED_MODE_CODE[1].pins);

```

```

        if (MODE_CHARGE == kb_led_mode)    gpio_bits_set
(KB_LED_MODE_CODE[0].port, KB_LED_MODE_CODE[0].pins);
        if (MODE_DISCHARGE == kb_led_mode)    gpio_bits_set
(KB_LED_MODE_CODE[1].port, KB_LED_MODE_CODE[1].pins);
    }

```

```

void KB_LED_set_io_220 (uint8_t value) {
    gpio_bits_reset                                (KB_LED_IO_CODE[0].port,
KB_LED_IO_CODE[0].pins);
    gpio_bits_reset                                (KB_LED_IO_CODE[1].port,
KB_LED_IO_CODE[1].pins);

```

```

        if (IO_220_IN == kb_led_io_220)                gpio_bits_set
(KB_LED_IO_CODE[0].port, KB_LED_IO_CODE[0].pins);
        if (IO_220_OUT == kb_led_io_220)                gpio_bits_set
(KB_LED_IO_CODE[1].port, KB_LED_IO_CODE[1].pins);
        if (IO_220_IN_OUT == kb_led_io_220)    {gpio_bits_set
(KB_LED_IO_CODE[0].port, KB_LED_IO_CODE[0].pins);
            gpio_bits_set                                (KB_LED_IO_CODE[1].port,
KB_LED_IO_CODE[1].pins);}
    }

```

```

void KB_LED_init(void) {
    memset(kb_led_time, 0x00, PIN_SEG_SIZE);
    kb_led_time_point = LED_OFF;
    kb_led_cycle = SEG_MIN_1;
    kb_led_charge = CL_EMPTY;

```

```

kb_led_mode    = MODE_UNACTIVE;
kb_led_io_220  = IO_220_UNACTIVE;
}

void KB_LED_main(void* params) {

while(1)
{
vTaskDelay(KB_LED_BLINK_DURATION_IN_MS);
KB_LED_set_code(kb_led_time[kb_led_cycle], kb_led_cycle);
KB_LED_set_charge (kb_led_charge);
KB_LED_set_mode  (kb_led_mode);
    KB_LED_set_io_220 (kb_led_io_220);
kb_led_cycle++;
    if (!(kb_led_cycle< PIN_SEG_SIZE)) kb_led_cycle = 0;
}
}

```

В файле *INVERTER*.сописаны функции для работы с платой инвертора.

В функции *INVERTER_main*происходит описание алгоритма работы задачи по работе с платой инвертора.

Ниже представлено описание файла *INVERTER.c*:

```

/*-----
* Name:   INVERTER.c
* Purpose: AT32F407VE (100-pin) Host connection functions
* Version: V1.0
* Date:   25.01.2024
* Note(s):
*-----*/

```

```

#include "INVERTER.h"

```

```

#include "../Hardware/GPIO.h"      /* GPIO function      */
#include "../Hardware/UART.h"      /* UART function      */

#include "FreeRTOS.h"
#include "task.h"
#include "stdio.h"

#define INVERTER_UART_LINK          UART_LINK_0

#define INVERTER_BUF_SIZE          16

const uint8_t INVERTER_ON[] = {0xFF, 0x08, 0x01, 0x00, 0x00, 0x11,
0x34, 0x25};
const uint8_t INVERTER_OFF[] = {0xFF, 0x08, 0x01, 0x00, 0x00, 0x13,
0xB5, 0xE4};

uint8_t inverter_buf[INVERTER_BUF_SIZE];

void INVERTER_init(void)
{uint8_t i;
memset(&inverter_buf[0], 0x00, INVERTER_BUF_SIZE);
}

void INVERTER_main(void* params)
{

vTaskDelay(100);
uart_switch_on_receive (INVERTER_UART_LINK);
while(1) {

```

```

vTaskDelay(1);

if (((GPIO_read_input_pin(PORT_INV_BUT, PIN_INV_BUT)) ||
(CONTROL_charge_level() > CL_EMPTY)) &&
(!GPIO_read_output_pin(PORT_INV_VKL, PIN_INV_VKL))) {
    GPIO_set_pin(PORT_INV_VKL, PIN_INV_VKL);
    GPIO_set_pin(PORT_BUZZER, PIN_BUZZER);
    vTaskDelay(100);
    uart_put_pkt(INVERTER_UART_LINK, INVERTER_ON);
    GPIO_reset_pin(PORT_BUZZER, PIN_BUZZER);
}

if (((!GPIO_read_input_pin(PORT_INV_BUT, PIN_INV_BUT) ||
(CONTROL_charge_level() == CL_EMPTY)) &&
(GPIO_read_output_pin(PORT_INV_VKL, PIN_INV_VKL))) {
    uart_put_pkt(INVERTER_UART_LINK, INVERTER_OFF);
    GPIO_set_pin(PORT_BUZZER, PIN_BUZZER);
    vTaskDelay(100);
    GPIO_reset_pin(PORT_INV_VKL, PIN_INV_VKL);
    GPIO_reset_pin(PORT_BUZZER, PIN_BUZZER);
}
}
}

```

В файле *CONTROL*.спредставлены функции для обработки принятых данных с платы BMS.

В функции *CONTROL_charge_level* происходит обработка уровня оставшегося заряда устройства.

В функции *CONTROL_init* происходит установка начального состояния устройства.

В функции *CONTROL_time* происходит обработка времени заряда/разряда устройства.

В функции *CONTROL_get_c_io_220* происходит обработка режима работы устройства: работа от аккумулятора либо работа от внешнего источника питания.

В функции *CONTROL_get_c_mode* происходит обработка показания зарядки/разрядки устройства.

В функции *CONTROL_main* происходит описание алгоритма работы задачи по обработке входящих данных с платы *BMS*.

Ниже представлено описание файла *CONTROL.c*:

```
/*-----  
* Name:   CONTROL.c  
* Purpose: AT32F407VE (100-pin) Main control functions  
* Version: V1.0  
* Date:   25.01.2024  
* Note(s):  
*-----*/  
  
#include "CONTROL.h"  
#include "KB_LED.h"  
#include "INVERTER.h"  
#include "BMS.h"  
  
#include "../Hardware/GPIO.h"    /* Ports init */  
  
#include "FreeRTOS.h"  
#include "task.h"  
#include "stdio.h"  
  
#define initial_capacity  200  
  
uint16_t      c_minutes;
```

```

KB_LED_CHARGE_LEVEL_tc_charge;
KB_LED_MODE_tc_mode;
KB_LED_IO_220_t      c_io_220;


int16_t      c_current;
int16_t      c_voltage;
int16_t      c_current_array[64];
int16_t      c_current_average;


uint8_t      seg_flag;
uint8_t      cnt_current = 0;


KB_LED_CHARGE_LEVEL_tCONTROL_charge_level(void)
{int16_t c;


    c = BMS_charge_level ();
    c  = (((c << 8) & 0xff00) | ((c >> 8) & 0x00ff));


    if (c > 99)      return CL_100;
    if (c > 84)      return CL_090;
    if (c > 69)      return CL_080;
    if (c > 54)      return CL_065;
    if (c > 39)      return CL_050;
    if (c > 24)      return CL_035;
    if (c > 14)      return CL_020;
    if (c > 9)       return CL_010;
    return CL_EMPTY;
}


void CONTROL_init(void) {

```

```

uint8_t i;
c_minutes = 0;
c_charge = CL_EMPTY;
c_mode = MODE_UNACTIVE;
c_io_220 = IO_220_UNACTIVE;
c_current = 0;
for (i = 0; i < 64; i++)
{
c_current_array[i] = 1;
}
}

void CONTROL_time (void)
{int16_t charge_lvl, current_capacity, time, sum;
uint8_t i;
charge_lvl =BMS_charge_level ();
c_current = BMS_output_current();
c_current = (((c_current<< 8) & 0xff00) | ((c_current>> 8) & 0x00ff));

current_capacity = (charge_lvl>> 8) * initial_capacity / 100;

if(cnt_current++ > 64) cnt_current = 0;
c_current_array[cnt_current] = c_current;

sum = 0;
for (i = 0; i < 32; i++)
{
sum += c_current_array[i];
}
}

```

```

c_current = sum / 32;

if (c_current < 11) c_current = 10;

switch(c_mode){
    case MODE_DISCHARGE:    time = 9 *
current_capacity / c_current * 6; break;
    case MODE_CHARGE:      time = 14 * (initial_capacity -
current_capacity) / c_current * 6; break;
    default:                time = 0;
break;
}

if(time > 600) {kb_set_time_hour(time); seg_flag = 0;}
else {kb_set_time (time); seg_flag = 1;}
}

KB_LED_IO_220_t CONTROL_get_c_io_220(void)
{
    if(BMS_get_info_charge()) return IO_220_OUT;
    return IO_220_IN_OUT;
}

KB_LED_MODE_t CONTROL_get_c_mode(void)
{
    if(BMS_get_info_charge()) return MODE_DISCHARGE;
    return MODE_CHARGE;
}

uint8_t CONTROL_get_seg_flag(void)
{

```

```

    return seg_flag;
}

void CONTROL_main(void* params)
{
    while(1) {
        vTaskDelay (100);
        kb_set_charge (c_charge);
        kb_set_mode (c_mode);
        kb_set_io_220 (c_io_220);

        c_charge =CONTROL_charge_level ();
        c_io_220 = CONTROL_get_c_io_220 ();
        c_mode = CONTROL_get_c_mode ();

        CONTROL_time();
    }
}

```

В файле *BMS*.сописаны функции для работы с платой *BMS*.

Вфункции*BMS_pkt_read* происходит считывание данных, приходящих с платы *BMS*.

Вфункции*BMS_get_battery_voltage_24V* происходит получение данных об напряжении на аккумуляторе.

Вфункции*BMS_output_current* происходит получение данных об выходном токе из аккумулятора.

Вфункции*BMS_charge_level* происходит получение данных об уровне заряда аккумулятора.

Вфункции*BMS_get_info_charge* происходит получение данных об показании зарядки/разрядки устройства.

В функции *BMS_main* происходит описание алгоритма работы задачи получения данных с платы *BMS*.

Ниже представлено описание файла *BMS.c*:

```
/*-----  
* Name:   BMS.c  
* Purpose: AT32F407VE (100-pin) Battery Management System functions  
* Version: V1.0  
* Date:   25.01.2024  
* Note(s):  
*-----*/  
  
#include "BMS.h"  
  
#include "../Hardware/GPIO.h"    /* GPIO function    */  
#include "../Hardware/UART.h"    /* UART function    */  
  
#include "FreeRTOS.h"  
#include "task.h"  
#include "stdio.h"  
#include "string.h"  
  
#define BMS_UART_LINK            UART_LINK_1  
  
#define BMS_MAX_FRAME_SIZE      512  
  
#define BMS_SCAN_PERIODE_IN_MS  100  
  
uint8_t bms_rxd[BMS_MAX_FRAME_SIZE];  
uint16_t bms_success_cnt;
```

```

typedef enum {
    BMS_BATTERY_VOLT_24V = 3,
    BMS_OUTPUT_CURRENT    = 4,
    BMS_CHARGE_LEVEL      = 6
}BMS_DATA_t;

```

```

typedef struct {
    uint16_t headline;
    int16_t a01;
    int16_t a02;
    int16_t battery_volt_24V;
    uint16_t output_current;
    int16_t a05;
    int16_t charge_level;
    int16_t a07;
    int16_t a08;
    int16_t a09;
    int16_t a0A;
    int16_t a0B;
    int16_t a0C;
    int16_t a0D;
    int16_t a0E;
    int16_t a0F;
    int16_t a10;
    int16_t a11;
    int16_t power;
}bms_read_answer_struct_t;

typedef union {
    uint8_t buf[64];
}bms_read_answer_struct_t data;

```

```

    }bms_read_answer_t;

    bms_read_answer_tbms_read_answer;

    int16_t      BMS_get_battery_voltage_24V      (void)      {return
bms_read_answer.data.battery_volt_24V;}

    int16_t BMS_output_current (void)
    {
        if((uint8_t)bms_read_answer.data.output_current> 0x50) return 0xFFFF -
bms_read_answer.data.output_current;
        return bms_read_answer.data.output_current;
    }

    int16_t      BMS_charge_level                  (void)      {return
bms_read_answer.data.charge_level;}

    uint8_t BMS_get_info_charge (void)
    {
        if((uint8_t)bms_read_answer.data.output_current> 0x50) return 1;
        return 0;
    }

    uint16_t BMS_headline (void) {return bms_read_answer.data.headline;}

    uint16_t BMS_pkt_read(uint8_t* buf)
    {uint8_t* ptr;
    uint16_t size;
    size = 0;
    if (uart_rxd_message_waiting (BMS_UART_LINK)) {
    ptr =uart_get_pkt (BMS_UART_LINK, &size);
    memcpy(&buf[index], ptr, size);
    }
    uart_receive_reinit(BMS_UART_LINK);

```

```

    return size;
}

void BMS_init(void)
{uint8_t i;

memset(&bms_rxd[0], 0x00, BMS_MAX_FRAME_SIZE);
}

void BMS_main(void* params)
{uint8_t i = 0;
uart_switch_on_receive (BMS_UART_LINK);
while(1) {
bms_success_cnt = BMS_pkt_read(bms_read_answer.buf);
vTaskDelay(100);
}
}

```

4 Расчет общего искусственного освещения в помещении с ПЭВМ

Целью расчета является определение числа светильников, а также числа и мощности ламп, необходимых для обеспечения требуемого нормативного уровня освещенности для работы оператора ПЭВМ.

Расчет искусственного освещения проводится для помещения общей площадью $S = 16 \text{ м}^2$ (длина помещения $a = 4 \text{ м}$, ширина $b = 4 \text{ м}$, высота $h = 3 \text{ м}$).

В соответствии с СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к ПЭВМ и организации работы» освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300 – 500 лк. Для расчета требуемый нормативный уровень освещенности E_n был выбран 400 лк.

Для освещения помещения используется система общего освещения, так как она обеспечивает большую равномерность освещенности рабочих поверхностей.

Расчет общего равномерного освещения будет производиться методом коэффициента использования светового потока, учитывающий величину светового потока, отраженного от потолка и стен.

Для освещения данного помещения был выбран тип светильника ЛСП02-2х40 с люминесцентными лампами типа ЛБ40 (в количестве двух штук, мощностью по 40 Вт каждая и световым потоком $F = 3120 \text{ лм}$), габаритные размеры которого 158х280х1234 мм.

Количество светильников с люминесцентными лампами заданного типа и мощности определяется по формуле 4.1:

$$N = \frac{E_n * S * z * K_z}{F * N_l * n}, (4.1)$$

где: $E_n = 400 \text{ лк}$ – требуемый нормативный уровень освещенности;

$S = 16 \text{ м}^2$ – площадь помещения;

$z = 1,1$ – коэффициент неравномерности помещения;

$K_z = 1,5$ – коэффициент запаса, который учитывает запыленность светильников в процессе эксплуатации;

$F = 3120$ лм – световой поток одной люминесцентной лампы в светильнике;

$N_l = 2$ – количество ламп в одном светильнике;

$n = 0,46$ – коэффициент используемого излучаемого светильником светового потока при освещении рабочей поверхности.

Тогда количество светильников с люминесцентными лампами заданного типа и мощности равно:

$$N = \frac{400 * 16 * 1,1 * 1,5}{3120 * 2 * 0,46} = 3,68.$$

Таким образом, принимаем количество светильников равным 4, каждый из которых укомплектован двумя люминесцентными лампами.

Фактический уровень освещенности по формуле 4.2:

$$E_f = \frac{N * N_l * F * n}{S * z * K_z}. (4.1)$$

Тогда фактический уровень освещенности равен:

$$N = \frac{4 * 2 * 3120 * 0,46}{16 * 1,1 * 1,5} = 434,9 \text{ лк}.$$

Полученное фактическое значение отличается от принятого требуемого уровня освещенности на 34,9 лк, но соответствует норме СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к ПЭВМ и организации работы» [30].

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были достигнуты следующие результаты:

1. Выполнен аналитический обзор научно-технической литературы.
2. Разработана схема электрическая принципиальная на отечественных компонентах.
3. Разработан алгоритм работы для системы на кристалле.
4. Разработано описание встроенного приложения для устройства управления автономными источниками питания.

В работе был исследован вопрос реализации устройства управления на основе отечественной компонентной базе. Были подобраны отечественные компоненты для интерфейсов *RS-232* и *RS-485*, система на кристалле, световые индикаторы, генератор тактовой частоты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Tian, J., et al. (2020). "A Cloud-Based Battery Management System for Second-Life Lithium-Ion Batteries". *IEEE Transactions on Industrial Informatics*, 16(5), 2921-2932.
2. Piller, S., Perrin, M., & Jossen, A. (2001). "Methods for state-of-charge determination and their applications". *Journal of Power Sources*, 96(1), 113-120.
3. Jiang Z., Dougal R. A. A novel, digitally-controlled, portable photovoltaic power source // *Twentieth Annual IEEE Applied Power Electronics Conference and Exposition*, 2005. APEC 2005. – IEEE, 2005. – Т. 3. – С. 1797-1802.
4. Fry D. N. et al. *Compact portable electric power sources*. – Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 1997. – №. ORNL/TM-13360.
5. Zhang, C., & Wang, L. (2018). "Battery Management System for Lithium-Ion Batteries: A Comprehensive Review". *IEEE Access*, 6, 59865-59888.
6. Fedak W. et al. *The concept of autonomous power supply system fed with renewable energy sources* // *Journal of Sustainable Development of Energy, Water and Environment Systems*. – 2017. – Т. 5. – №. 4. – С. 579-589.
7. Электронный ресурс [Режим доступа]:
<https://www.jkbms.com/product/jk-bd6a24s6p/>
8. Электронный ресурс [Режим доступа]: <https://www.wztianmo.com/>
9. Электронный ресурс [Режим доступа]:
<https://www.arterychip.com/en/product/AT32F407.jsp>
10. Fang Y., Chen X. *Design and simulation of UART serial communication module based on VHDL* // *2011 3rd International Workshop on Intelligent Systems and Applications*. – IEEE, 2011. – С. 1-4.
11. Laddha N. R., Thakare A. P. *A review on serial communication by UART* // *International journal of advanced research in computer science and software engineering*. – 2013. – Т. 3. – №. 1.

12. Ali L. et al. *Design of a micro-UART for SoC application* // *Computers & Electrical Engineering*. – 2004. – Т. 30. – №. 4. – С. 257-268.

13. Gupta A. K. et al. *Design and implementation of high-speed universal asynchronous receiver and transmitter (UART)* // *2020 7th International Conference on Signal Processing and Integrated Networks (SPIN)*. – IEEE, 2020. – С. 295-300.

14. Электронный ресурс [Режим доступа]:
<https://www.cta.ru/articles/cta/obzory/promyshlennye-seti/124959/>

15. Recio R. et al. *A remote direct memory access protocol specification*. – 2007. – №. rfc5040.

16. Kim D., Managuli R., Kim Y. *Data cache and direct memory access in programming mediaprocessors* // *IEEE Micro*. – 2001. – Т. 21. – №. 4. – С. 33-42.

17. Ahmed M. A., Aljumah A., Ahmad M. G. *Design and implementation of a direct memory access controller for embedded applications* // *International Journal of Technology*. – 2019. – Т. 10. – №. 2. – С. 309-319.

18. Электронный ресурс [Режим доступа]:
<https://diversedaily.com/understanding-dma-controllers-facilitating-direct-data-transfers-between-memory-and-peripherals/>

19. Tamboli S. et al. *Implementation of Modbus RTU and Modbus TCP communication using Siemens S7-1200 PLC for batch process* // *2015 international conference on smart technologies and management for computing, communication, controls, energy and materials (ICSTM)*. – IEEE, 2015. – С. 258-263.

20. Machacek J., Drapela J. *Control of serial port (RS-232) communication in LabVIEW* // *2008 International Conference-Modern Technique and Technologies*. – IEEE, 2008. – С. 36-40.

21. Электронный ресурс [Режим доступа]:
<https://приборэнерго.пф/press/interfeysy-rs-485-rs-232-i-rs-422/>

22. Kugelstadt T. *The RS-485 Design Guide: A short compendium for robust data transmission design* // *Texas Instruments: Dallas, TX, USA*. – 2008.

23. Электронный ресурс [Режим доступа]: <https://www.seeedstudio.com/blog/2022/09/08/uart-communication-protocol-and-how-it-works/?ysclid=mbgblzjff1205218640>
24. Barry R. et al. *FreeRTOS //Internet*, Oct. – 2008. – Т. 4. – С. 18.
25. Guan F. et al. *Open source FreeRTOS as a case study in real-time operating system evolution //Journal of Systems and Software*. – 2016. – Т. 118. – С. 19-35.
26. Nakano W., Shinohara Y., Ishiura N. *Full hardware implementation of FreeRTOS-based real-time systems //TENCON 2021-2021 IEEE Region 10 Conference (TENCON)*. – IEEE, 2021. – С. 435-440.
27. Электронный ресурс [Режим доступа]: https://www.securitylab.ru/blog/personal/xiaomite-journal/355349.php?utm_referrer=https%3A%2F%2Fya.ru%2F
28. Baraka J. K. *Determination of power consumption of a machine learning algorithm sunning on a microcontroller*. – 2024.
29. Электронный ресурс [Режим доступа]: https://arm-software.github.io/CMSIS_6/latest/General/index.html
30. СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к ПЭВМ и организации работы».

Приложение А
(обязательное)
Техническое задание на ВКР

Бланк задания на
выпускную квалификационную работу

«Утверждаю»
Заведующий кафедрой

(фамилия, имя, отчество)

« 22 » 05 2025 г.

З А Д А Н И Е

на выпускную квалификационную работу магистра

1. Студент гр. 23ПДм1 факультета ФИТЭ

Направление подготовки 11.04.04 «Электроника и нанoeлектроника»

Магистерская программа Нано- и микроэлектроника

Козин А.С.

(фамилия, инициалы)

2. Руководитель Головяшкин А.Н.

(фамилия, инициалы)

3. Тема выпускной работы Разработка устройства управления автономными источниками питания

Тема утверждена приказом ПГУ №588/о от « 28 » 04 2025 г.

1. Техническое задание на выпускную работу (разработать, исследовать (назначение объекта разработки, исследования), исходные данные, граничные условия разрабатываемого или исследуемого объекта, условия эксплуатации и т.д.)

Разработать устройство управления автономными источниками питания: входное напряжение 24 В постоянного напряжения, выходное напряжение 220 В переменного напряжения, зарядка от сети 220 В переменного напряжения, источник питания постоянного напряжения 24 В – литий-ионные батареи, отображение времени заряда/разряда источника питания, отображение уровня оставшегося заряда. Выполнить аналитический обзор научно-технической литературы. Разработать схемы электрические. Разработать алгоритм работы для системы на кристалле. Разработать описание встроенного приложения для устройства управления автономными источниками питания.

5. Объем и содержание основной части выпускной работы

5.1. Пояснительная записка (перечень вопросов, подлежащих разработке, обоснований, описаний, расчетов и т.д.)

1 Обзор современной литературы про устройства управления автономными источниками питания

1.1 Общие сведения про устройства управления автономными источниками питания

1.2 Обзор элементов, входящих в разработанную плату управления портативными источниками энергии

1.2 Обзор элементов, входящих в разработанную плату управления портативными источниками энергии

2 Обзор технологий, использованных в ходе разработки устройства управления автономными источниками питания

2.1 Обзор протокола *Modbus RTU* и последовательных интерфейсов

2.2 Обзор системы реального времени *FreeRTOS* и набора стандартных библиотек *CMSIS*

3 Разработка устройства управления автономными источниками питания

3.1 Разработка электрических схем для устройства управления автономными источниками питания

3.2 Алгоритм работы встроенного приложения

3.3 Разработка встроенного приложения для устройства управления автономными источниками питания

4 Расчет общего искусственного освещения в помещении с ПЭВМ

5.2. Иллюстративная часть (перечень и название слайдов презентации)

1. Титульный лист.

2. Цель работы

3. Структурная схема устройства управления автономными источниками питания.

4. Технические характеристики платы *BMSHKIVI-A06A24S-6P*.

5. Технические характеристики платы инвертора *TIANMOTPU1000W*.

6. Структурная схема платы управления автономными источниками питания.

7. Описание работы встроенного приложения.

8. Расчет общего искусственного освещения в помещении с ПЭВМ.

9. Заключение.

10. Список публикаций по теме ВКР.

11. Спасибо за внимание.

6. Консультанты и содержание дополнительных разделов

6.1. _____

Консультант _____

7. Календарный план по выполнению выпускной работы магистра

Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя, консультанта
Обзор современной литературы про устройства управления автономными источниками питания	15%	22.05.2025	
Обзор технологий, использованных в ходе разработке устройства управления автономными источниками питания	15%	24.05.2025	
Разработка устройства управления автономными источниками питания	50%	26.05.2025	
Расчет общего искусственного освещения в помещении с ПЭВМ	5%	1.06.2025	
Графическая часть	10%	2.06.2025	
Оформление пояснительной записки	5%	3.06.2025	

Дата выдачи задания «22» мая 2025 г.

Руководитель выпускной работы  подпись

Головяшкин А.Н.
фамилия, инициалы

Задание к исполнению принял  подпись

Козин А.С.
фамилия, инициалы

« 22 » мая 2025г.

Выпускную квалификационную работу к защите допустить

Декан факультета  подпись Креччик В.Д.
фамилия, инициалы

« 16 » 06 2025 г.

Приложение Б
(обязательное)
Схема электрическая структурная

Приложение В
(обязательное)
Схема электрическая функциональная

Приложение Г
(обязательное)
Схема электрическая принципиальная

Приложение Д
(обязательное)
Перечень компонентов

Приложение Е
(обязательное)
Список публикаций по теме ВКР

1. Головяшкин А.Н., Козин А. С. , Сухов И.В. Алгоритмы вычисления циклического избыточного кода // Труды международного симпозиума "Надежность и качество". – 2024. – Т. 1. – С. 421-422;

2. Головяшкин А.Н., Сухов И.В., Козин А. С., Баннов В.Я. Ускорение вычисления контрольной комбинации // Труды международного симпозиума "Надежность и качество". – 2024. – Т. 1. – С. 415-416.

3. И.Б. Мещеряков, И.В Сухов, М.Н. Рогулин, А.С.Козин, А.А. Фролов, В.Д.Гапоненко. Итеративная обработка кадровой последовательности, переданной *OFDM*-символами с канальным свёрточным кодированием и перемежение по спектру и времени//Радиолокация, навигация, связь: сборник трудов XXXI Международной научно-технической конференции (г. Воронеж, 15–17 апреля 2025 г.): в 6 т. / Воронежский государственный университет; АО «Концерн “Созвездие”». – Воронеж: Издательский дом ВГУ, 2025.–т.2. – с 313-322;

Приложение Ж

(обязательное)

Заявление и протокол проверки ВКР на оригинальность
в системе «Антиплагиат.ВУЗ»

Ректору Пензенского государственного
университета Гулякову А.Д.
от студента 2 курса ФИТЭ очной формы
обучения

Козина Алексея Сергеевича

ЗАЯВЛЕНИЕ

Я, Козин Алексей Сергеевич, студент 2 курса очной формы обучения направления подготовки 11.04.04 Электроника и наноэлектроника, профиль «Нано- и микроэлектроника», прошу провести проверку с использованием системы «Антиплагиат.ВУЗ» магистерской выпускной квалификационной работы на тему «Оптимизация параметров структуры лавинно-пролетных диодов», выполненной мной самостоятельно, на содержание элементов плагиата.

Все прямые заимствования из печатных и электронных источников, а также из защищенных ранее выпускных квалификационных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Я ознакомлен с действующим Стандартом университета «Выпускная квалификационная работа обучающихся по образовательным программам высшего образования - программам бакалавриата, программам специалитета и программам магистратуры», согласно которому обнаружение плагиата является основанием для недопуска ВКР к защите и отчисления из университета.



Подпись студента

ВКР представлена на проверку 09.06.2025

Руководитель ВКР  Головяшкин А.Н.

Протокол

Проверки на оригинальность в системе «Антиплагиат.ВУЗ» выпускной магистерской работы на тему «Разработка устройства управления автономными источниками питания»

студента 2 курса Козина А.С.

направления подготовки 11.04.04 Электроника и микроэлектроника, профиль «Нано- и микроэлектроника»

Руководитель ВКР Головяшкин А.Н.



СПРАВКА

о результатах проверки текстового документа на наличие заимствований

Федеральное государственное бюджетное образовательное учреждение высшего образования «Тюменский государственный университет»

ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Козин Алексей Сергеевич
Самодитирование: Козин Алексей Сергеевич
рассчитано для: Козин Алексей Сергеевич
Название работы: Разработка устройства управления автономными источниками питания
Тип работы: Магистерская диссертация
Подразделение: Кафедра «Нано- и микроэлектроника»

РЕЗУЛЬТАТЫ

■ ОТЧЕТ О ПРОВЕРКЕ КОРРЕКТИРОВАЛСЯ: НИЖЕ ПРЕДСТАВЛЕНЫ РЕЗУЛЬТАТЫ ПРОВЕРКИ ДО КОРРЕКТИРОВКИ

СОВПАДЕНИЯ	14.2%	СОВПАДЕНИЯ	14.2%
ОРИГИНАЛЬНОСТЬ	85.8%	ОРИГИНАЛЬНОСТЬ	85.8%
ЦИТИРОВАНИЯ	0%	ЦИТИРОВАНИЯ	0%
САМОЦИТИРОВАНИЯ	0%	САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 09.06.2025

ДАТА И ВРЕМЯ КОРРЕКТИРОВКИ: 09.06.2025 14:50

Структура документа:
Модули поиска:

Проверенные разделы: основная часть с 2, 5-73, титульный лист с 1, содержание с 4, введение с 3, выводы с 74.
Перепарафразирование по коллекции: IEEE, СПС ГАРАНТ: нормативно-правовая документация, Матрица, Цитирование, Шаблоны фразы, Диссертация ИБ, ОМР России и СНГ; Переводные заимствования; ИПС Адилет; Публикации РГБ; Перепарафразированные заимствования по коллекции Интернет в английском сегменте; Коллекция НБУ, IEEE, Патенты СССР, РФ, СНГ, СПС ГАРАНТ: аналитика; Кольцо вузов; Сводная коллекция ЗБС; Публикации eLIBRARY; Переводные заимствования по коллекции Гарант: аналитика; Публикации РГБ (переводы и перепарафразирование); Публикации eLIBRARY (переводы и перепарафразирование); Кольцо вузов (переводы и перепарафразирование); Перепарафразирование по СПС ГАРАНТ: аналитика; Перепарафразированные заимствования по коллекции Интернет в русском сегменте; Переводные...

Работу проверил: Головяшкин Алексей Николаевич

ФИО проверяющего:

Дата подписи:

Подпись проверяющего



Чтобы убедиться в подлинности справки, используйте QR-код, который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.

Руководитель ВКР

« 09 » 06 2025 год

подпись

Головяшкин А.Н.
фамилия, инициалы



АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

о результатах проверки текстового документа
на наличие заимствований

Федеральное государственное бюджетное
образовательное учреждение высшего
образования "Пензенский государственный
университет"

ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Козин Алексей Сергеевич
Самоцитирование
рассчитано для: Козин Алексей Сергеевич
Название работы: Разработка устройства управления автономными источниками питания
Тип работы: Магистерская диссертация
Подразделение: Кафедра "Нано- и микроэлектроника"

РЕЗУЛЬТАТЫ

■ ОТЧЕТ О ПРОВЕРКЕ КОРРЕКТИРОВАЛСЯ: НИЖЕ ПРЕДСТАВЛЕНЫ РЕЗУЛЬТАТЫ ПРОВЕРКИ ДО КОРРЕКТИРОВКИ

СОВПАДЕНИЯ	16.2%	СОВПАДЕНИЯ	16.2%
ОРИГИНАЛЬНОСТЬ	83.8%	ОРИГИНАЛЬНОСТЬ	83.8%
ЦИТИРОВАНИЯ	0%	ЦИТИРОВАНИЯ	0%
САМОЦИТИРОВАНИЯ	0%	САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 09.06.2025

ДАТА И ВРЕМЯ КОРРЕКТИРОВКИ: 09.06.2025 14:53

Структура документа: Проверенные разделы: основная часть с.2, 5-73, титульный лист с.1, содержание с.4, введение с.3, выводы с.74
Модули поиска: Перефразирования по коллекции IEEE; СПС ГАРАНТ: нормативно-правовая документация; Медицина; Цитирование; Шаблонные фразы; Диссертации НББ; СМИ России и СНГ; Переводные заимствования; ИПС Адилет; Публикации РГБ; Перефразированные заимствования по коллекции Интернет в английском сегменте; Коллекция НБУ; IEEE; Патенты СССР, РФ, СНГ; СПС ГАРАНТ: аналитика; Кольцо вузов; Сводная коллекция ЭБС; Публикации eLIBRARY; Переводные заимствования по коллекции Гарант: аналитика; Публикации РГБ (переводы и перефразирования); Публикации eLIBRARY (переводы и перефразирования); Кольцо вузов (переводы и перефразирования); Перефразирования по СПС ГАРАНТ: аналитика; Перефразированные заимствования по коллекции Интернет в русском сегменте; Переводные...

Работу проверил: Головяшкин Алексей Николаевич

ФИО проверяющего

Дата подписи: 09.06.2025

Подпись проверяющего



Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование
корректным, система оставляет на усмотрение проверяющего.
Предоставленная информация не подлежит использованию
в коммерческих целях.

Приложение 3
(обязательное)
Презентация ВКР

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пензенский государственный университет»

Факультет информационных технологий и электроники

Кафедра: «Нано- и микроэлектроника»

Разработка устройства управления автономными источниками питания

Выполнил: Козин Алексей Сергеевич

Научный руководитель: кандидат технических наук, доцент, заместитель
заведующего кафедрой нано- и микроэлектроники Пензенского
государственного университета Головяшкин Алексей Николаевич

Цель работы: разработать устройство управления автономными источниками питания.

- ▶ Задачи:
- ▶ 1. Разработать устройство управления автономными источниками питания: входное напряжение 24 В постоянного напряжения, выходное напряжение 220 В переменного напряжения, зарядка от сети 220 В переменного напряжения, источник питания постоянного напряжения 24 В - литий-ионные батареи, отображение времени заряда/разряда источника питания, отображение уровня оставшегося заряда.
- ▶ 2. Выполнить аналитический обзор научно-технической литературы.
- ▶ 3. Разработать схемы электрические.
- ▶ 4. Разработать алгоритм работы для системы на кристалле.
- ▶ 5. Разработать описание встроенного приложения для устройства управления автономными источниками питания.

Структурная схема устройства управления автономными источниками питания



Технические характеристики платы

BMS HK1VI-A06A24S-6P

Литий-ионная АКБ	Значения
Способ балансировки	Активный Баланс
Текущий активный баланс	0,6A
Непрерывный ток разряда	60A
Пиковый ток разряда	100A
Защита от перегрузки по току	10A-60A регулируется
Шкала напряжения с одной ячейкой	1-5 V
Точность напряжения	±3 мВ
Напряжение защиты от перегрузки	1,2 В-4,35 В регулируется
Снятие защиты от перезаряда	1,2 В-4,55 В регулируется
Время срабатывания защиты от перегрузки по току	Регулируемый 2S-120S
Напряжение защиты от перегрузки	1,2 В-4,35 В регулируется
Защита от чрезмерного разряда	1,2 В-4,35 В регулируется
Количество датчиков температуры	3
Защита от короткого замыкания	ДА
Функция беспроводного управления	Поддержка Android и IOS
Другой коммуникационный порт	RS-485/ CAN
Размер	136*83*17 мм

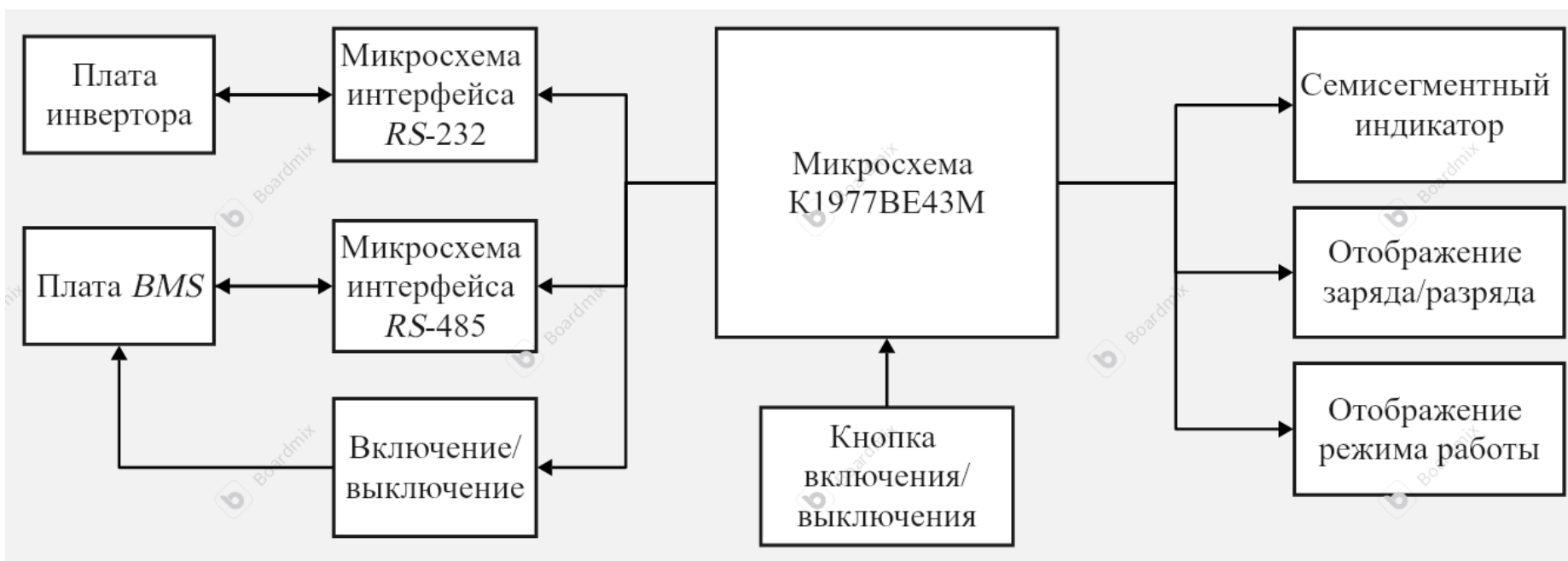


Технические характеристики платы инвертора *TIANMOTPU1000W*

- ▶ Основные характеристики:
- ▶ - номинальная мощность: 1000 Вт (пиковая до 2000 Вт);
- ▶ - входное напряжение: 24 В (DC);
- ▶ - выходное напряжение: 220-240 В (AC), чистая синусоида;
- ▶ - выходная частота: 50 Гц;
- ▶ - разъемы: 1-2 розетки (евро/*USB*, зависит от модификации);
- ▶ - защита: от перегрузки, перегрева, короткого замыкания, низкого/высокого напряжения;
- ▶ - охлаждение: вентилятор;
- ▶ - габариты и вес: 20 × 15 × 5 см, вес ~1-1,5 кг.



Структурная схема платы управления автономными источниками питания



Описание работы встроенного приложения

- ▶ Система на кристалле получает данные с платы *BMS*. После этого происходит извлечение необходимой информации: напряжение на АКБ, уровень оставшегося заряда АКБ, выходной ток с АКБ. На основе этих данных формируются сигналы для отображения полученной информации.
- ▶ 1. Контроль параметров.
- ▶ 2. Отображение оставшегося заряда АКБ.
- ▶ 3. Отображение заряда/разряда устройства.
- ▶ 4. Отображение режима работы устройства.
- ▶ Также включение/выключение платы инвертора может производиться вручную, с помощью кнопки.

Расчет общего искусственного освещения в помещении с ПЭВМ

- ▶ Целью расчета является определение числа светильников, а также числа и мощности ламп, необходимых для обеспечения требуемого нормативного уровня освещенности для работы оператора ПЭВМ.
- ▶ В соответствии с СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к ПЭВМ и организации работы» освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300 - 500 лк. Для расчета требуемый нормативный уровень освещенности E_n был выбран 400 лк.
- ▶ Количество светильников с люминесцентными лампами заданного типа и мощности определяется по формуле:
$$N = \frac{E_n * S * z * K_z}{F * N_l * n}$$
- ▶ Рассчитанное количество светильников: 4.
- ▶ Фактический уровень освещенности по формуле:
$$E_f = \frac{N * N_l * F * n}{S * z * K_z}$$
- ▶ Рассчитанный уровень освещенности равен 434,9 лк.
- ▶ Полученное фактическое значение отличается от принятого требуемого уровня освещенности на 34,9 лк, но соответствует норме СанПиН 2.2.2/2.4.1340-03 «Гигиенические требования к ПЭВМ и организации работы»

Заключение

- ▶ В ходе выполнения выпускной квалификационной работы были достигнуты все поставленные цели:
- ▶ 1. Разработано устройство управления автономными источниками питания с заданными параметрами.
- ▶ 2. Выполнен аналитический обзор научно-технической литературы.
- ▶ 3. Разработаны схемы электрические: структурная, функциональная и принципиальная.
- ▶ 4. Разработан алгоритм работы для системы на кристалле.
- ▶ 5. Разработано описание встроенного приложения для устройства управления автономными источниками питания.

Список публикаций по теме ВКР

- ▶ 1. Головяшкин А.Н., Козин А. С. , Сухов И.В. Алгоритмы вычисления циклического избыточного кода // Труды международного симпозиума "Надежность и качество". - 2024. - Т. 1. - С. 421-422;
- ▶ 2. Головяшкин А.Н., Сухов И.В., Козин А. С., Баннов В.Я. Ускорение вычисления контрольной комбинации // Труды международного симпозиума "Надежность и качество". - 2024. - Т. 1. - С. 415-416.
- ▶ 3. И.Б. Мещеряков, И.В Сухов, М.Н. Рогулин, А.С. Козин, А.А. Фролов, В.Д. Гапоненко. Итеративная обработка кадровой последовательности, переданной *OFDM*-символами с канальным свёрточным кодированием и перемежение по спектру и времени//Радиолокация, навигация, связь: сборник трудов XXXI Международной научно-технической конференции (г. Воронеж, 15-17 апреля 2025 г.): в 6 т. / Воронежский государственный университет; АО «Концерн “Созвездие”». - Воронеж: Издательский дом ВГУ, 2025. - т.2. - с 313-322;

Спасибо за внимание!