ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ
ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ГОРОДА МОСКВЫ
«ПЕРВЫЙ МОСКОВСКИЙ ОБРАЗОВАТЕЛЬНЫЙ КОМПЛЕКС»

Допустить к защите
Зав. отделением СПО
факультета «Информационных
технологий и управления»
_____/Пашохонова Е.Е.
«_____» июня 2021 г.

ДИПЛОМНАЯ РАБОТА

по специальности

09.02.01 Компьютерные системы и комплексы
(код, наименование специальности)

Тема: Разработка микропроцессорной системы для управления программируемым светодиодным 3D кубом

Студента Косякова Дениса Александровича

Группы 41 КС

Руководитель Гюльмалиев Эльдар Агаджанович

Работа защищена «____» июня 2021 г. с оценкой _____

Секретарь ГЭК _____/ Степина Вера Владимировна
Подпись Ф.И.О.

Москва
2021

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

АЦП – Аналогово-Цифровой Преобразователь

BSRR – Bit Set Reset Register

CAN – Controller Area Network

DAC – Digital-To-Analog-Converter

ДУ – Дистанционное Управление

DMA – Direct Memory Access

ЕСКД – Единый Стандарт Конструкторской Документации

EGR – Event Generation Register

I2C – Inter-Integrated Circuit

ODR – Output Data Register

ООП – Объектно-Ориентированное Программирование

МК – Микроконтроллер

МПС – Микропроцессорная Система

NVIC – Nested Vector Interrupt Controller (контроллер вложенных векторов прерываний)

RCC – Reset and Clock Control (блок управления тактированием и сбросом)

RNG – Генератор Случайных Чисел

TRNG – True Random Number Generator

САПР – Система Автоматического Проектирования

SPI – Serial Peripheral Interface

SWD – Serial Wire Debug

ШИМ – Широтно-Импульсная Модуляция

USB – Universal Serial Bus

USART – Universal Synchronous/Asynchronous Receiver/Transmitter

ФНЧ – Фильтр Низких Частот

ФОВ – Функция Обратного Вызова

ВВЕДЕНИЕ

Микроконтроллеры можно встретить всюду в электронных блоках совершенно разных устройств: на материнских платах компьютеров, в контроллерах различных приводов, жестких и твердотельных накопителей, в калькуляторах, на платах управления стиральных машин, микроволновок, телефонов, пылесосов, посудомоечных машин, внутри домашних роботов, программируемых реле, в модулях управления станками и т.д.

В рамках данной выпускной квалификационной работы будет спроектирована микропроцессорная система для управления светодиодным 3D кубом 8x8x8 на базе микроконтроллера STM32F103C8T6, а также микропроцессорная система, позволяющая передавать данные управления в основную МПС, и представляющая собой джойстик.

Актуальность работы обусловлена развитием технологий беспроводной передачи данных и постепенной интеграцией этих технологий во встраиваемые системы. На данный момент выпускается и разрабатывается множество умной бытовой техники, которой можно управлять, используя мобильный телефон, благодаря технологиям беспроводной передачи данных. Этот факт делает специалистов, умеющих разрабатывать системы с использованием технологий беспроводной передачи данных, актуальными на рынке труда.

Объектом исследования является микропроцессорная система для управления светодиодным кубом и джойстик для передачи необходимых данных в куб.

Предмет исследования - структура и функции микропроцессорных систем управления светодиодным кубом и джойстика, а также передача данных между этими двумя системами.

Цель работы: проектирование микропроцессорной системы для управления светодиодным кубом 8x8x8 на базе микроконтроллера STM32F103C8T6 вместе с пультом управления на базе такого же микроконтроллера для вывода трехмерных анимаций и объемного текста.

Для достижения поставленных целей необходимо выполнить следующие задачи:

- Спроектировать принципиальные схемы устройств
- Спроектировать печатные платы устройств
- Написать код для обеих МПС
- Рассчитать экономическую эффективность проекта

Методы исследования: сбор, синтез и анализ полученной информации.

1. Микроконтроллер как средство управления светодиодным кубом

1.1 Анализ существующих решений

Идея светодиодного куба не нова и от варианта к варианту реализация отличается, но идея остается одной и той же – сдвиговые регистры включают нужный слой и нужные точки на слое в один момент времени. Более подробно принцип работы будет рассмотрен в практической части. На рисунке 1 представлена типовая принципиальная схема светодиодного куба.

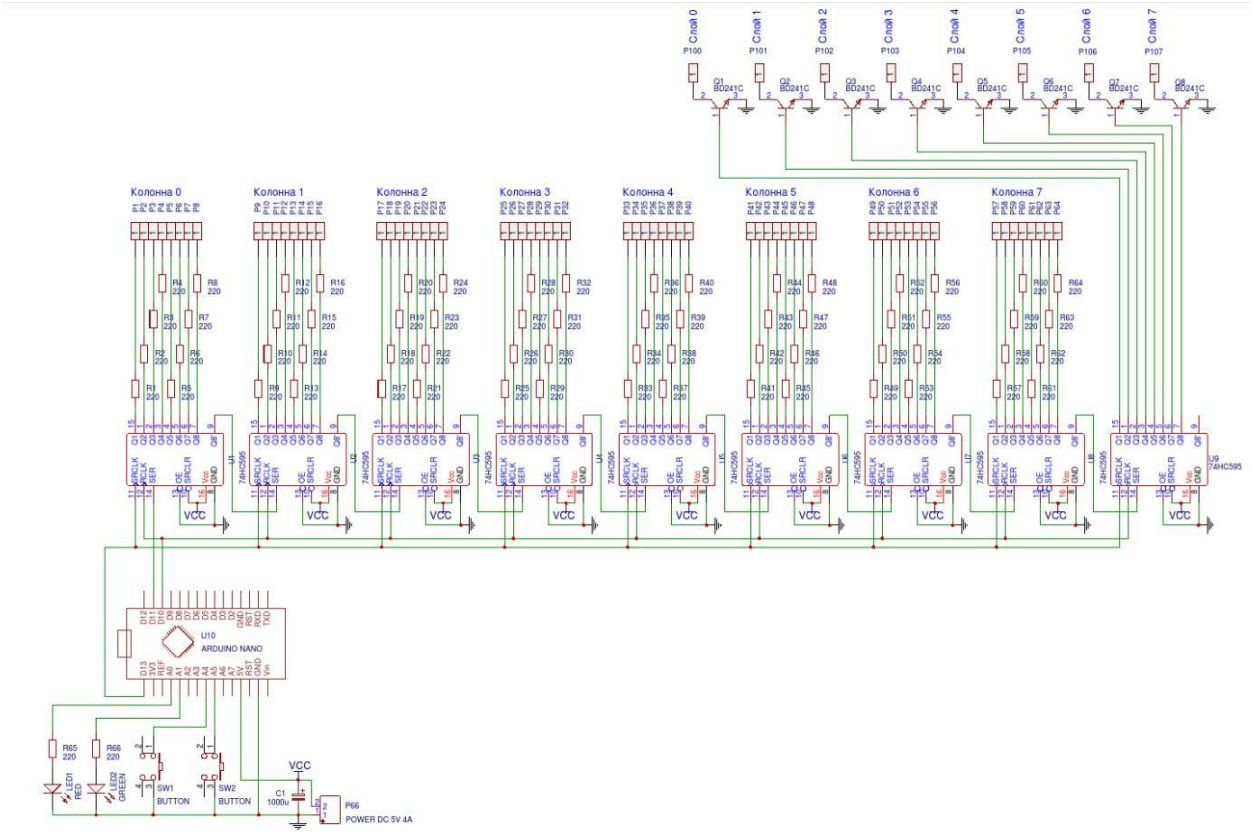


Рисунок 1 - Типовая схема светодиодного куба

В этом проекте будет использоваться другой микроконтроллер и другие транзисторы.

В моей реализации есть нововведение в виде отдельной МПС, которая представляет собой пульт дистанционного управления, которая пересылает данные управления в контроллер, отвечающий за светодиодный куб. Контроллер куба же на основании этих данных принимает решение о том, что ему

нужно сделать в данный момент. Пульт управления устройством будет обладать экраном и будет работать от аккумулятора.

1.2 Выбор элементной базы МПС

Самым главным компонентом любой МПС является микроконтроллер, и первоочередной задачей является выбор микроконтроллера. Рассмотрим два наиболее популярных семейства микроконтроллеров, выделим их преимущества и недостатки и выберем конкретный контроллер для проекта. Наиболее популярными считаются 2 семейства контроллеров:

- AVR (Arduino-подобные контроллеры)
- STM8/STM32

1.2.1 Микроконтроллеры AVR

Микроконтроллеры данного семейства производятся компанией Microchip и являются самым популярным семейством контроллеров. На базе этого контроллера в 2002-м году инженером Массимо Банзи была начата разработка устройства, которое позволяет студентам быстро собирать электронные устройства. Этим устройством стало Arduino. Первый прототип был выпущен в 2005-м году. На текущий момент платы Arduino используются электронщиками-любителями во всем мире ввиду простоты программирования этих контроллеров.

В целом оригинальная плата Arduino стоит достаточно дорого, но существует множество таких же, как и оригинальная Arduino плат, благодаря принципиальной схеме, которая находится в открытом доступе, но цена их в разы меньше. В зависимости от модели платы зависит память и количество выводов устройства. Так же стоит отметить огромную поддержку сообщества, благодаря которому было выпущено огромное множество модулей и библиотек.

Самой популярной моделью Arduino является Arduino UNO rev 3, которая основана на базе контроллера Atmega 328(p).

Эта модель стала настолько популярной, что отладочные платы других производителей используют дополнительные дублирующие выводы под такое же расположение выводов, для использования плат расширения, изначально разработанных для Arduino.

Технические характеристики контроллера отображены в таблице 1.

Таблица 1 - Характеристики Arduino UNO rev3

Характеристика	**Значение**
Архитектура	RISC
Разрядность шины данных	8 бит
Напряжение питания и напряжение высокого логического уровня	5 В
Максимальный потребляемый ток	200 мА
Цифровые входы/выходы	14 (из них 6 могут использоваться в качестве ШИМ-выходов)
Аналоговые входы	6
Максимальный ток одного вывода	40 мА
Flash-память	32 КБ из которых 0.5 КБ используются загрузчиком
SRAM	2 КБ
EEPROM	1 КБ
Размер вектора прерываний	2 инструкции на вектор
Тактовая частота	16 МГц
Интерфейсы передачи данных	USART – 1 SPI – 2 I2C - 1
Таймеры	8-бит – 2 16-бит - 1
АЦП	8 каналов, 10 бит

Так же стоит отметить наличие сторожевого таймера, позволяющий перезапуск контроллера в случае зависания программы и встроенный на плату

адаптер USB-USART позволяющий прошивать контроллер без дополнительных аппаратных средств. Еще одним преимуществом можно выделить библиотеки симуляции в САПР Proteus, которые позволяют симулировать работу устройства с Arduino, не имея под рукой контроллера.

Среди недостатков можно выделить следующее: 8-битная архитектура, относительно маленькая частота тактирования, отсутствие прямого доступа к памяти (DMA), довольно «тяжелая» и не гибкая стандартная библиотека.

1.2.2 Контроллеры семейства STM32

Контроллеры этого семейства основаны на 32-битных ядрах процессоров ARM Cortex и имеют RISC архитектуру. Компания ST для каждого микроконтроллера выбирает свой набор опций, что позволяет выпускать контроллеры как с высокой производительностью, так и с низким энергопотреблением.

Существуют следующие платформы контроллеров:

- Высокопроизводительные: F2, F4, F7, H7
- Широкого применения: F0, G0, F1, F3, G4
- Сверхнизкого потребления: L0, L1, L4, L4+, L5
- Беспроводные WB, WL

В данном семействе самой популярным контроллером является STM32F103C8T6, ввиду ее низкой стоимости и малых размеров отладочной платы. Технические характеристики отображены в таблице 2.

Таблица 2 - Характеристики STM32F103C8T6

Характеристика	**Значение**
Архитектура	RISC
Разрядность шины данных	32 бит
Напряжение питания и напряжение высокого логического уровня	3.3 В
Максимальный потребляемый ток	150 мА

Продолжение таблицы 2

Порты ввода-вывода	37 (из них 15 могут использоваться в качестве ШИМ-выходов и 21 из них толерантны к 5В)
Аналоговые входы	10
Максимальный ток одного вывода	20 мА
Flash-память	64 КБ из которых 0.5 КБ используются загрузчиком
SRAM	20 КБ
EEPROM	Нет
Тактовая частота	72 МГц
Интерфейсы передачи данных	USART – 3 SPI – 2 I2C – 2 USB – 1 CAN - 1
Таймеры	16 bit – 3 С расширенным управлением - 1
АЦП	2 АЦП. 12 бит 10 каналов

Также стоит отметить наличие сторожевых таймеров, нескольких режимов энергопотребления, контроллер DMA, встроенные часы реального времени и датчик температуры. Производитель рекомендует использовать датчик температуры для измерения изменения температуры, а не абсолютного значения.

Код для контроллеров можно писать как на чистом языке программирования С, так и на объектно-ориентированном С++ после некоторых манипуляций в среде разработки. Так же производитель предлагает работу с контроллером через две библиотеки – CMSIS и HAL. Библиотека CMSIS является низкоуровневой, потому что, используя ее, программист работает с периферией напрямую (запись битов в регистры с помощью маски, обработка флагов прерываний и т.д.). Библиотека HAL является более высокоуровневой, потому что программисту не нужно знать как и к каким регистрам периферии нужно обращаться.

Пожалуй, единственным значительным недостатком является не самый удобный встроенный загрузчик на отладочной плате, хотя и эта проблема решается с помощью внешнего SWD программатора ST-Link V2.

1.2.3 Обоснование выбора процессоров STM в качестве основы МПС

Сравнивая характеристики контроллеров Atmega 328 и STM32F103C8T6 очень легко заметить, что за относительно небольшой прирост в цене мы мало того, что получаем в разы улучшенные характеристики самого процессора (максимальная частота тактирования, аппаратные коммуникационные блоки, объем FLASH и SRAM), но и наличие технологи DMA, которая позволяет ускорить передачу данных и снять нагрузку с ядра контроллера. Так же на выбор контроллера для данного проекта повлияла система тактирования. У контроллеров Atmega используются биты конфигурирования тактирования. Их недостаток в том, что если указать при конфигурации тактирование от внешнего кварцевого резонатора, то в случае нештатной работы этого источника тактирования, контроллер перестанет выполнять инструкции. У контроллеров STM совсем иная ситуация. При запуске контроллеры STM тактируются от внутреннего RC генератора, затем в любом месте программы можно внести изменения в блок RCC, и если с указанным источником тактирования что-то случится, то контроллер сам перейдет на тактирование от внутреннего генератора.

Таким образом было принято решение взять в качестве основы обеих МПС контроллер STM32F103C8T6.

1.2.4 Выбор остальной компонентной базы

Светодиоды для этого проекта должны иметь следующие характеристики: d = 5mm, $I_{\text{пр.макс.}} \sim 20\text{мА}$.

Путем нехитрых математических вычислений можно понять, что нам необходимо 512 светодиодов. Однако у выбранного контроллера всего 37 управляемых выходов, что говорит о том, что необходимо разработать способ

управления таким количеством светодиодов, используя дополнительные микросхемы. Еще одна проблема состоит в том, что мощность, выдаваемая выводом микроконтроллера сильно ограниченна. Ввиду этого управление любой нагрузкой напрямую от контроллера является плохой практикой. Для решения этих проблем будут использоваться сдвиговые регистры 74HC595 для динамической индикации. Один из них будет открывать соответствующий полевой транзистор IRLML6401 в соответствующий момент времени, а остальные будут открывать соответствующие светодиоды. Для изменения яркости будет использоваться переменный резистор номиналом 50 кОм.

Для платы джойстика, кроме контроллера, были подобраны следующие компоненты:

- LCD TFT экран на базе чипа ILI9225
- Микросхемы заряда и защиты батареи TP4056 и DW01A
- Микросхема повышающего преобразователя MT3606
- Стабилизатор питания на 3.3В NCP1117

Для передачи данных между двумя МПС используются Bluetooth модули на базе чипа HC-05.

Вся остальная компонентная база подобрана исходя из типовых подключений этих микросхем с использованием формул, представленных в технической документации.

Полный перечень компонентов для обеих МПС представлен в приложениях.

1.3 Выбор программных средств для проектирования микропроцессорной системы

1.3.1 Altium Designer

Altium Designer – САПР для проектирования электронных устройств и трассировки печатных плат с возможностью создания собственных библиотек компонентов и 3D моделирования конечного устройства. В данном проекте он

будет использоваться ввиду больших возможностей по формированию документации, с соблюдением как отечественных, так и международных стандартов, и файлов для производства печатных плат. Однако стоит отметить, что для формирования документации по ЕСКД необходимо установить дополнительные файлы или создать их самостоятельно.

1.3.2 STM32CubeIDE

До недавнего времени чтобы прошить микроконтроллер, необходимо было использовать, как минимум, 3 программы:

- CubeMX, в котором мы конфигурируем контроллер и его выводы;
- Atollic TRUE Studio – собственно сама среда разработки, которая в конечном итоге формирует .hex/.bin файлы, которые можно записать в контроллер. Однако в ней были некоторые неудобства с отладкой, связанные с тем, что приходилось каждый раз перед отладкой создавать новый файл отладки;
- ST-Link Utility, чтобы записать программу в контроллер.

Для разработки только кода иметь 3 программы как минимум неудобно. Но компания ST в конце 2019-го года выпустила среду разработки STM32CubeIDE, в которой совмещены все три программы, описанные выше. Эта среда использует платформу Eclipse и поддерживает языки программирования C и C++.

1.3.3 Программирование микроконтроллера с использованием языка C++

C – компилируемый язык программирования общего назначения со статической типизацией данных (тип каждой переменной и возвращаемый тип функции должен прописываться вручную), разработанный в 1969-1973 годах Деннисом Ритчи как развитие языка B. Первоначально он разрабатывался для реализации операционной систему UNIX. Согласно дизайну языка, его конструкции близко сопоставляются типичным машинным инструкциям, благо-

даря чему он нашел применение как в операционных системах, так и в различном прикладном программном обеспечении для множества устройств. Этот язык не зря называют отцом языков программирования ввиду того, что его синтаксис стал основой C-Like языков программирования (C++, C#, Java, Objective-C).

C++, как и C, является компилируемым языком программирования общего назначения со статической типизацией данных. В начале 80-х годов Бьёрн Страуструп дополнил язык C под свои нужды таким образом, что в конечном итоге на C++ можно писать так как и на C, но и при этом можно писать код используя парадигму ООП. А потому, его конструкции достаточно близко сопоставляются типичным машинным инструкциям, но при этом имеет более высокую функциональность.

Стоит отметить, что C++ активно начинает использоваться во страиваемых системах, потому что обработку любого устройства можно заключить в класс и потом, если вдруг необходимо добавить несколько таких же устройств, то можно создать еще 1 экземпляр класса на каждое устройство, что снижает трудозатраты на написание рутинного кода.

В данном проекте, чтобы продемонстрировать возможность написания кода для микроконтроллеров на разных языках, код для основной МПС будет написан на C++, а для МПС пульта управления на C.

2. Разработка микропроцессорных систем для управления светодиодным кубом

2.1 Проектирование платы управления светодиодным кубом

Как было сказано выше, управлять всеми светодиодами напрямую не получится, поэтому необходимо разработать схему динамической индикации. Идея такого типа индикации заключается в том, что человеческий глаз не может различать быстро переключающиеся точки, и если разбить все светодиоды на группы и быстро и последовательно их переключать, то человеческий глаз будет видеть, что все светодиоды будут гореть одновременно, несмотря на то, что в один момент времени горит только одна группа светодиодов.

В данном случае 512 светодиодов были поделены на 8 групп по 64 светодиода, и каждая группа светодиодов представляет собой адресуемый слой.

Формирование группы светодиодов происходит по следующему принципу:

- Каждый анод светодиода внутри слоя соединен с каждым другим анодом
- Катоды светодиодов внутри слоя не имеют общих соединений и образуют собой шину данных.
- Каждый катод внутри одного слоя соединен с каждым другим соответствующим катодом на всех остальных слоях.
- Аноды слоев не имеют общих соединений с анодами других слоев и образуют шину адресов.

Из принципа построения следует, что для того, чтобы зажечь любой конкретный светодиод на получившемся кубе необходимо подать напряжение питания на соответствующий анод слоя (шина адресов), а соответствующий катод необходимо соединить с общим проводом.

Также стоит упомянуть то, что сопротивление открытого диода само по себе очень низкое, а ограничение по току составляет 20 мА. Поэтому для каждой линии шины данных необходимо поставить сопротивление, которое обычно рассчитывается по закону Ома:

$$R_{\text{св.}} = \frac{U_{\text{пит}}}{I_{\text{св.макс.}}} = \frac{5\ \text{В}}{0{,}02\ \text{А}} = 250\ \text{Ом}$$

Вычисления выше были бы актуальны только если адресация была статической. Но при динамической индикации светодиод не горит постоянно, а в течение определенного времени с определенной периодичностью. На рисунке 2 представлена временная диаграмма переключения групп светодиодов при динамической индикации цифровых семисегментных индикаторов.

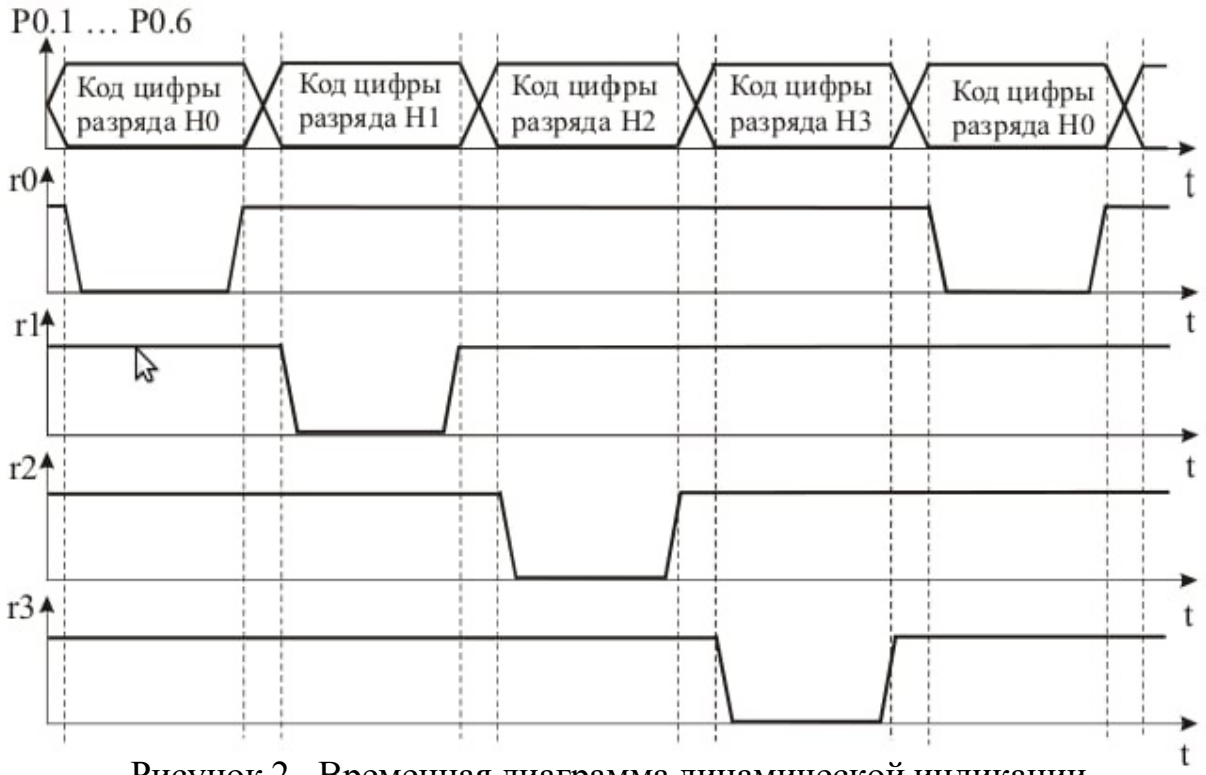


Рисунок 2 - Временная диаграмма динамической индикации

В данном случае графики r0…r3 показывают потенциалы в каждый момент времени на управляющей шине для каждой группы светодиодов. Здесь можно заметить, что сигналы периодичны вне зависимости от кода цифр для разрядов, а значит, если рассматривать каждую линию отдельно от остальных,

получается, что нагрузка управляется сигналом с Широтно-Импульсной Модуляцией.

Широтно-Импульсная Модуляция (ШИМ) – метод управления нагрузкой путем периодичного включения и выключения управляемой нагрузки с помощью транзистора. ШИМ сигнал обладает следующими характеристиками:

- Период (T) – время от начала первого импульса сигнала, до начала второго сигнала.
- Частота (f) – величина, обратная периоду
- Коэффициент заполнения (D) – отношение длины импульса, открывающего транзистор к периоду сигнала.
- Скважность (S) – величина, обратная коэффициенту заполнения.

Среднее значение напряжение, которое можно использовать для вычисления необходимого сопротивления, вычисляется по формуле:

$$U_{\text{ср}} = U_{\text{амп}} * D$$

На данном этапе нельзя точно сказать, чему будет равен период или частота сигнала, но можно вычислить коэффициент заполнения. Из рисунка 2 также видно, что время импульсов на управляющей шине имеет фиксированную длительность, которая обратно пропорциональна количеству линий на этой шине.

Соответственно если известно количество переключаемых групп, коэффициент заполнения вычисляется по формуле:

$$D = \frac{1}{n} = \frac{1}{8} = 0{,}125$$

Где n – количество переключаемых групп.

Таким образом среднее значение напряжения равно:

$$U_{\text{ср}} = 5\ \text{В} * 0{,}125 = 0{,}625\ \text{В}$$

Отсюда необходимое сопротивление:

$$R_{\text{св.}} = \frac{U_{\text{пит}}}{I_{\text{св.макс.}}} = \frac{0{,}625\ \text{В}}{0.02\ \text{А}} = 31{,}25\ \text{Ом}$$

Однако это минимальное сопротивление, при которой схема работает уже при индикации. Во время работы так же не стоит исключать то, что при перепрошивке контроллера он перестает выполнять инструкции, а данные в адресной шине сохраняются и из-за этого увеличивается время импульса на шине адресов, что приведет к выходу из строя светодиода. Поэтому сопротивление должно быть таким же, как и при статической индикации.

В данном проекте я использую сопротивление 220 Ом, поскольку из-за того, что каждый светодиод даже внутри одной партии отличается по характеристикам, ток через один светодиод уменьшается по мере увеличения количества светодиодов в группе динамической индикации.

Для реализации шин адресов и данных используются сдвиговые регистры 74HC595.

В технической документации описана принципиальная схема и временная диаграмма для последовательной записи.

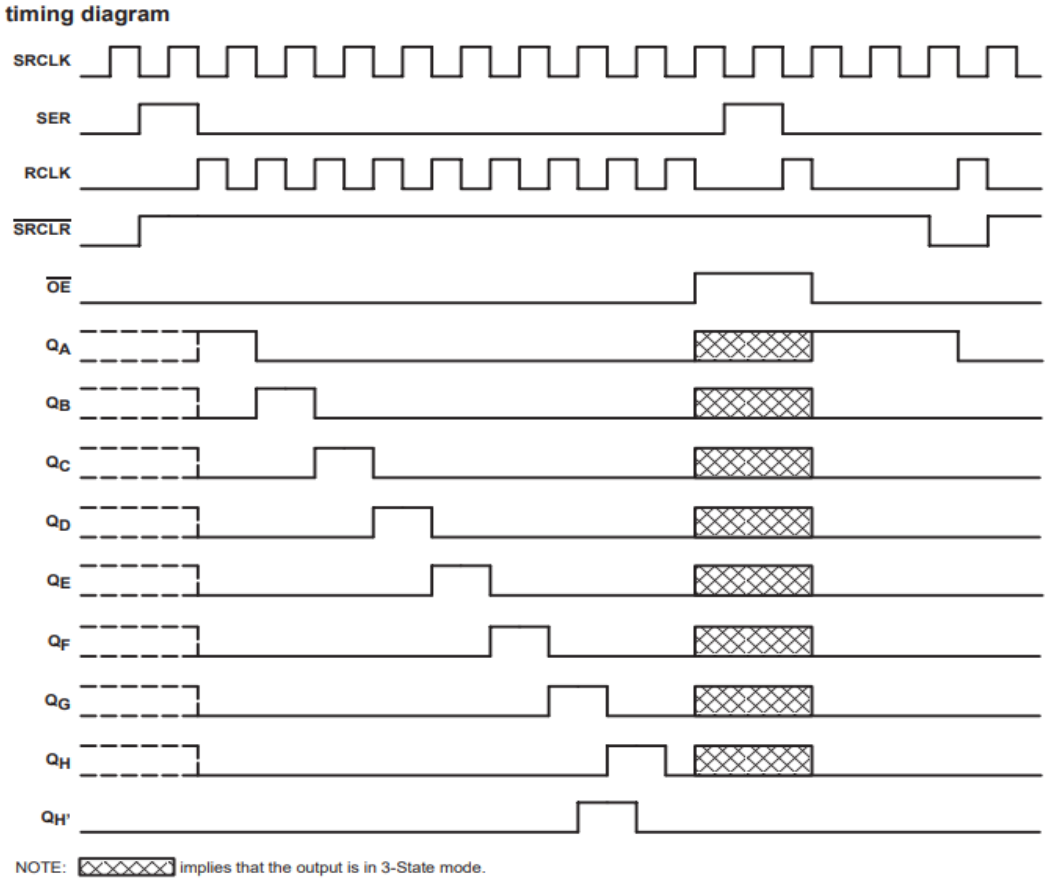


Рисунок 3 - Временная диаграмма работы сдвигового

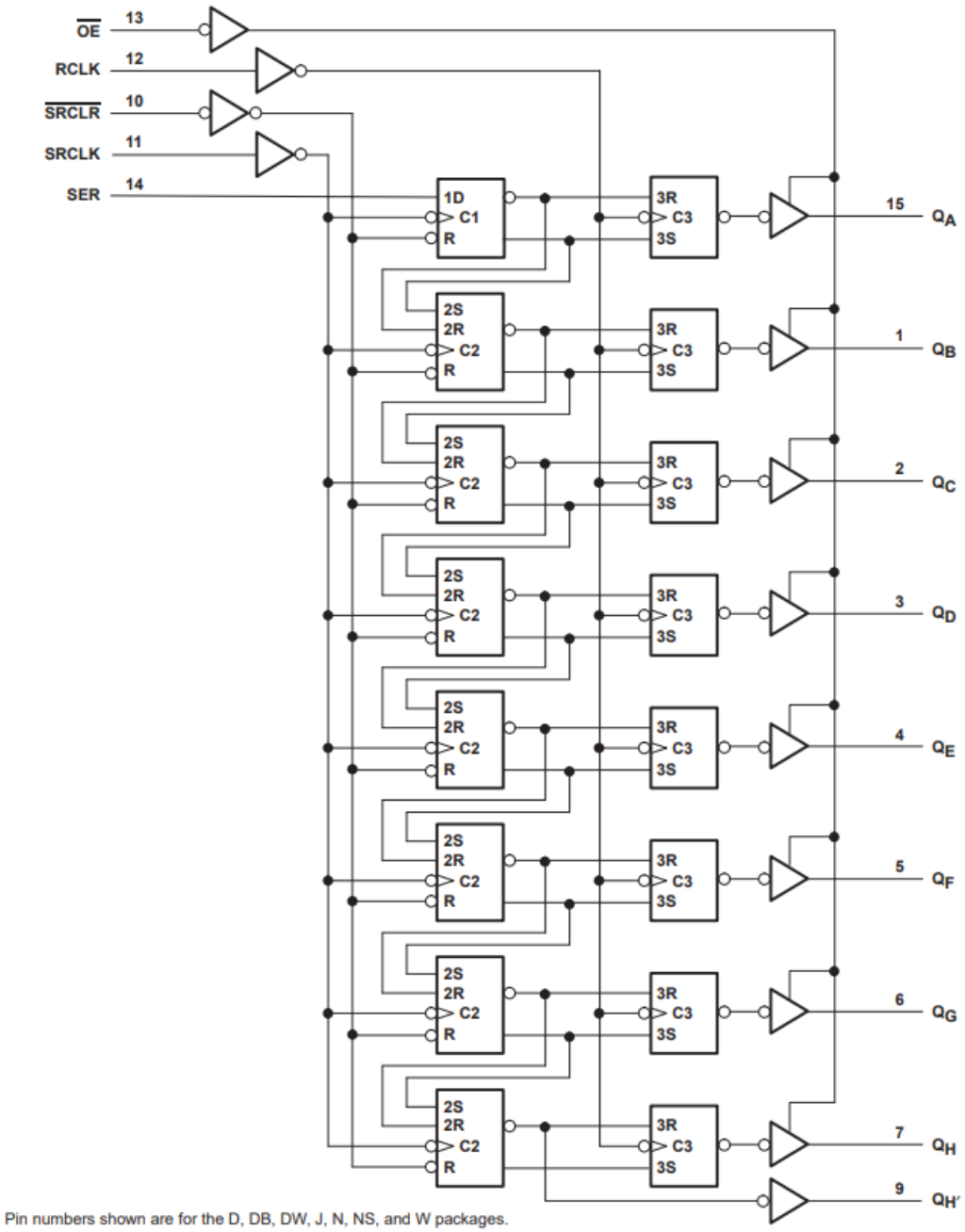


Рисунок 4 - Внутренняя принципиальная схема сдвигового регистра

Из рисунков выше можно сформулировать принцип записи 1 байта информации в сдвиговый регистр:

1. Подать на вывод RCLK логический ноль
2. Подать на вывод SERCLK логический ноль
3. Сформировать бит, который должен записаться в данной итерации и установить вход SER в соответствующий логический уровень

4. Подать на вывод SERCLK логическую единицу (в этот момент происходит сдвиг внутри буфера, и запись бита внутрь регистра)
5. Повторять шаги 2…5 до тех пор, пока буфер не заполнится
6. Подать на вывод RCLK логическую единицу

Последний сдвиговый регистр (DD10) отвечает за переключение слоя, но выходной ток у сдвигового регистра недостаточен (выходной ток сдвигового регистра не превышает 20 мА). Для решения этой проблемы необходимо поставить P-канальный полевой транзистор. Максимальный ток сток-исток транзистора рассчитываем в соответствии с формулой:

$$I_{ds} = \frac{VCC}{R_н} * N; \qquad (1)$$

Где VCC – напряжение питания, Rн – сопротивление резистора, подключенного к светодиоду. N – количество светодиодов в группе. В нашем случае VCC = 5 В; Rн = 220 Ом; N = 64.

Подставляем в формулу (1) и получаем следующее:

$$I_{ds} = \frac{5}{220} * 64 \approx 1{,}45 \text{ А}$$

С запасом возьмем транзистор с максимальным током сток-исток 4,3 А, а именно IRLML6401TRPBF.

Схема подключения транзисторов и сдвиговых регистров показана на рисунках 5 и 6.

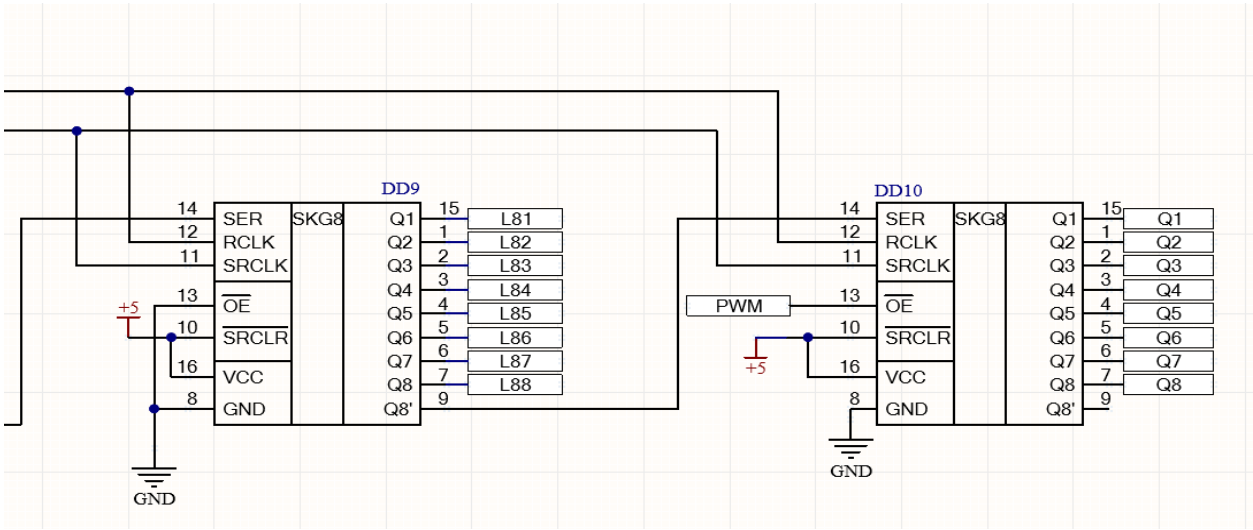


Рисунок 5 - Подключение сдвиговых регистров

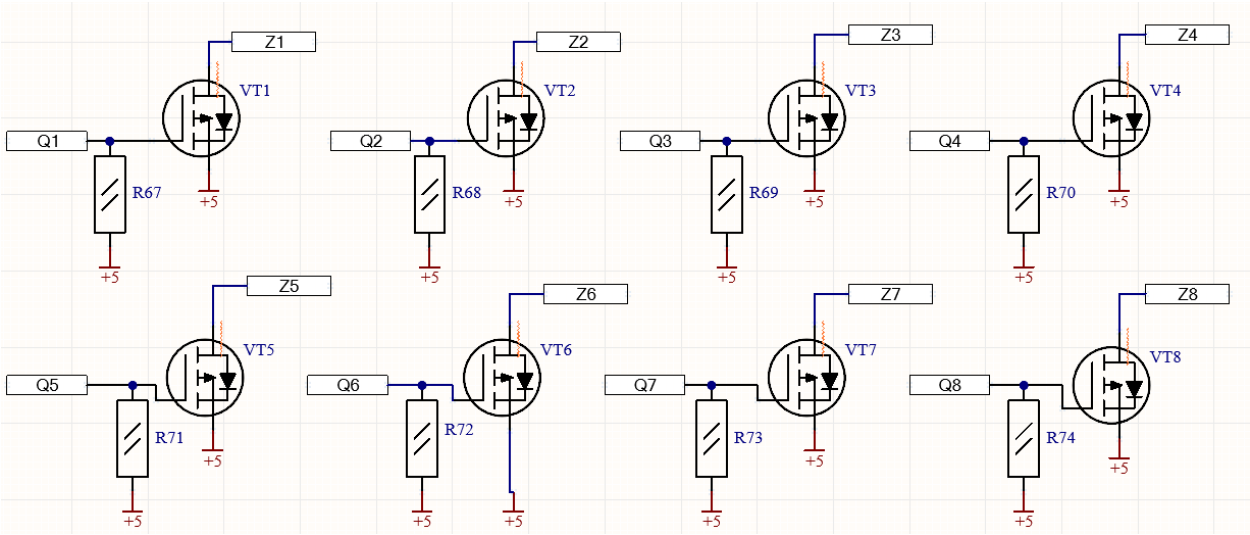


Рисунок 6 - Подключение транзисторов

Так же на рисунке 4 видно, что для изменения яркости свечения светодиодов используется ШИМ сигнал, подключенный к выводу OE сдвигового регистра, отвечающего за переключение слоев.

На рисунке 6 присутствуют резисторы R67…R74. Их назначение в следующем: как было сказано выше, для возможности изменения яркости к выводу OE сдвигового регистра, отвечающего за адресную шину, подключен ШИМ выход микроконтроллера. Но тут есть две особенности:

1. Вывод OE включает или выключает выходной буфер сдвигового регистра. Соответственно в момент, когда буфер выключен, затвор транзистора электрически ни с чем не соединен (т.е. находится в высокоимпедансном состоянии).
2. Структура полевого транзистора такова, что между затвором и стоком образуется емкость, которая при открытии и переводе затвора в высокоимпедансное состояние оставляет транзистор открытым.

Поэтому для того, чтобы яркость регулировалась корректно, необходимо подключить подтягивающий резистор номиналом 10 кОм между затвором и истоком транзистора.

Для изменения скважности ШИМ сигнала используется переменный резистор, средний вывод которого необходимо подключить к выводу АЦП микроконтроллера.

В соответствии с технической документацией на LM317, для получения выходного напряжения 3.3V его необходимо подключить по схеме, представленной на рисунке 7.

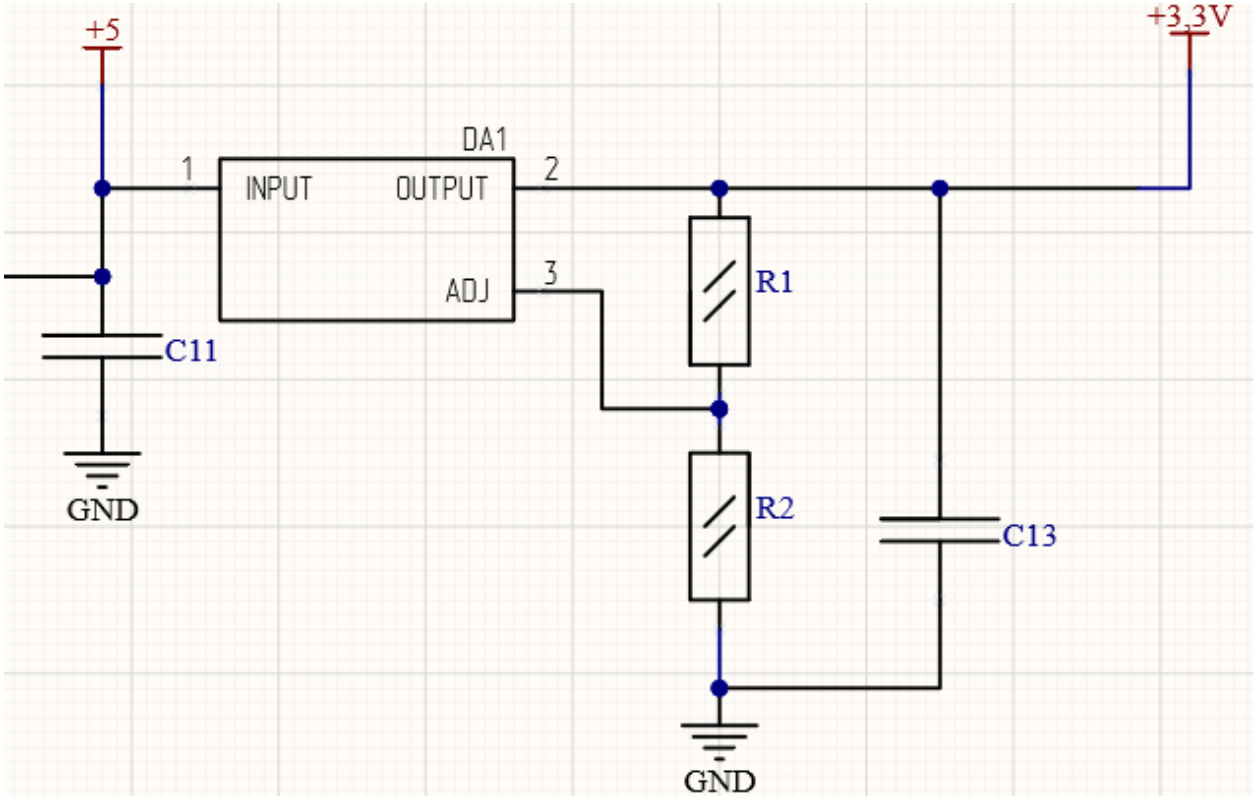


Рисунок 7 - Подключение стабилизатора напряжения

Номиналы сопротивлений рассчитываются по формуле:

$$V_o = V_{ref} * \left(1 + \frac{R2}{R1}\right) + I_{adj} * R2, \qquad \text{где } V_{ref} = 1{,}25\ V;$$

В технической документации описано, что I_{adj} не может превышать 100 uA, поэтому для простоты вычислений этой величиной можно пренебречь.

В данном случае, номиналы резисторов следующие: R1 – 240 Ом, R2 – 390 Ом.

Полный чертеж принципиальной схемы платы управления куба и перечень компонентов представлены в приложении А, Б и В соответственно.

2.2 Проектирование платы пульта управления

Руководствуясь тем, что плата пульта управления работает от аккумулятора, построим блок зарядки аккумулятора на микросхеме TP4056. Типовая схема подключения представлена на рисунке 8.

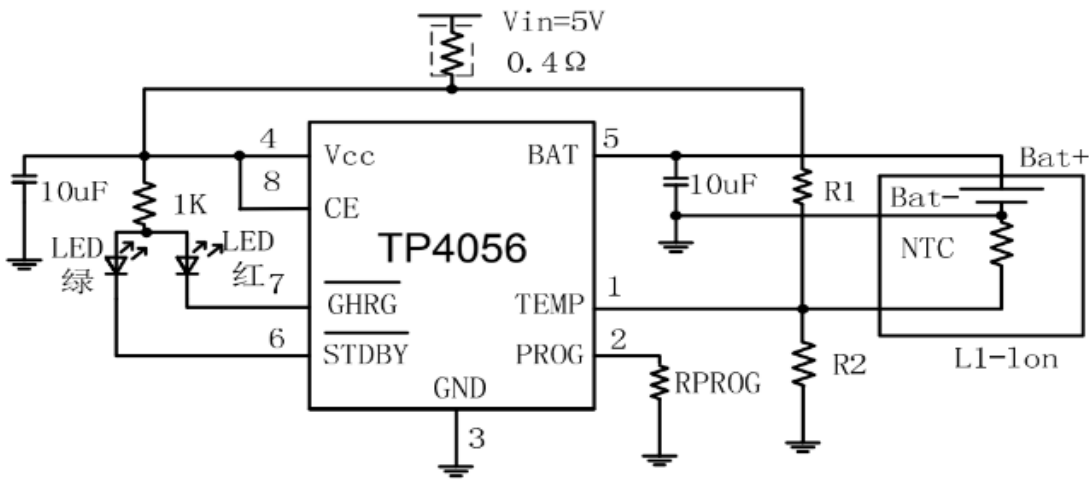


Рисунок 8 - Типовая схема подключения микросхемы заряда аккумулятора

Максимальный ток заряда регулируется резистором RPROG возможные вариации тока заряда представлены в таблице 3.

Таблица 3 - Зависимость тока заряда от токозадающего резистора

Токозадающий резистор (кОм)	**Ток заряда (мА)**
30	50
20	70
10	130
5	250
4	300
3	400
2	580
1.66	690
1.5	780
1.33	900
1.2	1000

В данном случае будет использоваться резистор номиналом 1200 Ом, что соответствует току заряда 1 А.

На этой микросхеме есть выхода, которые показывают статус заряда аккумулятора. Обычно к этим выходам подключают светодиоды разных цветов,

как и показано на рисунке 8. Но в данном случае я буду использовать только один из них, а именно сигнал CHRG. Этот выход необходимо соединить с выводом микроконтроллера. В зависимости от уровня сигнала на этом выводе на экран будет выводится символ, сигнализирующий о том, что в данный момент аккумулятор заряжается. Стоит отметить, что как сигнал CHRG, так и STDBY могут быть только в двух состояниях – напряжение логического ноля и высокоимпедансное состояние. Как и в случае с транзисторами на плате светодиодного куба, здесь необходимо использовать подтяжку к питанию. В данном случае подтяжка к питанию организована внутри микроконтроллера.

Обычно вместе с этой микросхемой используют или аккумуляторы с защитой от перезаряда и переразряда или отдельные микросхемы защиты. В данном проекте используется микросхема защиты DW01A. Она обладает защитой от разряда, переразряда, перегрузки и короткого замыкания аккумулятора.

Типовая схема подключения аккумулятора к микросхеме защиты представлена на рисунке 9.

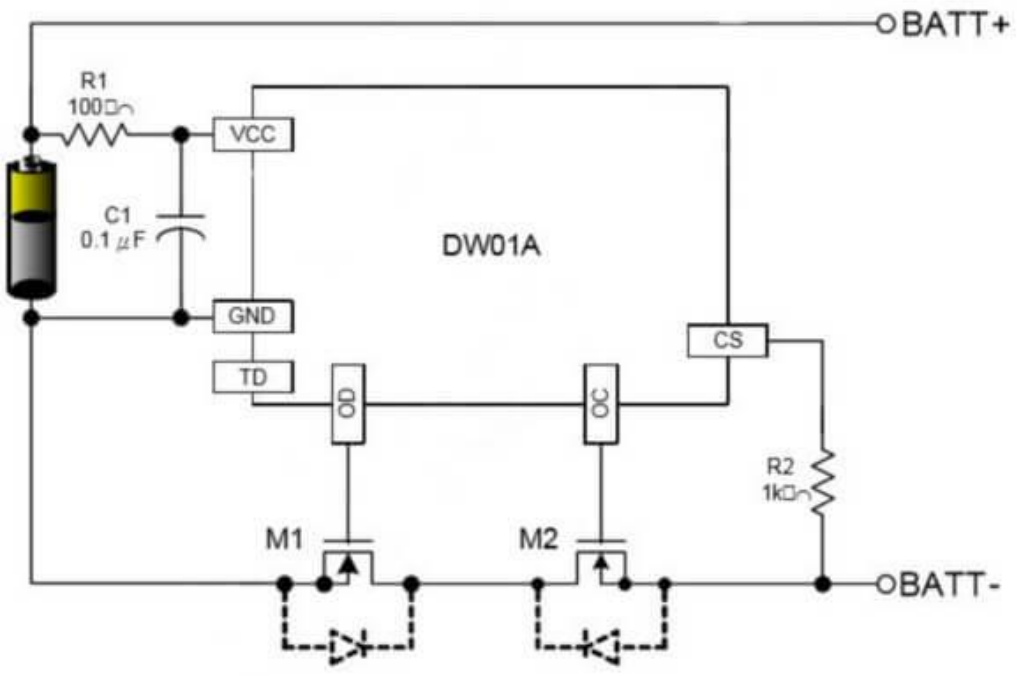


Рисунок 9 - Типовая схема подключения микросхемы защиты аккумулятора

Следующий блок функциональной схемы – блок повышение и стабилизации напряжения. Для питания Bluetooth модуля и экрана необходимо напряжение питание 5 В несмотря на то, что логические уровни обоих элементов 3.3 В. Поэтому в данном проекте необходимо использовать DC-DC повышающий преобразователь на микросхеме MT3608. Типовая схема подключения представлена на рисунке 10.

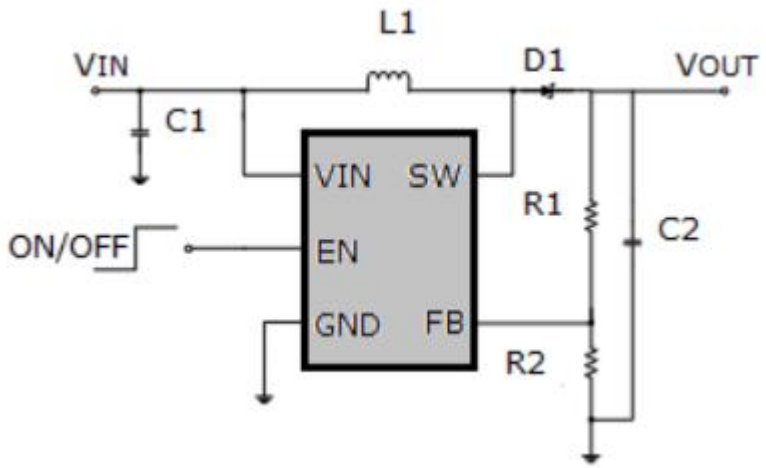


Рисунок 10 - Типовая схема подключения микросхемы повышающего преобразова-

В данном случаем VIN это положительная клемма аккумулятора, GND – отрицательная клемма. Вход EN необходимо соединить с VIN. Номиналы конденсаторов C1, C2 и катушки индуктивности L1 типовые и взяты из технической документации. Номиналы резисторов R1 и R2 рассчитаны исходя из следующей формулы:

$$V_{out} = V_{ref} * \left(1 + \frac{R1}{R2}\right), \text{где } V_{out} = 5.1\ V;\ V_{ref} = 0.6V$$

Для формирования напряжения питания микроконтроллера используется стабилизатор NCP1117, который продается как в виде стабилизатора с настраиваемым напряжением выхода, так и в виде стабилизатора с фиксированным напряжением выхода. В данном случае используется второй вариант. Для вывода изображения используется модуль с LED TFT экраном со стабилизатором напряжения внутри. Интерфейс передачи SPI, и соответственно

необходимо соединить соответствующие выводы между экраном и микроконтроллером. Для управления используется 9 кнопок, которые должны быть подключены к портам микроконтроллера таким образом, что каждая кнопка занимает свою линию внешних прерываний. Более подробно внешние прерывания будут рассмотрены в блоке программирования микроконтроллеров.

Для измерения напряжения необходимо организовать делитель напряжения на два из двух резисторов номиналом 47 кОм.

Для связи обеих МПС необходимо подключить Bluetooth модули к микроконтроллерам по схеме, показанной в таблице 4.

Таблица 4 - Подключение Bluetooth модулей к микроконтроллеру

Вывод микроконтроллера	**Вывод Bluetooth модуля**
UART 3 RX	TX
UART 3 TX	RX
VSS	GND

Микроконтроллер и модуль имеют собственное питание, и оно развязано относительно друг друга.

Подключение источника тактирования и другой необходимой обвязки является типовым и описано в технической документации на микроконтроллер.

Полная принципиальная схема и перечень компонентов представлен в приложениях Ж, З и И соответственно.

2.2 Трассировка печатной платы в Altium Designer

При трассировке печатных плат для обеих МПС используются следующие допуски:

- минимальная ширина проводников 0,3 мм
- минимальный зазор от края печатной платы до проводников 0,254 мм
- минимальный зазор между элементами печатного монтажа 0,127 мм
- минимальный диаметр отверстия выводного монтажа – 0.9 мм
- диаметр переходных отверстий – 0.2 мм

При разработке платы учитывается стандарт IPC-A-610F (правила приема электронных сборок)

Библиотека необходимых компонентов формируются с помощью расширения IPC footprint tool.

Для производства печатной платы необходимо выполнить экспорт файлов в нужном для производства формате. Обычно на сайте производства можно найти инструкцию о том, как сделать экспорт файлов в соответствующей программе для трассировки печатных плат. В моем случае производству нужны файлы Camtestic, которые являются промежуточным этапом между файлом печатной платы и файлами, который можно загрузить в автоматизированный станок для производства печатных плат. Производитель, которого я выбрал для изготовления прототипа, сам преобразует файлы Camtestic в нужный формат.

Сборочный чертеж и компоновка печатных плат предоставлены в приложениях Г, Д, К, Л.

2.3 Программирование микроконтроллеров STM32

Прежде чем писать код для обеих мпс, необходимо разделить общую задачу на две и решить каждую из них отдельно от другой.

Подзадачи для основной МПС:

- Интерфейс для вывода изображения на куб
- Генерация случайных чисел
- Регулировка яркости
- Режимы и их переключение.

Подзадачи для МПС пульта управления:

- Обработка кнопок
- Вывод изображения на экран
- Пользовательский интерфейс
- Измерение напряжения питания аккумулятора

После решения этих подзадач необходимо дополнить код каждой из МПС блоком обработки и передачи данных.

Для написания кода используется STM32CubeIDE и генератор кода инициализации STM32CubeMX внутри среды.

2.3.1 Написание кода для основной МПС

Чтобы не тратить большое количество времени на изучение технической документации, воспользуемся генератором кода инициализации STM32CubeMX. Здесь необходимо инициализировать выходы для передачи данных в сдвиговый регистр как GPIO_Output. На рисунке 11 показаны настройки для портов ввода вывода.

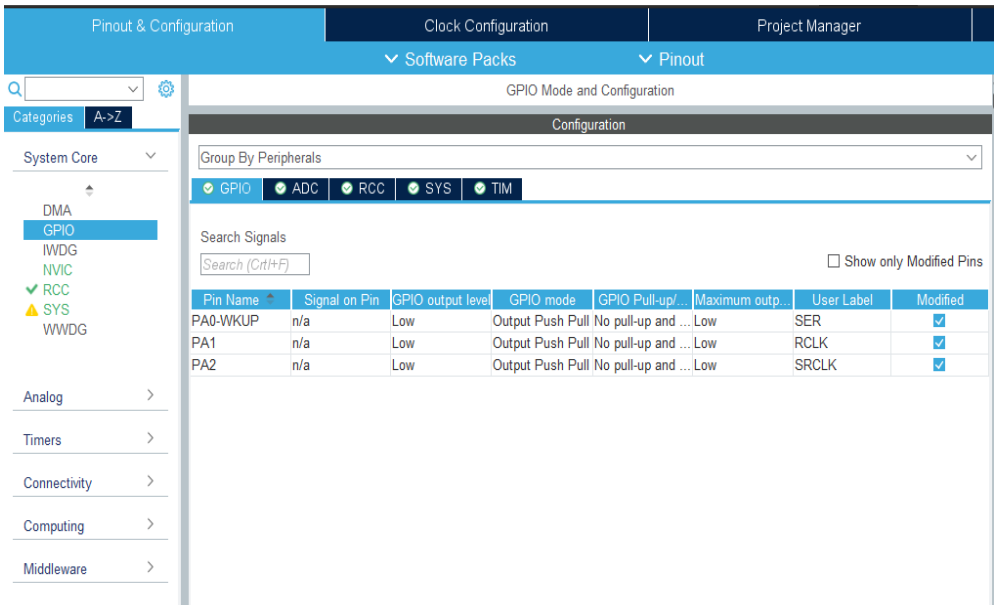


Рисунок 11 - Настройки портов ввода-вывода

Также настраиваем тактирование и интерфейс программирования и отладки (См. Рисунок 12 - 13).

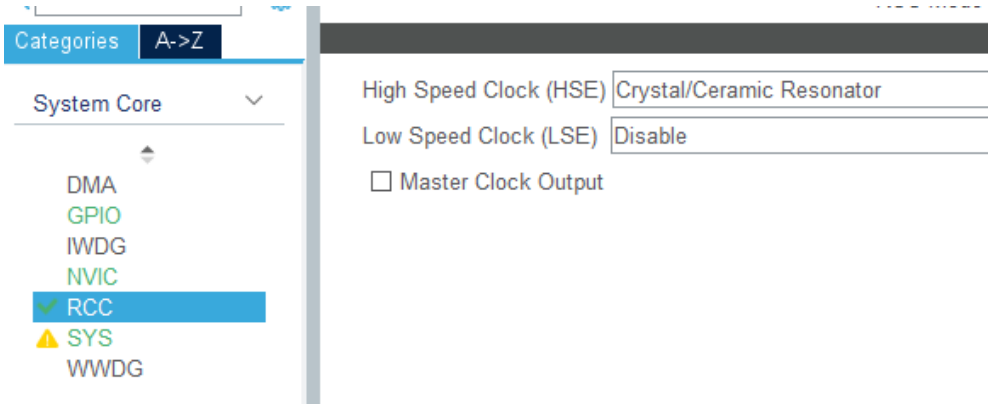


Рисунок 12.1 - Настройки тактирования

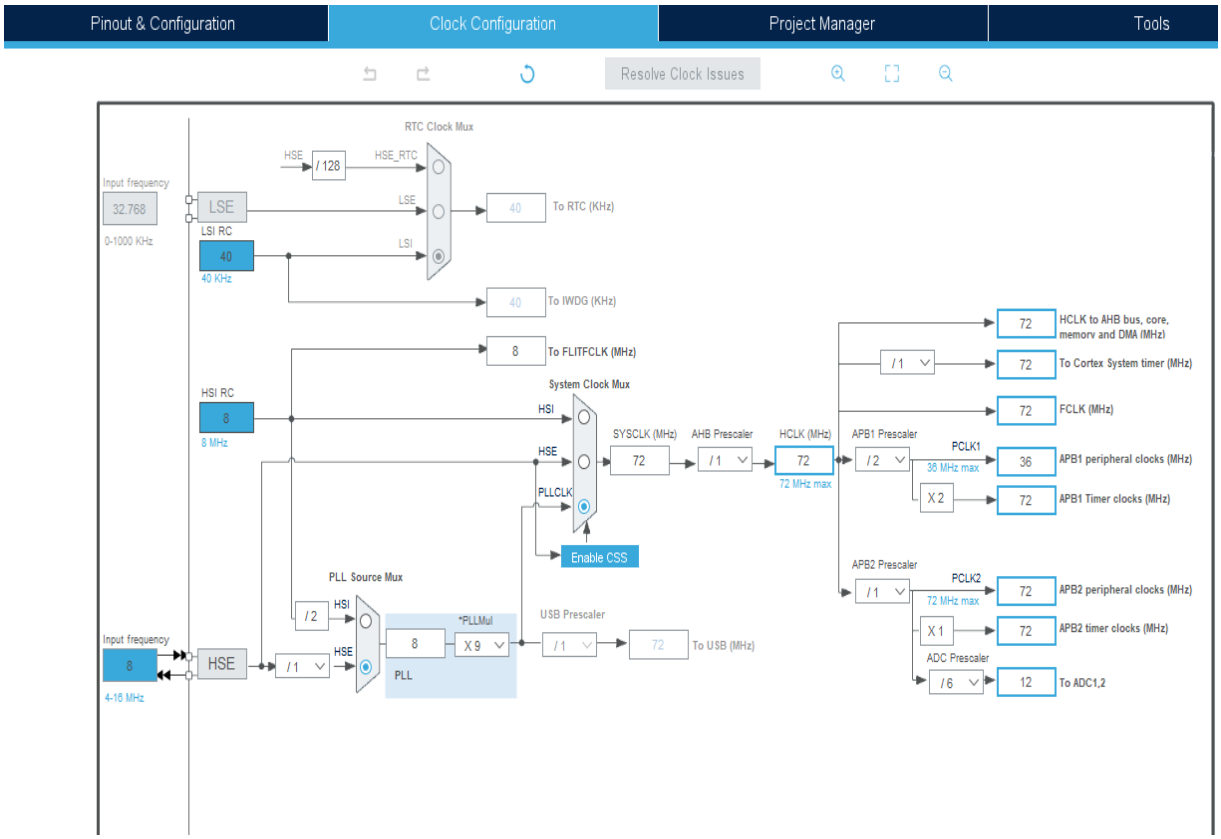


Рисунок 12.2 - Настройки тактирования

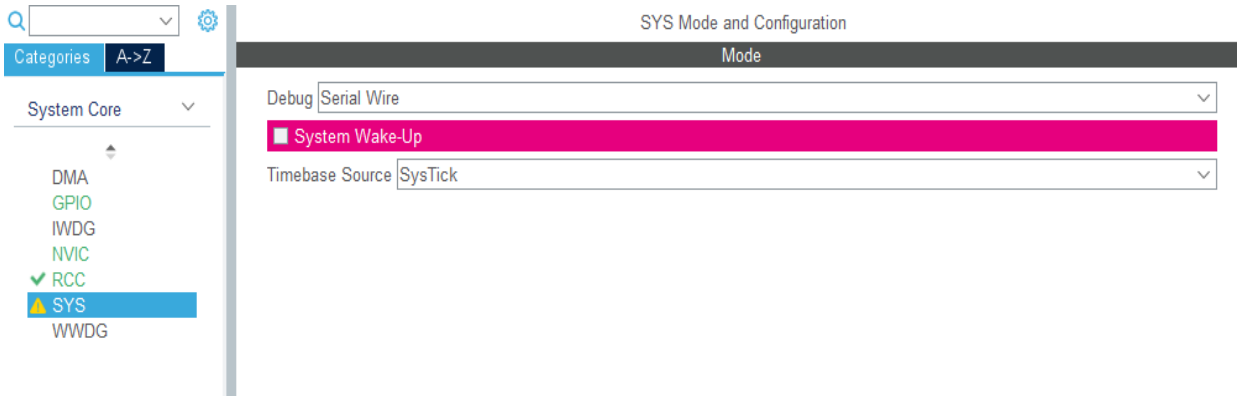


Рисунок 13 - Настройка интерфейса отладки

Для управления яркостью используется четвертый канал таймера два (режим генерации ШИМ), который подключен к выводу OE сдвигового регистра, который отвечает за переключение слоев. Скважность ШИМ сигнала зависит от напряжения, полученного со среднего вывода переменного резистора.

Для измерения напряжения настроим АЦП. Для этого в меню Analog -> ADC2 необходимо выбрать девятый канал (IN9). Затем необходимо выбрать источник сигнала начала измерения (External Trigger Conversion Source). Можно было циклично опрашивать АЦП и увеличить время измерения, увеличив тем самым точность, но нужно иметь ввиду то, что частые измерения могут привести к тому, что контроллер будет заниматься только обработкой прерываний от таймера и АЦП. Поэтому я считаю рациональным проводить измерения напряжения не чаще чем 2 раза в секунду.

Перейдем к таймерам. Сначала настроим третий таймер для запуска измерения напряжения следующим образом: prescaler – 720; Counter Period – 65000; Trigger Event Selection – Update Event. Для определения периода срабатывания таймера используем формулу 2.

$$F(t) = \frac{f(mcu)}{prescaler * counter\ period}; \quad (2)$$

Где f(mcu) – частота тактирования контроллера. Важно отметить, что значения предделителя и периода счета необходимо указывать на 1 меньше фактических. Используя формулу 2 и значения предделителя и периода счета получаем значение частоты таймера равное ~1.5 Гц и периода, равному 0.65 секунды. На рисунке 14 показаны настройки таймера 3.

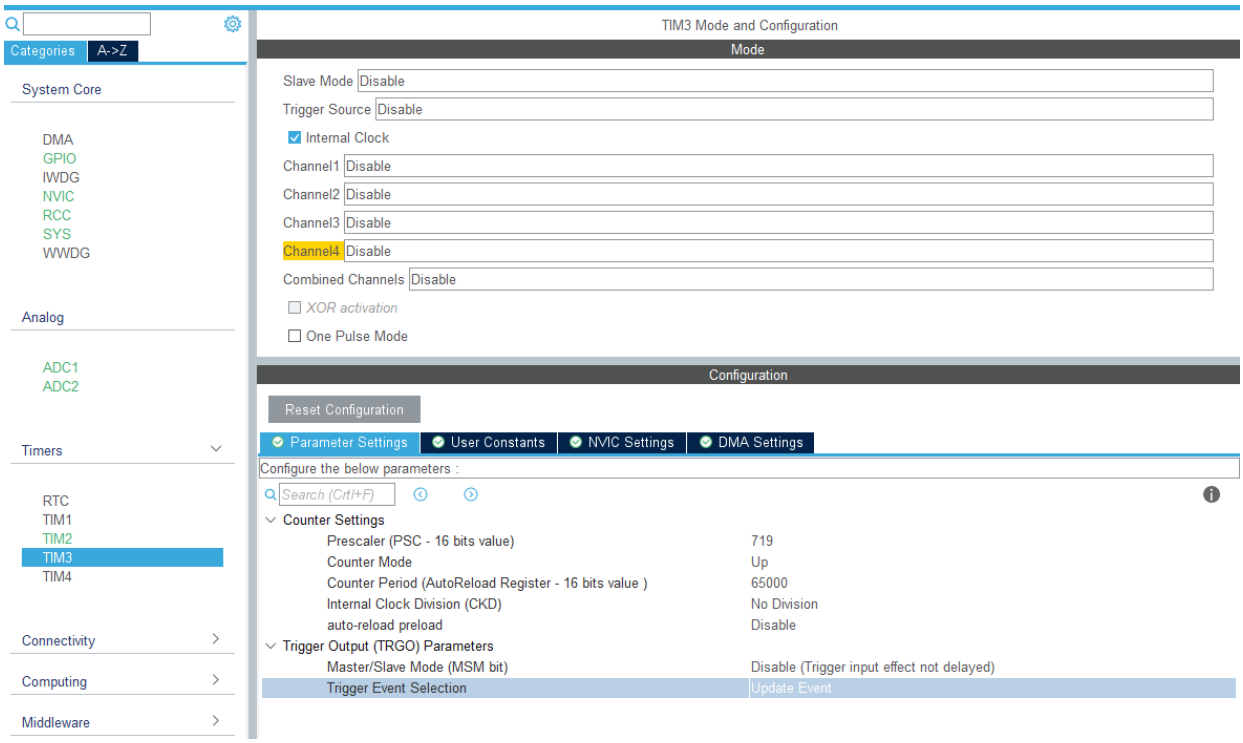


Рисунок 14 - Настройка таймера 3

Как было сказано выше, для управления яркостью используется ШИМ сигнал. Для этого необходимо настроить таймер два следующим образом: prescaler – 12; counter period – 4095. Подставив значения в формулу 2, получим что частота ШИМ сигнала составляет примерно 1.5 КГц. Так же из технической документации можно понять, что для управления яркостью нам на самом деле понадобится обратный ШИМ. Это можно настроить или в пункте PWM Mode или CH polarity (что на самом деле одно и тоже). Однако настройка PWM Mode 2 и CH polarity = Low эквивалентна настройке PWM mode 1 и CH polarity = High. Поэтому в данном случае необходимо поменять только один из параметров. Настройки таймера 2 приведены на рисунке 15.

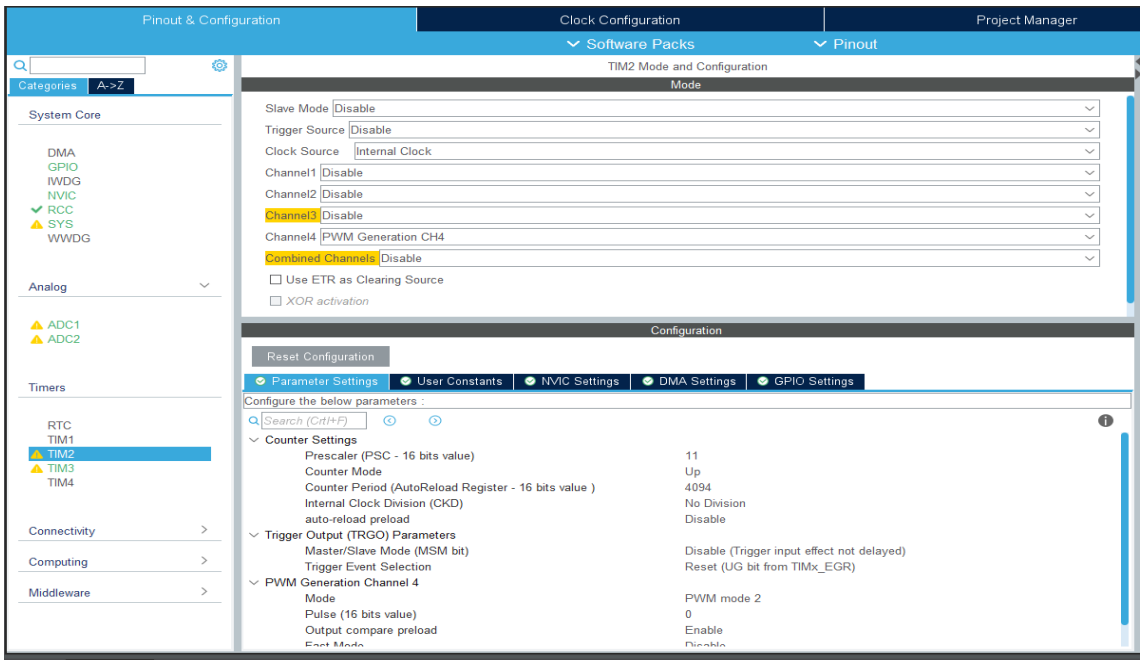


Рисунок 15 - Настройка ШИМ таймера

Для генерации случайных чисел настроим ADC1 для считывания значения с температурного встроенного датчика. На рисунке 16 показана настройка температурного датчика.

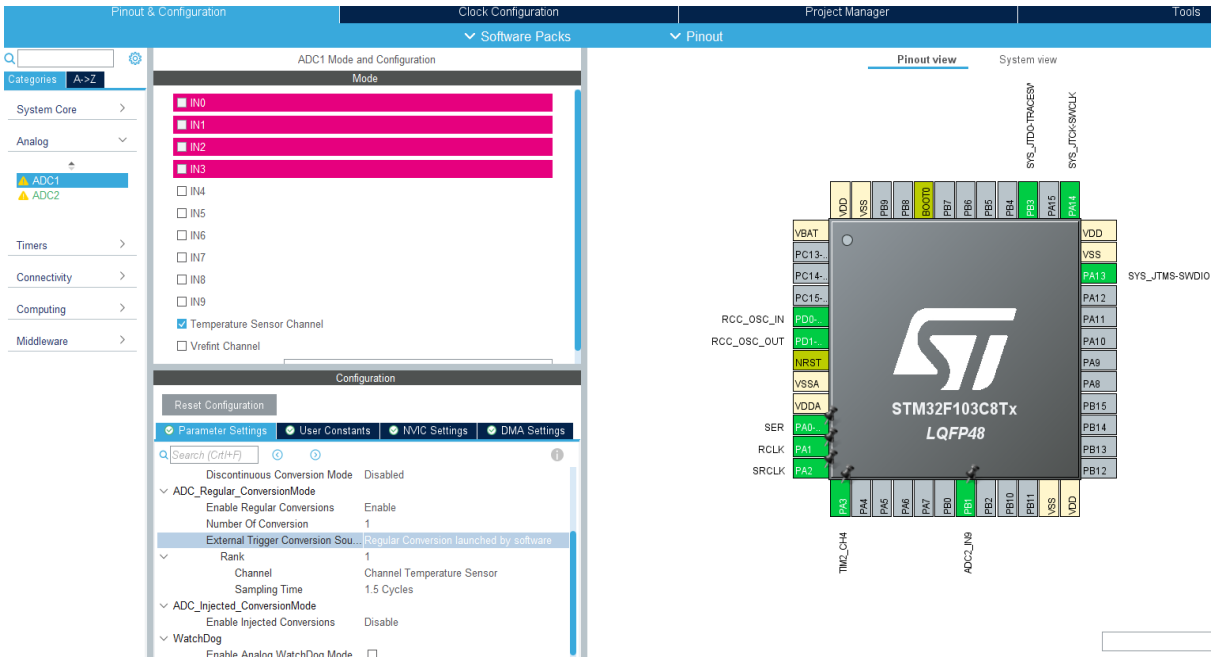


Рисунок 16 - Конфигурация ADC1

Для передачи данных между МПС настроим UART3 в асинхронном режиме, остальные настройки должны остаться без изменений (115200 Bits/s, 8bit, no parity, 1 stop bit).

На этом конфигурация контроллера закончена и можно генерировать код.

Вернемся к блокам программы: Интерфейс, генерация случайных чисел, управления яркостью. Рассмотрим каждый из них.

2.3.1.1 Интерфейс

Интерфейс – это то, каким образом можно управлять тем или иным устройством. Для удобства интерфейс будет реализован с помощью класса Cube. Алгоритм передачи в целом был описан в теоретической части, но нужно теперь понять, что нужно отправлять в сдвиговые регистры, а самое главное – как это должно происходить. Самое простое - выделить трехмерный массив, каждая ячейка которого представляет состояние каждого конкретного светодиода, но из-за того, что тип bool всегда занимает 1 байт, это решение не является оптимальным. Поэтому можно выделить двумерный массив с типом uint8_t (беззнаковое целое число):

```
uint8_t m_OutFrame[8][8] =
{
        {0b11111111, 0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111},
        {0b11111111, 0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111},
        {0b11111111, 0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111},
        {0b11111111, 0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111},
        {0b11111111, 0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111},
        {0b11111111, 0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111},
```

```
                {0b11111111, 0b11111111, 0b11111111, 0b11111111,
    0b11111111, 0b11111111, 0b11111111, 0b11111111},
                {0b11111111, 0b11111111, 0b11111111, 0b11111111,
    0b11111111, 0b11111111, 0b11111111, 0b11111111},
```

В таком случае выделяется всего 64 байта, вместо 512 в случае с трехмерным массивом. Теперь чтобы задать или считать состояние конкретного светодиода, необходимо байт в ячейке [7 - z][x] сдвинуть на y бит вправо, а затем наложить маску единицу. В коде это выглядит так:

```
void Cube::setDot(uint8_t x, uint8_t y, uint8_t z, uint8_t set_num)
{
  m_OutFrame[7 - z][x] &= ~(128 >> y);
      //сбрасываем бит
  m_OutFrame[7 - z][x] |= (128 >> y) * set_num;
      //устанавливаем 0 или 1 в зависимости от set_num
}
```

Для того чтобы отправить все данные мы побайтово записываем их в сдвиговые регистры по следующему алгоритму:

1. Подать на вывод RCLK логический ноль
2. Подать на вывод SERCLK логический ноль
3. Сформировать бит, который должен записаться в данной итерации
4. Подать на вывод SERCLK логическую единицу (в этот момент происходит сдвиг внутри буфера, и запись бита внутрь регистра)
5. Повторять шаги 2…5 до тех пор, пока буфер (все 8 байт) не заполнится
6. Подать на вывод RCLK логическую единицу

Реализовано это следующим образом

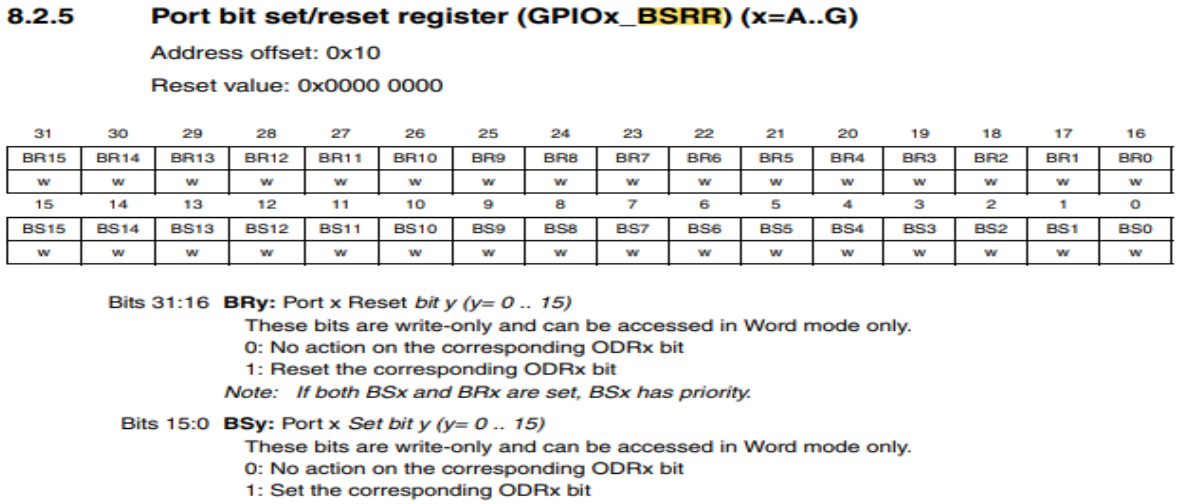
```
#define RESETSERCLK GPIOA -> BSRR |= 1 << (2 + 16)
#define SETSERCLK GPIOA -> BSRR |= 1 << 2

#define RESETRCLK GPIOA -> BSRR |= 1 << (16 + 1)
```

```
#define SETRCLK GPIOA -> BSRR |= 1 << 1

void Cube::printFrame()
{
for (uint8_t z = 0; z < 8; ++z)
{
	uint8_t currentLayer = 1 << z;

	RESETRCLK;

	for (uint8_t i = 0; i < 8; ++i)
//Посылаем байт с текущим слоем в сдвиговый регистр
	{
		RESETSERCLK;
		GPIOA -> BSRR |= 1 << (16 *
((currentLayer >> i) & 1));
		SETSERCLK;
	}

	for (uint8_t x = 0; x < 8; ++x)
	{
		for (uint8_t y = 0; y < 8; ++y)
		{
			RESETSERCLK;
			GPIOA -> BSRR |= 1 <<
(16 * ((m_OutFrame[7 - z][x] >> y) & 1 ));
			SETSERCLK;
		}
	}
	SETRCLK;
```

}

}

Теперь подробно о том, как это работает. Для того чтобы поднимать или опускать выводы можно использовать функцию HAL_GPIO_WritePin, но у этой функции есть один недостаток: она делает некоторые проверки что само по себе медленнее чем запись напрямую в регистр ODR или BSRR. В данном случае лучше подойдет BSRR. На самом деле запись в регистр BSRR довольно простая. Структура этого регистра представлена на рисунке 17.



8.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Reset the corresponding ODRx bit
Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Set the corresponding ODRx bit

Рисунок 17 - Описание регистра BSRR (рисунок взят из Reference Man-

То есть, чтобы поднять выход в единицу надо в регистр записать единицу сдвинутую на n влево, где n – номер вывода, через побитовое ИЛИ. А чтобы опустить – записать в регистр единицу сдвинутую на 16+n. Теперь о том, как взять конкретно единицу или ноль из байта, который мы записываем. В цикле мы сдвигаем байт на i бит вправо и накладываем единицу через побитовое И, тем самым мы отбросили ту часть байта, с которой мы не работаем в текущей итерации цикла. Если бы куб был собран по схеме с общим катодом, то это число надо было бы инвертировать, но поскольку у нас схема с общим анодом, то логика будет обратной и инверсия не нужна. Теперь осталось получившееся число умножить на 16, прибавить к этому числу номер вывода (в случае вывода SER ничего не нужно прибавлять, тк его номер равен нулю) и

сдвинуть единицу на это количество бит влево (ровно это в коде и написано). Возвращаясь к передаче данных. Отрисовка происходит сверху вниз и каждый слой отрисовывается следующим образом. Самым первым должен передаваться байт, который включает определенный слой. Отправка этого байта происходит на строчке 26 – 30 по алгоритму, описанному выше. Затем с помощью вложенного цикла мы отправляем 8 байт состояний слоя по тому же алгоритму, что мы использовали для передачи байта-слоя, но байты должны быт инвертированы (строчки 33 – 41). И таким образом отрисовывается еще 7 слоев. Таким образом, используя регистр, удалось добиться увеличения скорости выполнения, что является достаточно критичным при динамической индикации.

2.3.1.2 Управление яркостью.

Для управления яркостью будем использовать значение, полученное с АЦП и передавать его в регистр для управления скважностью таймера, не используя какие-либо преобразования. Именно для этого во время конфигурации периферии значение counter period таймера два было указано равным 4095. Далее необходимо ввести понятие функции обратного вызова. Функция обратного вызова (Callback) – функция, которая передается в качестве параметра другой функции. В контексте прерываний это функция, которая вызывается после завершения обработки прерывания. Как раз-таки на момент выполнения ФОВ аналогово-цифровое преобразование завершилось. Поэтому можно написать следующее:

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)     //по
окончании измерения вызывается функция обратного вызова
{
   if(hadc->Instance == ADC2)                    //Если прерывание при-
шло от ADC2
   {
```

```
//преобразование не нужно, тк максимальные значения импульса и ацп равны
    TIM2 -> CCR4 = HAL_ADC_GetValue(&hadc1);  //Обращаемся к регистру CCRx, где x - номер канала таймера, для управления скавжностью
  }
```

Эту функцию не нужно нигде вызывать поскольку она вызывается сама изнутри обработчика прерывания, а предварительное объявление находится в библиотеки HAL. Перезапускать вручную измерение не нужно, поскольку оно происходит при переполнении таймера 3.

2.3.1.3 Генерация случайных чисел

Для генерации случайных чисел существует несколько методов:

- использование встроенной функции HAL, которая управляет модулем TRNG;
- Генерация шумов DAC;
- Использование генерации seed генератора случайных чисел, встроенного в C++, с помощью АЦП (шумы на выводе контроллера или значение температурного датчика);

Использовать HAL функцию и DAC невозможно, поскольку в STM32F103C8T6 нет ни того, ни другого, поэтому будет использоваться третий метод. Если бы программа была написана для ПК, то язык Си без нашего вмешательства использовал бы время системы Windows (RTC на материнской плате, если быть до конца откровенными) в качестве источника энтропии генератора случайных чисел, а затем srand(time_t) во время каждого запуска программы, но поскольку время использовать не представляется возможным, то будем использовать значение с температурного датчика. Т.е., вместо srand(time_t), будем использовать srand(HAL_ADC_GetValue(&hadc1)). И,

чтобы каждый раз по прерыванию не переопределять seed генератора случайных чисел, в соответствующем ФОВ будем выключать ADC. Затем перед бесконечным циклом в функции main необходимо написать следующее:

```
HAL_ADC_Start_IT(&hadc1);
```

Стоит отметить, что калибровку проводить не нужно.

В ФОВ пишем следующее:

```
else if(hadc->Instance == ADC1)
{
    srand(HAL_ADC_GetValue(&hadc1));          //по завершении
```

считывания значения с 5-го канала ADC1, устанавливаем seed ряда случайных чисел

```
    HAL_ADC_DeInit(&hadc1);
}
```

HAL_ADC_DeInit нужен для того, чтобы отключить АЦП2, чтобы seed не перезаписывался. На рисунке 18 показано, что ФОВ АЦП1 срабатывает 1 раз.

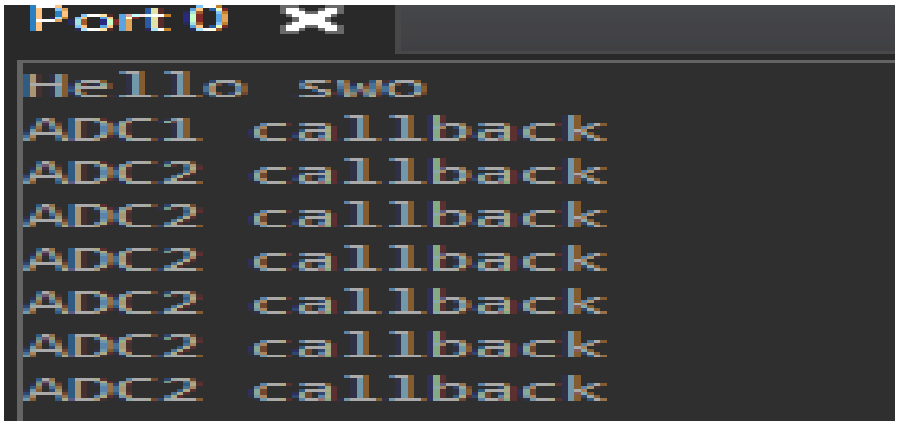


Рисунок 18 - Демонстрация единственного срабатывания АЦП1

Теперь можно написать функцию, которая будет давать нам случайное число в заданном диапазоне, потому что rand() всегда возвращает 32-битное число. Для того чтобы распределить 32-битное число в диапазон можно воспользоваться или делением всего диапозона на части, или использовать остаток от деления. Первый вариант нам не очень подходит, потому что у нашего контроллера нет блока для вычислений с плавающей точкой (о наличии такого

блока можно прочитать в технической документации, используя поиск слова FPU).

```
#include "main.h"
// Генерируем случайное число между значениями min и max
// Предполагается, что функцию srand() уже вызывали
int getRandomNumber(int min, int max)
{
   // Равномерно распределяем случайное число в нашем диапазоне
   return (rand() % (max - min + 1)) + min;
}
```

На рисунке 19 показано, что числа действительно генерируются случайно и при перезапуске контроллера ряд отличается. Диапазон случайных чисел от 0 до 7.

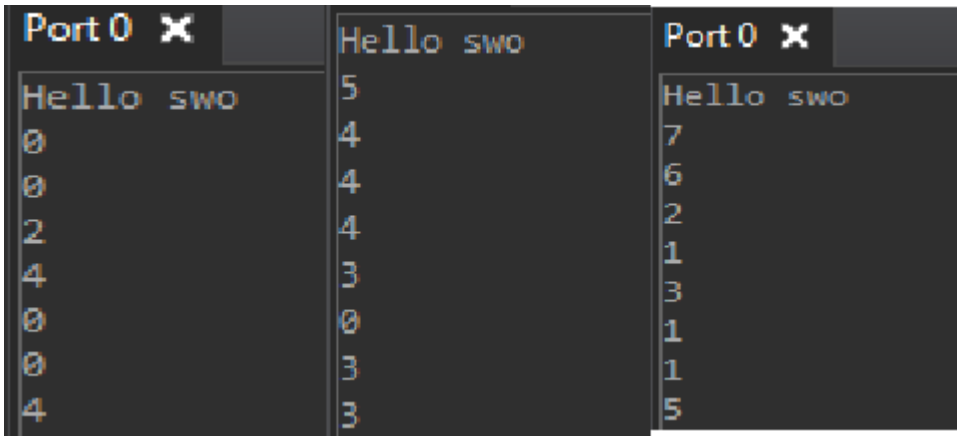


Рисунок 19 - Демонстрация генерации действительно случайных чисел

2.3.1.4 Режимы и их переключение

Почти все режимы, написанные для данного проекта, основаны на базовой математике и используют случайные числа и в подробном описании не нуждаются, чего нельзя сказать о том, как эти режимы переключаются и вызываются. Самым простым способом вызовов любых режимов является конструкция switch-case. Единственная проблема этой конструкции заключается в том, что при увеличении количества режимов, время вызова режима также

увеличивается. Для решения этой проблемы необходимо использовать указатели на метод класса. Но перед этим рассмотрим указатели на функции, а затем рассмотрим, как на самом деле работают методы класса в C++.

Поскольку каждая функция обладает своим адресом в памяти, можно создать переменную, которая содержит адрес этой функции. В свою очередь, переменная, которая содержит адрес чего-либо, называется указателем. Тип указателя строится по следующей конструкции: [тип возвращаемого значения] (*<имя указателя>)({список аргументов}).

Теперь в этот указатель можно записывать адреса функций, которые имеют такой же тип возвращаемого значения и список аргументов. Вызов функций через указатель происходит так же как и вызов обычной функции (<имя функции>({список аргументов});). Однако обычно используют переопределение типа для лучшей читаемости. Для этого необходимо воспользоваться следующей конструкцией: typedef [тип возвращаемого значения] (*<имя нового типа>)({список аргументов}). Затем, когда необходимо создать указатель на функцию, мы создаем переменную, аналогично переменной с любым другим типом.

Для того, чтобы понять, что такое указатель на метод класса, как его создать и как вызывать метод через него, необходимо понять, как вызываются методы в C++.

Для вызова метода определенного экземпляра класса, используется оператор выбора члена (<экземпляр>.<метод>({список аргументов});). Но на самом деле метод класса является функцией, который имеет скрытый первый аргумент в виде указателя на экземпляр класса, метод которого мы вызываем (т.е. <метод>(&<экземпляр>, {список аргументов});). При чем тип первого аргумента является именем класса. Поэтому очевидно, что если метод является функцией, то его можно вызывать через указатель образом, похожим на вызов обычной функции через указатель. Конструкция определения указателя метода класса выглядит следующим образом: [тип возвращаемого значения] (<имя класса>::*<имя указателя>)({список аргументов}). И конструкция для

переопределения типа: typedef [тип возвращаемого значения] (<имя класса>::*<имя нового типа>)({список аргументов}). Вызов метода происходит как (<экземпляр>.*<имя указателя>)({список аргументов}). Таким образом внутри класса для переключения режимов было написал следующее:

typedef void (Cube::*ModePtr)(); //тип указателя, который указывает на метод текущего режима, который должен вызываться в обработчике таймера

ModePtr m_currentModePtr = &Cube::lightCube; //по умолчанию указывает на режим светящегося куба

Затем в бесконечном цикле вызывается режим через указатель на метод.

Переключение режимов происходит в соответствующем методе путем перезаписывания указателя (См. приложение Д, файл Cube.cpp метод Cube::ChangeMode).

2.3.2 Написание кода для МПС пульта управления

Обычно в самом начале конфигурации настраивается интерфейс отладки и тактирования, но в данном случае сначала настроим интерфейс отладки, а тактирование настроим позже.

Как было сказано выше, в данном случае для обработки кнопок используются внешние прерывания. Как и внутренние прерывания, внешние имеют соответствующий обработчик, регистры конфигурации, источник события и приоритетность по умолчанию (об этом можно узнать в технической документации в разделе Interrupts and events в таблице векторов прерываний). Блок внешних прерываний в контроллере, который используется в данном проекте, имеет 19 линий внешних прерываний. К первым 16 линиям можно присоединить 16 портов ввода-вывода руководствуясь рисунком 20.

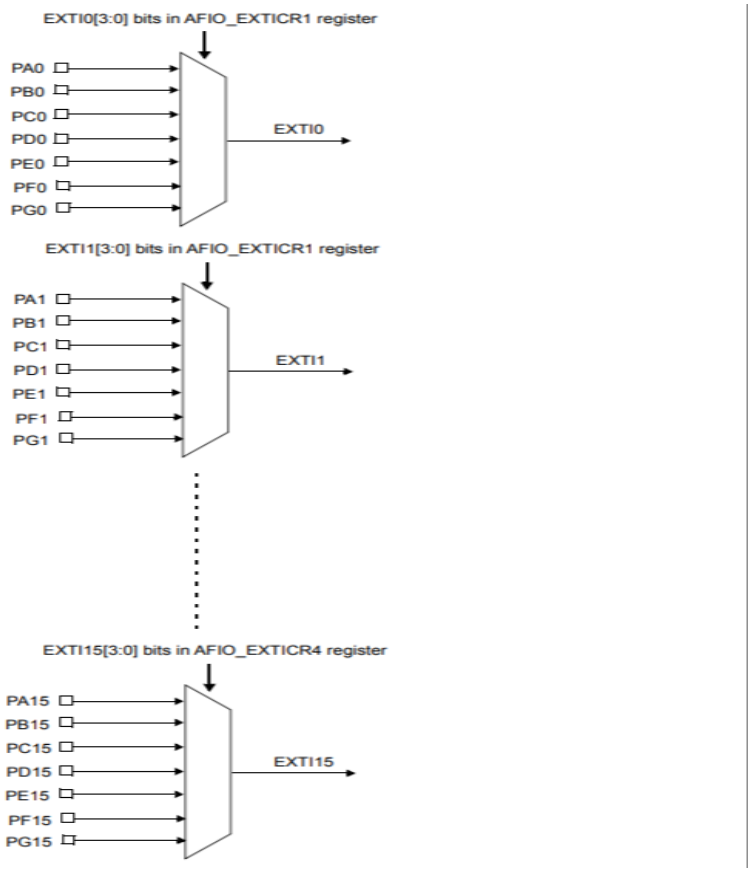


Рисунок 20 – порты ввода-вывода, формирующие линию внешних прерываний (рисунок взят из Reference manual)

То есть любой порт ввода вывода может быть подключен к линии внешних прерываний с таким же номером. При чем 2 вывода с одинаковым номером, но на разных портах не могут быть подключены одновременно к линии внешних прерываний, поскольку соответствующие поля в регистрах AFIO_EXTI являются адресами для внутренних мультиплексоров.

Источником внешних прерываний является фронт сигнала. При чем в конфигурации можно выбрать по какому именно фронту (нарастающий, спадающий, оба сразу) должно сработать прерывание.

Для конфигурации внешних прерываний в STM32CubeMX необходимо нажать на необходимый вывод контроллера и в выпадающем списке выбрать GPIO_EXTIx, где x соответствующий номер линии прерывания. Затем необходимо зайти в меню System Core -> GPIO и настроить все выводы (подтяжка

к питанию, переопределение имен выводов для удобства, interrupt event – falling edge trigger detection). В том же окне необходимо выбрать вкладку EXTI и включить прерывания от всех присутствующих линий. Здесь можно заметить, что выводов выбрано 9, а линий прерываний (а соответственно и обработчиков) всего 6 (Для первых 5 у каждой своя линия, а у линий 5…7 – общая). Причина, по которой NVIC так устроен, известна только инженерам, разработавшим ядро CortexM, но для разработки важно только понимать, как обрабатывать линии внешних прерываний с 5 по 9 независимо друг от друга внутри одного обработчика.

Для упрощения трассировки печатной платы порты ввода вывода для внешних прерываний были выбраны на одной стороне микроконтроллера, кроме вывода для EXTI8.

Для измерения напряжения батареи используется АЦП, который подключен к средней точке делителя напряжения. Настройки АЦП и таймера аналогичны настройкам на основной МПС, за исключением того, что период таймера равен почти 4 секунды.

Bluetooth модуль подключается точно таким же образом, как и в основной МПС.

Для дисплея необходимо настроить SPI. Поскольку выводы SPI1 заняты выводами внешних прерываний, остается использовать SPI2. Настройки нужно оставить по умолчанию. Также для дисплея дополнительно необходимо настроить выводы RS, DC и CS как GPIO_OUTPUT.

Однако использование SPI само по себе является довольно емким процессом при передачи большого количества данных (в нашем случае как раз передается большое количество данных, так как цвет каждой точки экрана требует передачи 16 бит), потому что за один раз блок SPI может передать 8 или 16 бит, затем после передачи, в соответствующий регистр необходимо записать новый пакет данных. И этим обычно занимается ядро процессора, однако в контроллерах семейства STM32 есть блок DMA, который может сам брать информацию из одной ячейки памяти и записать ее в регистр периферии или

обратно. При чем процессорное ядро запускает блок DMA, а затем блок DMA самостоятельно посредством запросов производит пересылку до 4 КБ данных из памяти в периферию или обратно множеством пакетов данных с необходимым размером.

Для настройки DMA в настройке SPI или System Core -> DMA необходимо добавить запрос, нажав кнопку add. Затем в поле DMA Request необходимо выбрать SPI2_RX, Channel оставить без изменений, Direction выбрать Memory To Peripheral, Priority оставить без изменений. Тут важно отметить, что это приоритет не NVIC, а арбитра DMA. На самом деле у DMA есть множество каналов, и он может обрабатывать только один канал одновременно. Для того чтобы решить ситуации, когда два блока периферии сообщают DMA о том, что готовы к пересылке новых данных, существует арбитр. Он работает аналогично NVIC и имеет схожую таблицу с приоритетами по умолчанию. В дополнительных настройках запроса необходимо установить режим normal, Data Witdh = byte как в блоке памяти, так и периферии, и поднять флаг инкремента адреса памяти.

Теперь вернемся к настройкам тактирования. Все дело в том, что DMA тактируется с такой же частотой, как и ядро, а SPI2 в данном микроконтроллер тактируется шиной, максимальная частота которой равна 36 МГц. Если выставить максимальную частоту тактирования контроллера, то SPI не будет успевать отправить все данные, которые DMA пересылает между памятью и SPI. Поэтому частота ядра должна совпадать с частотой SPI, и она равна 36 МГц.

На этом конфигурации контроллера завершена и можно генерировать код.

2.3.2.1 Обработка кнопок

Для понимания того, как необходимо обрабатывать кнопки, необходимо понимать, что механические кнопки обладают дребезгом. То есть на самом деле кнопка переключает состояние выхода не мгновенно и не прямолинейно,

что может приводить к ложным срабатываниям. Для решения проблемы дребезга, существует как аппаратные, так и программные решения.

Аппаратные решения основаны на ФНЧ в виде фильтра из резистора и конденсатора. На рисунке 21 представлено подключение кнопки с обвязкой, которое используется в отладочных платах STM32 Nucleo.

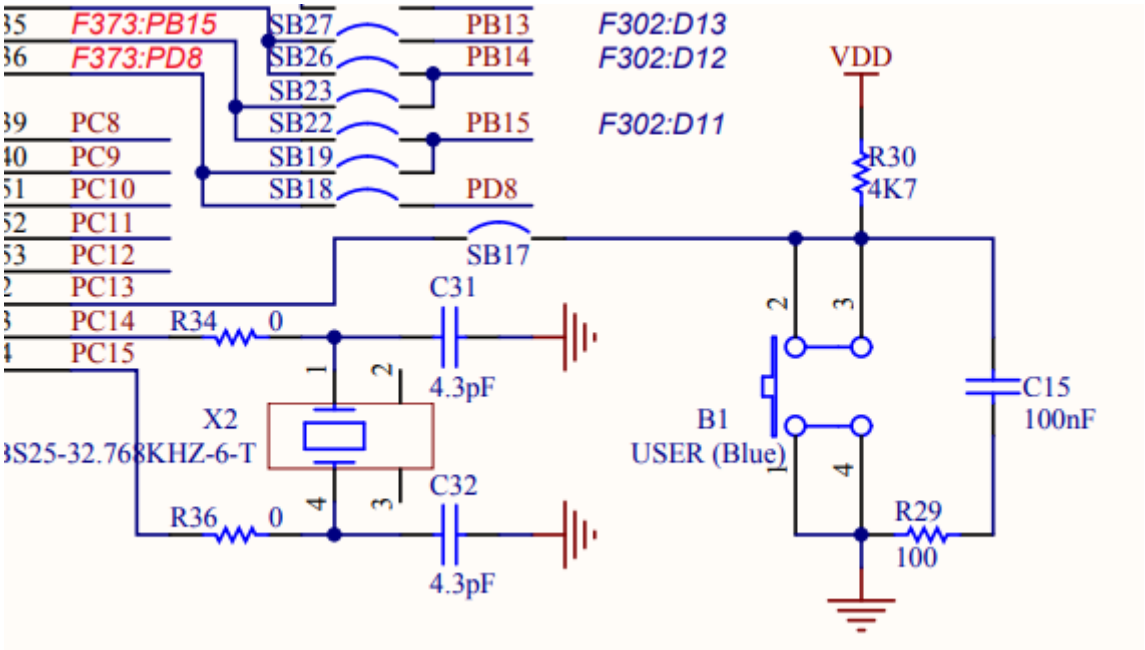


Рисунок 21 – подключение кнопок на платах STM32 Nucleo (рисунок с сайта производителя плат Nucleo)

Существуют также схемы, с использованием триггера Шмидта, которые на самом деле также основаны на RC фильтре.

Программные решения состоят в том, чтобы не считывать кнопку пока дребезг контакта не закончится. В данном проекте используется только программная обработка дребезга.

Как было сказано выше, кнопки подключены к внешним прерываниям. Поэтому разберем обработчики прерываний и ФОВ. Сначала необходимо выделить 2 переменных uint16_t для флагов, которые показывают состояние всех кнопок, при чем одна из переменных будет означать, что кнопку необходимо обработать и отправляться по UART в основную МПС, а также использоваться для меню, а вторая является вспомогательной и показывает факт того, что об-

работка дребезга началась. Так же необходим массив из 9 ячеек uint32_t, который необходим для хранения последнего времени нажатия на каждую из кнопок.

В обработчике прерывания первых 5 линий прерываний фиксируется время срабатывания прерывания с помощью функции HAL_GetTick и записывается в соответствующую ячейку массива. Затем вызывается ФОВ, общая для всех линий прерываний. В ней через побитовое ИЛИ записывается единица в соответствующий бит обоих флагов, а затем выключается линия прерываний на соответствующей линии и отправляется основная переменная с флагами по UART. Затем, в основном цикле программы, спустя 120 мс опустится вспомогательный флаг и включится соответствующая линия прерываний (функция Post_Processing_Buttons).

С обработчиком EXTI9…5 немного другая ситуация. Здесь необходимо проверить 4 бита регистра EXTI->PR, который показывает, какая конкретно линия вызвала прерывание. И когда первый попавшийся бит равен единице, происходит такая же обработка, как и в случае первых 5 линий прерываний.

Стоит отметить, что ввиду того, что контроллер прерываний является вложенным, при срабатывании нескольких кнопок, ни одна не будет утеряна, поскольку вызовется такое же количество обработчиков.

2.3.2.2 Вывод изображения на экран

Для вывода изображения за основу взята библиотека пользователя Basic-Code сайта GitHub, которая называется ILI9225. На самом деле в проекте осталась только инициализация дисплея (хотя некоторые части инициализации переписаны для корректной работы), а все функции переписаны. В данном блоке рассмотрим передачу данных на дисплей, а не формирование данных. Для того, чтобы отправить любые данные в дисплей, необходимо выбрать устройство, к которому будет происходить передача данных, путем установки вывода CS в логический ноль, затем установить вывод DS в зависимости от того,

записываются в дисплей данные или команды. Затем заставить DMA отправлять данные в SPI, а когда передача данных завершится, вызовется прерывание, сообщающее об окончании передачи, и в ФОВ прерывания SPI необходимо установить CS в логическую единицу. Процессы, описанные выше, происходят в функциях lcd_write_register, lcd_write_command, lcd_write_data и spi_write.

Как было сказано выше, к сожалению, за одну транзакцию нельзя передать больше 4КБ данных, поэтому все данные, размер которых больше 4 КБ, делятся на части и отправляются по очереди.

2.3.2.3 Пользовательский интерфейс

Интерфейс в данном проекте не обладает высокой степенью вложенности, поэтому, по нажатию кнопки «ОК» происходит инверсия флага режима и, как в основной МПС, изменяется указатель на функцию, а затем нужная функция вызывается через него.

Первый режим вызывается при включении и показывает текущий режим и время между изменением кадра режима в миллисекундах. При нажатии кнопок «RB» и «RU» время уменьшается и увеличивается соответственно. На рисунках 22, 23 отображен первый режим меню.

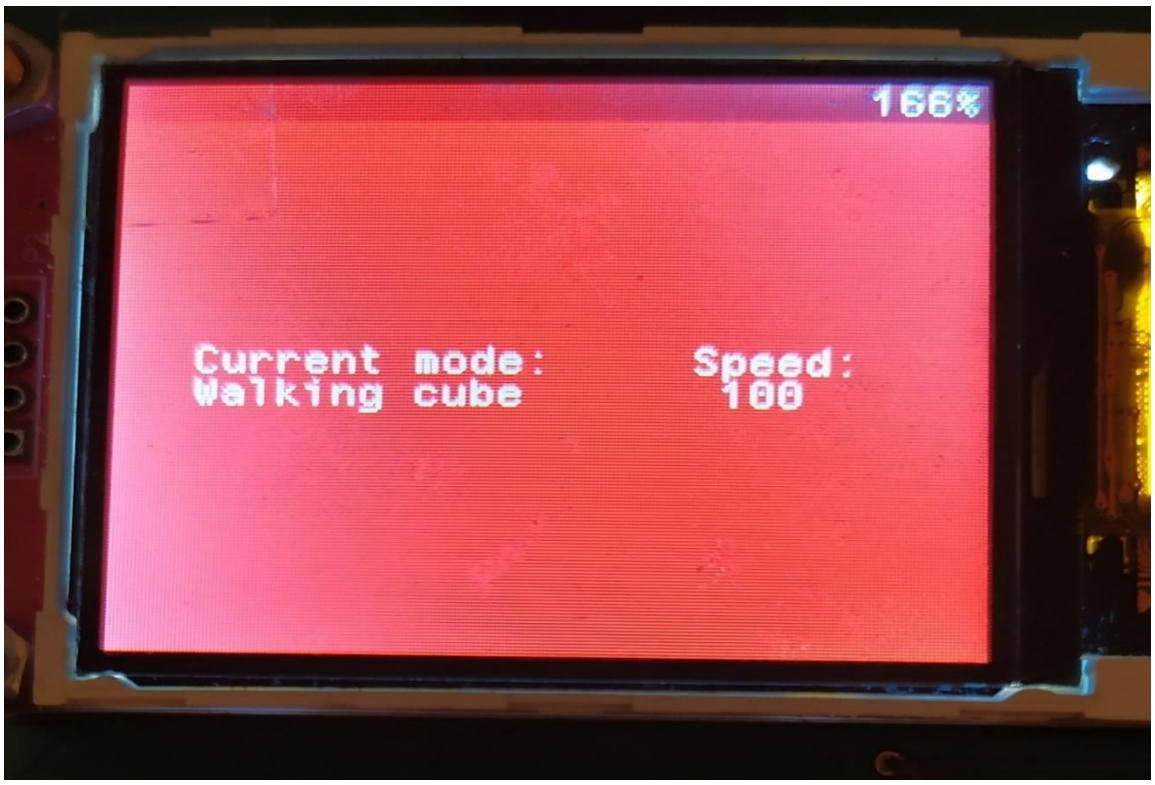


Рисунок 22 – состояние экрана в момент включения

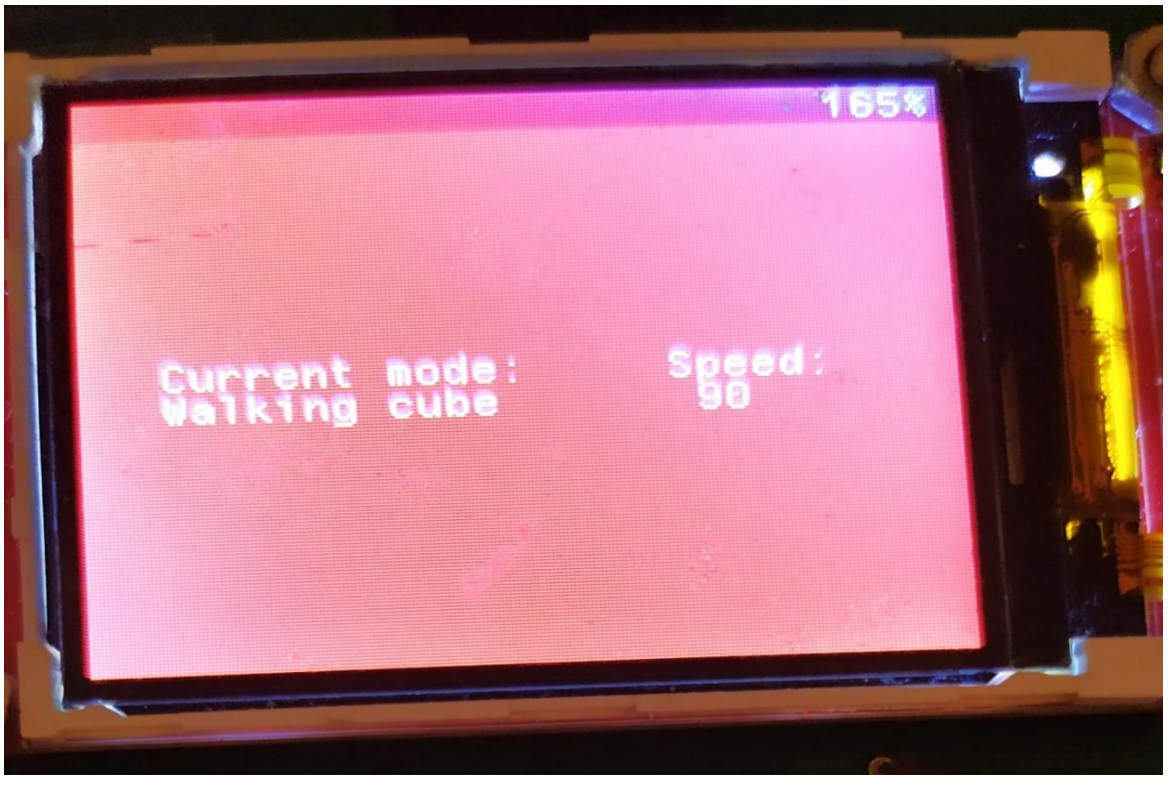


Рисунок 23 – реакция экрана на нажатие кнопки «RU»

После нажатия на кнопку «OK» режим меню переключится на режим переключения режима куба. В этом режиме выводится весь список режимов с

курсором в центре экрана. При нажатии кнопок «LU» и «LB» список перемещается в кольцевом режиме. После нажатия кнопки «ОК» настройка применяется и первый режим перестраивается соответствующим образом. На рисунках 24 – 26 демонстрируется смена режима (исходное состояние представлено на рисунке 23).

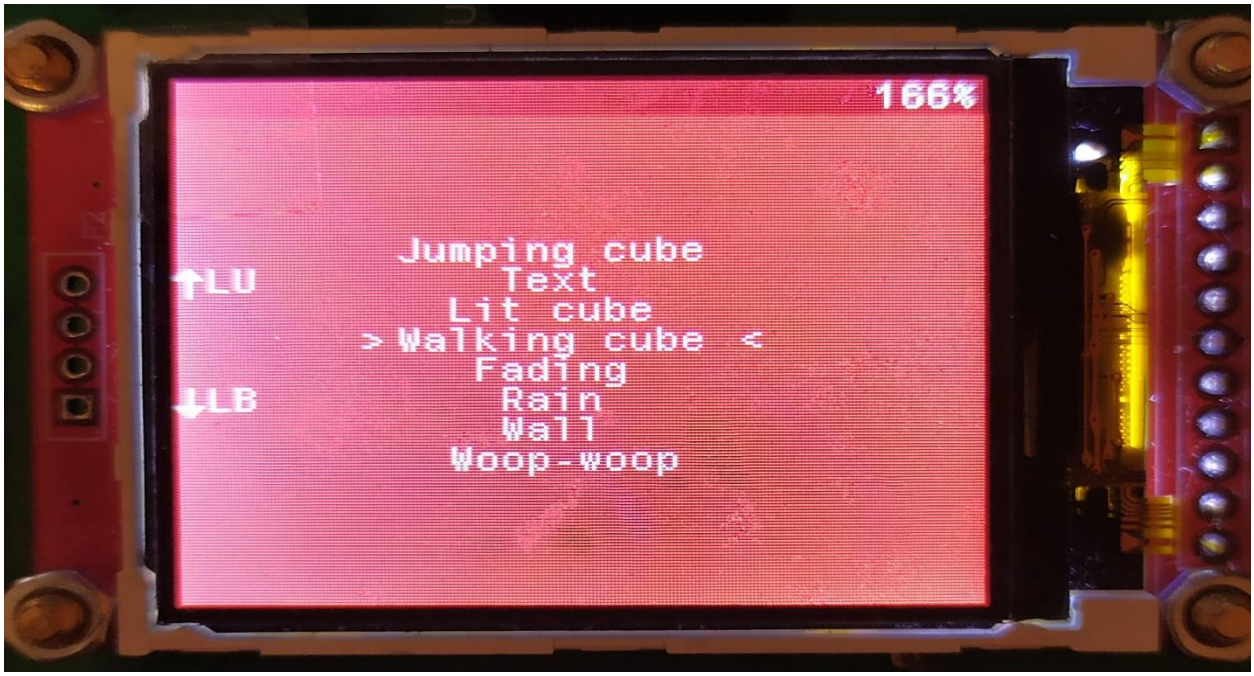


Рисунок 24 – реакция экрана на нажатие кнопки «ОК»

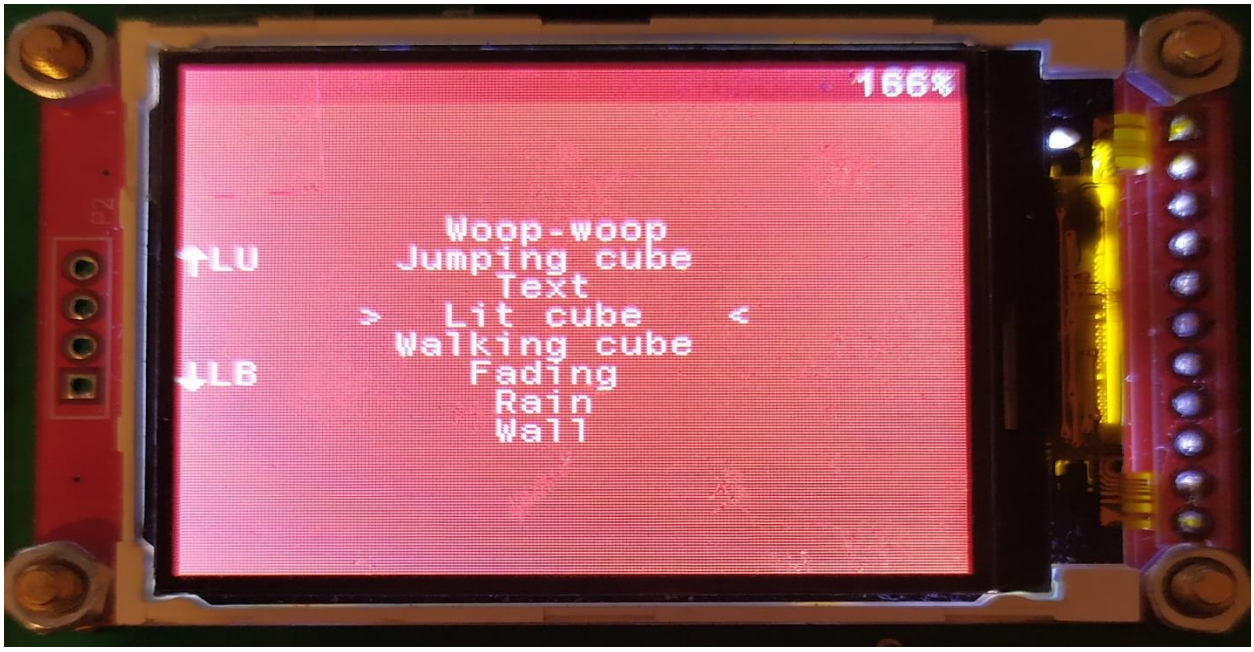


Рисунок 25 – реакция экрана на нажатие кнопки «LU»

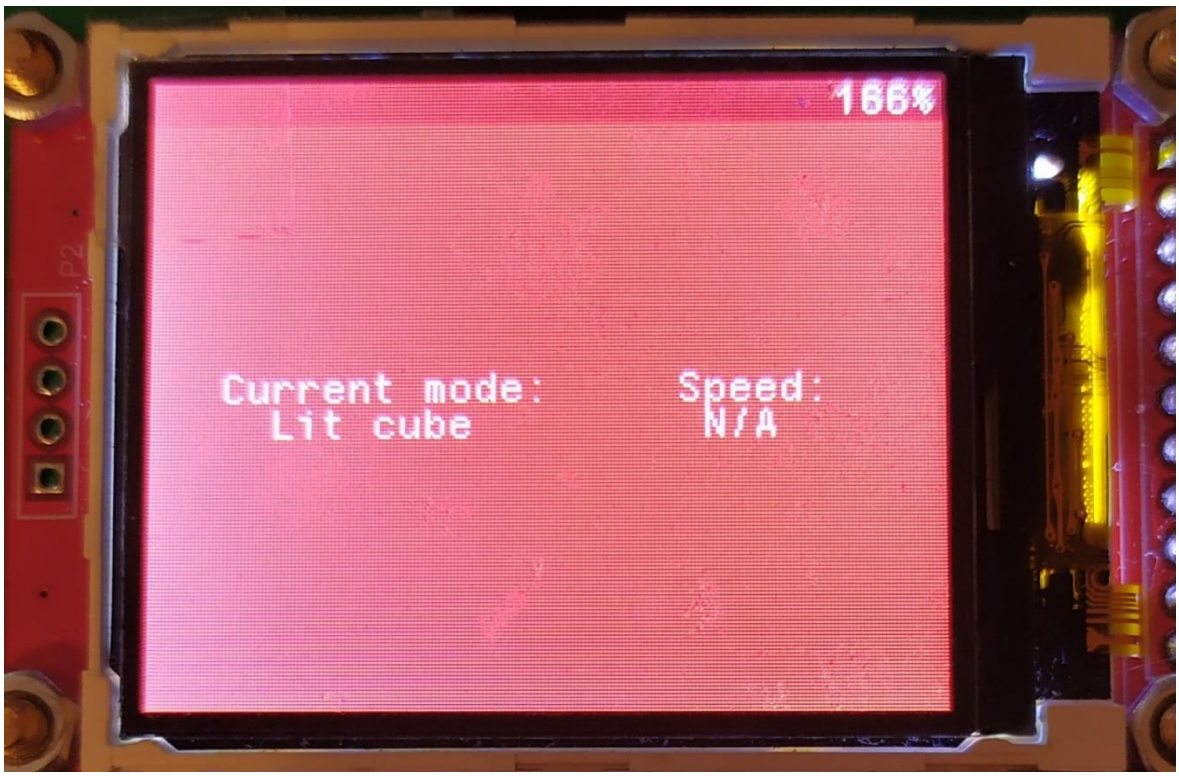


Рисунок 26 – демонстрация подтверждения выбора режима

2.3.2.4 Измерение напряжения аккумулятора

Номинальное напряжение аккумулятора 18650 находится в пределах от 3,3 до 4,2 В, поэтому измерение должно происходить через делитель напряжения. Опытным путем были найдены значения АЦП для напряжения батареи 3,3 и 4,2 В, затем полученные значения используются для преобразования получаемого напряжения аккумулятора в процент заряда, который выводится в правом верхнем углу.

Для того, чтобы измерение началось при запуске, а не когда таймер переполнится, необходимо принудительно вызвать генерацию события написав перед бесконечным циклом следующее:

TIM3->EGR = TIM_EGR_UG;

Регистр EGR используется как самим таймером, так и программным путем для того, чтобы дать понять блоку NVIC, что произошло некоторое событие, связанное с конкретным таймером. В данном случае в поле Update Generation этого регистра записывается единица. В свою очередь единица в этом поле является источником начала измерения АЦП. Измерение происходит не

в ФОВ, а в самом конце итерации бесконечного цикла, в случае, когда поднят соответствующий флаг.

На этом решение базовых задач при программировании МПС пульта управлении и приготовления к соединению обеих МПС путем беспроводной передачи данных завершены. Полный код программы для МПС пульта управления представлен в приложении Е.

2.3.3 Передача данных между двумя МПС

Рассмотрим прием данных от джойстика, обработку и обратный ответ, который будет использоваться джойстиком для отображения статуса соединения.

Сначала необходимо настроить модули Bluetooth HC05 с помощью любой терминальной программы и моста UART-USB путем отправки AT команд в соответствующем режиме. Переход в этот режим осуществляется или удерживанием кнопки на модуле при включении или при подаче высокого логического уровня на соответствующий вывод. Модули настраиваются следующим образом:

Модуль для МПС пульта ДУ:

Роль – Master

Скорость передачи – 115200

Стоп-бит – 1

Контроль четности – нет

Адрес slave устройства – адрес модуля для основной МПС

Модуль для основной МПС:

Роль – Slave

Скорость передачи – 115200

Стоп-бит – 1

Контроль четности – нет

Модули HC05 передают данные в управляющее устройство (микроконтроллер в данном случае) по шине UART, соответственно, поскольку блок

UART в STM32 имеет свое место в таблице векторов прерываний, логичным является прием двух байт флагов нажатий в массив по прерыванию в соответствующей ФОВ. Для этого перед бесконечным циклом вызывается HAL функция, которой необходимо передать ссылку на структуру UART, указатель на буфер и размер буфера:

HAL_UART_Receive_IT(&huart3, IsButtonPressedRxBuff, 2);

В ФОВ полученный массив записывается в соответствующую переменную через побитовое ИЛИ, чтобы случайно не опустить флаг до его обработки. Затем, в этом же месте, отправляется номер режима в качестве ответа, и вновь вызывается функция HAL_UART_Receive_IT. Стоит отметить, что в данном случае эти данные являются лишь подтверждением того, что основная МПС получила данные, и не влияют на отображаемый режим на экране. Однако, чтобы была функция синхронизации после возобновления соединения, при нажатии кнопки LL в режиме отображения текущего режима поднимаются соответствующие флаги, и основная МПС только отправляет свой текущий режим и меняет режим на тот же, чтобы сбросить таймер режима. Джойстик же получает режим и использует его для обновления отображаемых данных.

Дальнейшая обработка флагов со стороны основной МПС аналогична обработке флагов со стороны джойстика и была описана в пункте 2.3.2.3, за исключением того, что внутри меню вызываются нужные методы класса Cube, вместо функций отрисовки дисплея.

В МПС джойстика в ФОВ внешних прерываний сразу отправляется 2 байта флагов состояний нажатий и запускается таймер. Затем, в течение примерно 500 мс вызовется две ФОВ – блока UART и таймера. В первом поднимается флаг, сигнализирующий о том, что ответ пришел, и, если была нажата кнопка LL, происходит обработка приема данных синхронизации. Во второй ФОВ таймер останавливается, и, если сработала первая ФОВ, происходит отрисовка иконки статуса соединения по Bluetooth, иначе это место затирается фоном.

На этом написание кода завершено. Полный код для обеих МПС представлен в приложениях Е (основная МПС) и М (МПС пульта ДУ).

На рисунках 27 – 31 представлена работа некоторых режимов устройства.

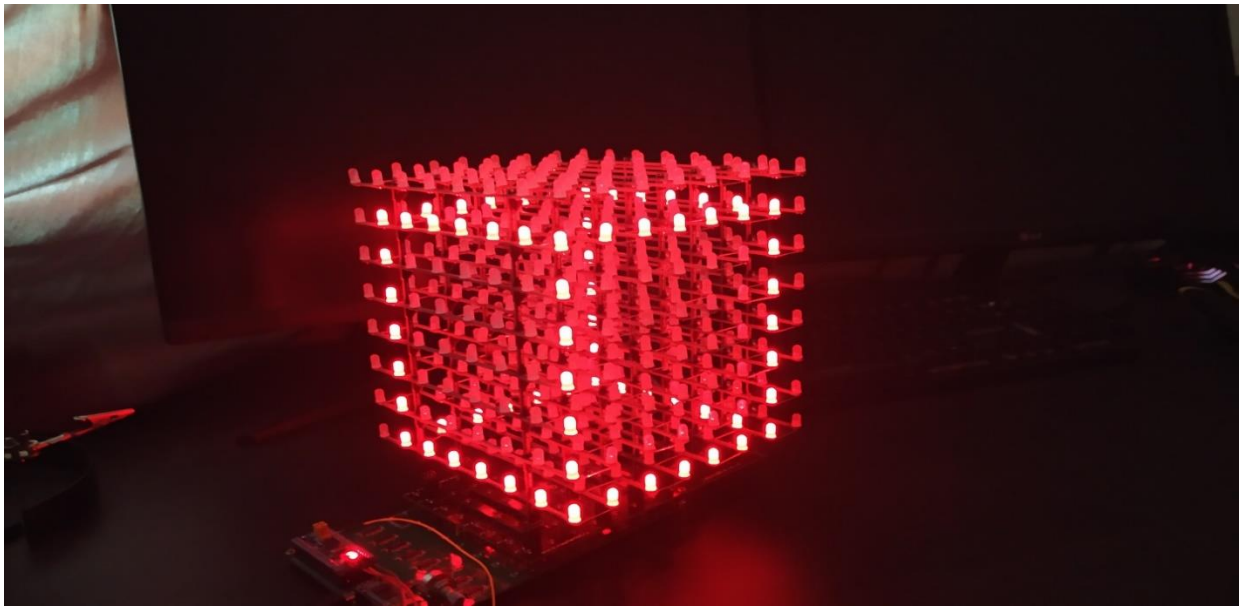


Рисунок 27 – демонстрация режима «Сжимающийся куб»

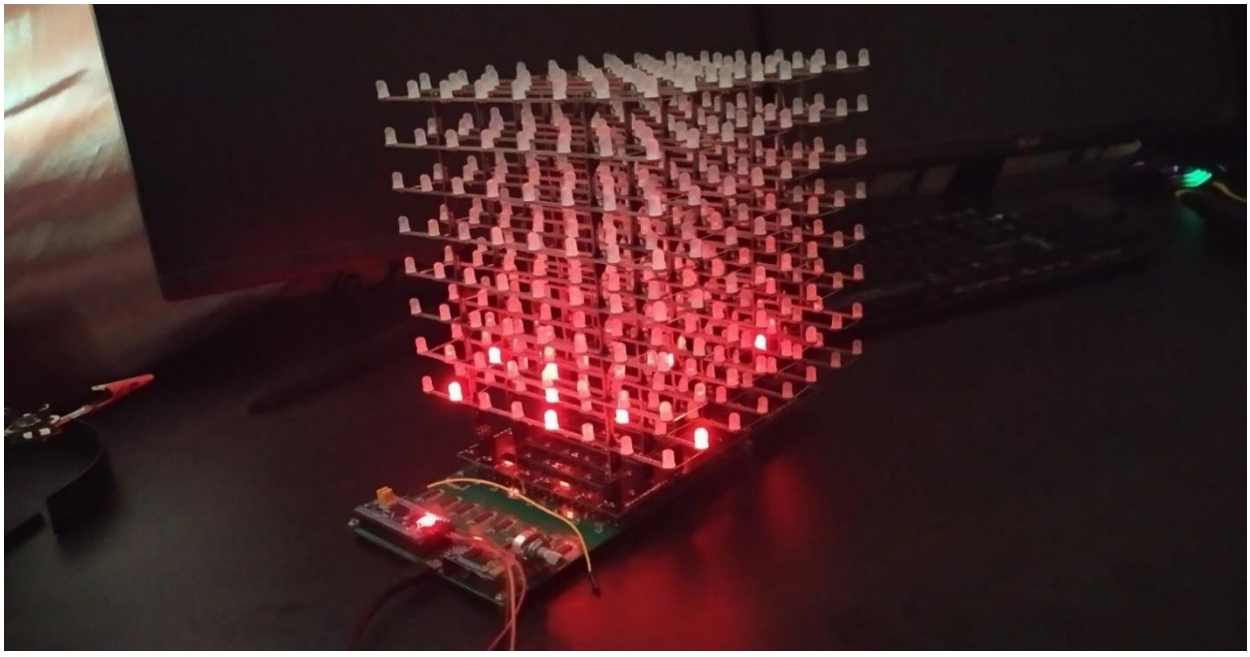


Рисунок 28 – демонстрация режима «Дождь»

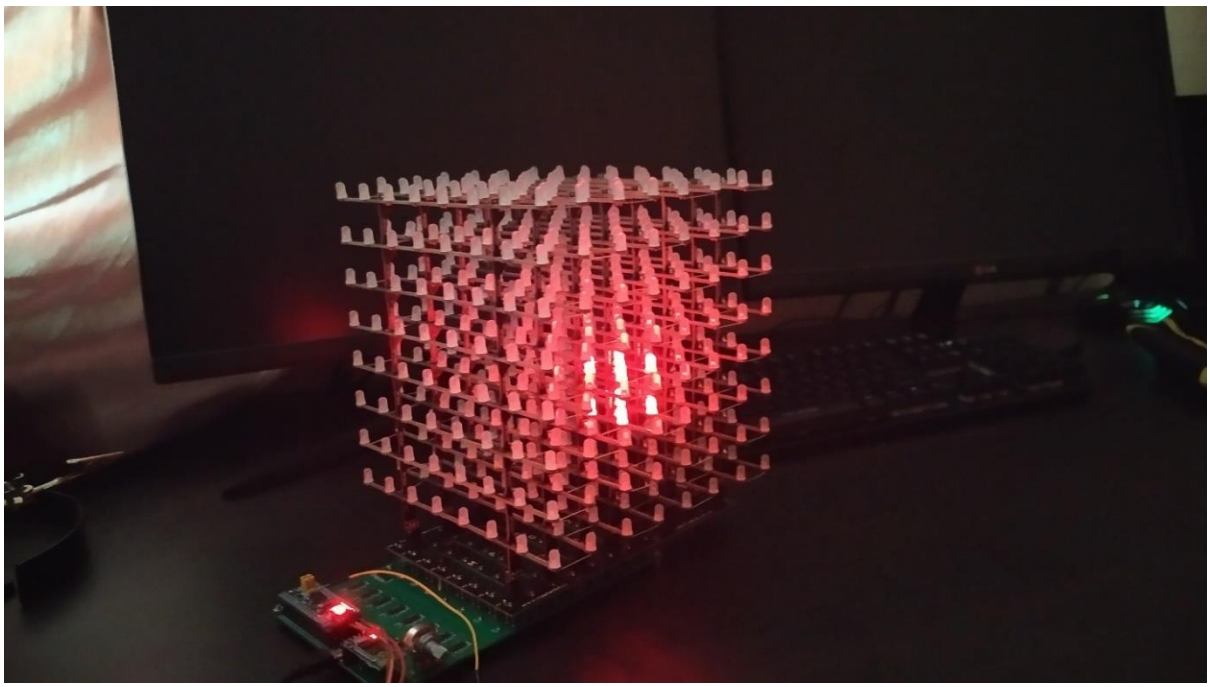


Рисунок 29 – демонстрация режима «Летающий кубик»

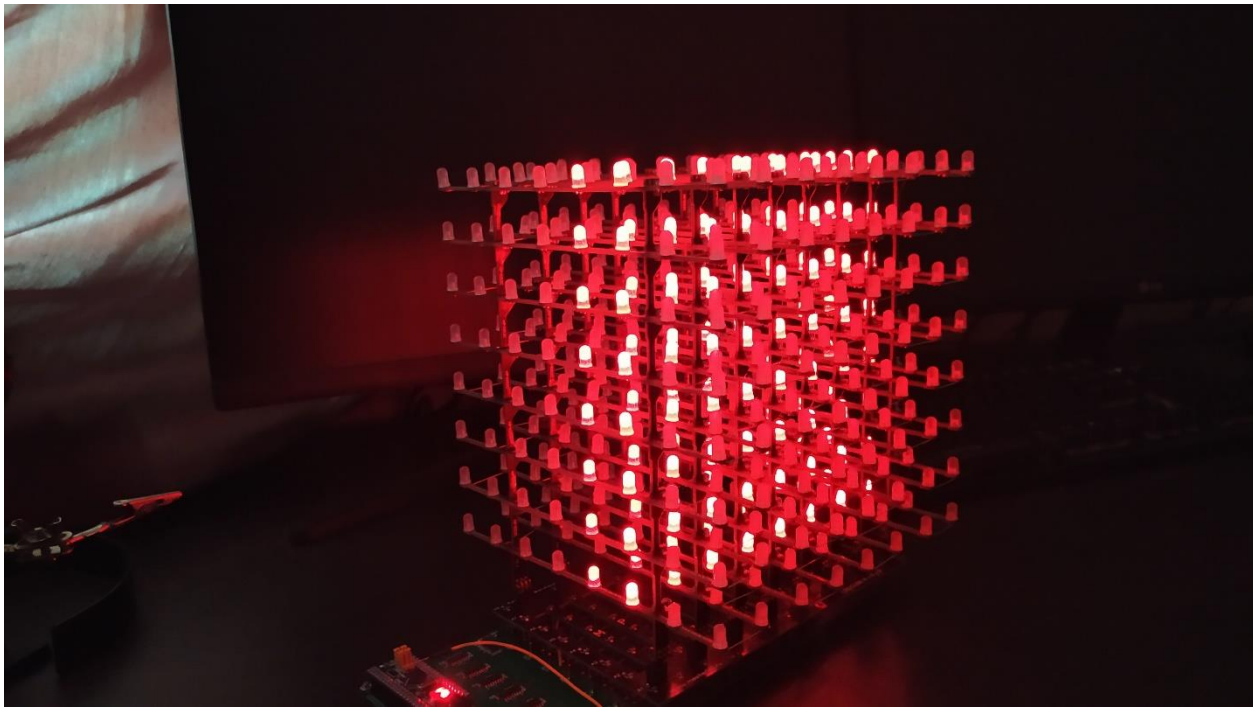


Рисунок 30 – демонстрация режима «Стена»

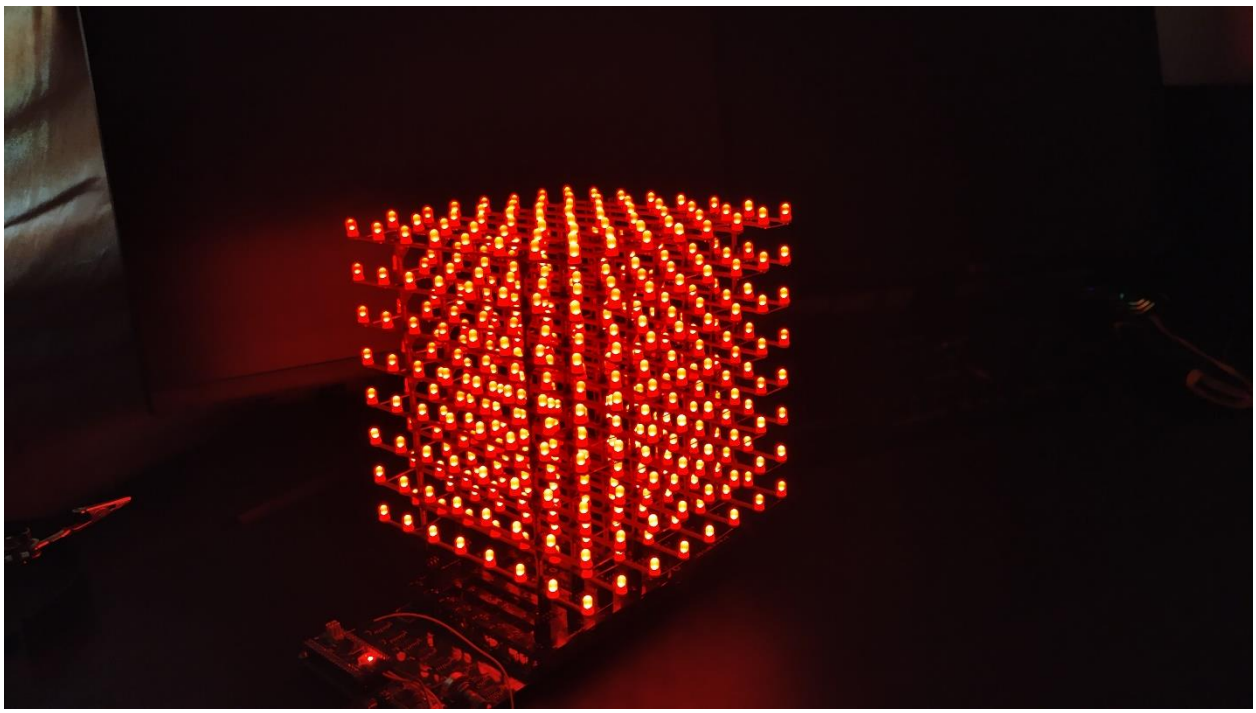


Рисунок 31 – демонстрация режима «Заполненный куб»

3. Экономическая целесообразность проекта

В экономике одним из основных понятий является «экономическая эффективность» хозяйственных мероприятий.

Теория эффективности четко разграничивает понятия эффекта и эффективности, понимая под первым – результат мероприятия, а под вторым – соотношение эффекта и затрат, которые его вызвали.

Эффект (от лат. effectus – исполнение, действие) означает результат, следствие каких – либо причин, действий. Эффект может измеряться в материальном, социальном, денежном выражении.

Экономический эффект – выраженный в стоимостной (денежной) форме результат каких-либо действий

Хотя по самому определению результат и эффект являются родственными понятиями (эффект – определенная форма результата), потребности экономической практики вынуждают в ряде случаев разграничивать указанные термины. При этом под условным понятием «экономический результат» обычно подразумевают общий (брутто) результат (в частности, выручка, доход), а под понятием «экономический эффект» – чистый (нетто) результат (в частности, прибыль).

Принципиальная взаимосвязь между указанными двумя понятиями может быть выражена формулой

Э = Р – З

Где Э – величина условного экономического эффекта;

Р – величина условного экономического результата;

З – полные затраты по мероприятию, вызвавшему эффект.

Эффективность определяется отношением результата (эффекта) к затратам, обеспечившим его получение.

Эффективность чаще всего характеризуется относительными показателями, которые рассчитываются на основе двух групп характеристик (параметров) – результата и затрат. Это, впрочем, не исключает рассмотрения в системе

показателей эффективности и самих абсолютных значений исходных параметров.

Для расчета экономической целесообразности разработки дипломного проекта и определения экономической эффективности капиталовложений в проект проведем технико-экономическое обоснование разработки, анализ существующих аналогичных разработок и определим экономический эффект от использования проектируемого устройства.

Расчет стоимости основных материалов, затраченных на создание проекта, представлен в таблице 5

Таблица 5 - Расчет затрат на материалы

Группа компонентов	**Кол-во элементов**	**Цена, руб/ед**	**Сумма, руб.**
Печатные платы	1	450	450
Пассивные компоненты (резисторы и конденсаторы)	106	0,4	42,4
Пассивные компоненты (катушки индуктивности)	1	30	30
Пассивные компоненты (переменное сопротивление)	1	50	50
Микросхемы (цифровые и аналоговые)	18	Варируется от 30 до 120	855
Активные компоненты (диоды и транзисторы)	10	0,6	6
Активные компоненты (светодиоды)	512	1,62	830
Активные компоненты (TFT экран)	1	320	320
Разъемы	2	20	40
ИТОГО			2583,4

Таким образом, себестоимость проектируемого устройства: З = 2583,4 рубля.

Стоимость существующих аналогичных разработок на российском рынке, например, masterkit LED CUBE 8x8x8 Nano равна Р = 3920 рублей.

Экономический эффект от использования проектируемого устройства равен:

Э = Р – З

Э = 3920 - 2583 = 1337 (руб).

4. Техника безопасности и охраны труда во время монтажных работ

Мероприятия по охране труда направлены прежде всего на создание безопасных условий труда, обеспечивающих всестороннюю защиту работающих, от опасных и вредных производственных факторов.

Безопасность работ на электроустановках обеспечивается правильной организацией труда, соблюдением технологии и безусловным выполнением всех требований техники безопасности.

Чтобы обезопасить работающих в цехе от поражения электрическим током, необходимо: включать и отключать оборудование в строгом соответствии с инструкцией по обслуживанию данного прибора или установки; производить какие-либо работы по ремонту и наладке оборудования (замена предохранителей и др.) только специальному наладочному персоналу;

При обнаружении проводов с плохой изоляцией прекратить работу и сообщить мастеру;

При работе на оборудовании с напряжением 220 В и выше постелить под ноги диэлектрический коврик;

Оберегать электропровод и аппаратуру от попадания на них воды, кислот, щелочей и других химикатов, так как они разрушают изоляцию, что может привести к поражению электрическим током;

Иметь переносной инструмент или лампу на 36 или 12 В.

Кроме того, запрещается оставлять без присмотра и под напряжением или поручать его незнакомому лицу, а также снимать установленные переносные плакаты “Не включать – работают люди” и др.

Для предупреждения возможности накопления зарядов статического электричества на человеке необходимо применять электростатическое заземление – специальные браслеты. Персонал должен быть обеспечен спецодеждой из мало электризующихся материалов. Безопасность использования элект-

роустановок обеспечивается путем применения: защитных ограждений с соблюдением необходимых расстояний до токоведущих частей; блокировки ограждений с аппаратами для предотвращения ошибочных операций; заземления корпусов электрооборудования и элементов установок, могущих оказаться под напряжением. Безопасность создается также надежным и быстродействующим защитным отключением при однофазном замыкании на землю частей электрооборудования или поврежденного участка сети, когда устройство заземления вызывает большие трудности или не обеспечивает безопасность обслуживания электроустановки. Кроме того, в целях безопасности применяют: понижающие трансформаторы с малым вторичным напряжением (42 В и ниже) для питания переносных электроинструментов, переносных ламп; разделяющие трансформаторы для включения в сеть электроприемников напряжением до 380 В с целью надежного изолирования; предупредительные надписи, плакаты и защитные средства.

Правила безопасности требуют, чтобы неизолированные токоведущие части были ограждены или закрыты кожухами, крышками и т. д. Открытые неизолированные провода и шины разрешается располагать на высоте не менее 3,5 м от уровня пола, при меньших высотах их следует ограждать для защиты от соприкосновения с токоведущими частями.

Защитные ограждения должны быть механически прочными; снимать и отрывать их допускается лишь с помощью ключей или инструментов.

Меры противопожарной безопасности являются неотъемлемой частью комплекса мероприятий по охране труда.

Необходимо соблюдать требования по содержанию помещений. Запрещается загромождать эвакуационные пути и выходы, производить уборку помещений и стирку спецодежды с применением бензина, керосина и прочих горючих жидкостей, оставлять неубранным промасленный обтирочный материал, оставлять после работы включенные в сеть электронагревательные приборы.

Короткое замыкание, перегрузка – характерные проявления аварийных режимов. Короткое замыкание возникает в местах нарушения изоляции токоведущих частей, присоединения проводов к токоприемникам и т. д.

Предупредить короткое замыкание можно правильным подбором электропроводов, способов их прокладки, применением надежной предохранительной защиты. Для защиты электросетей от перегрузки в сетях напряжением до 1000 В применяются плавкие предохранители и автоматические выключатели.

Неисправности в электросетях и электроаппаратуре, которые могут вызвать искрение, короткое замыкание, перегрев изоляции, должны немедленно устраняться. Неисправные электросети следует отключать до приведения ее в пожароопасное состояние. Тушить электропроводку можно только углекислотными и бромэтиловыми огнетушителями.

Прекращение пожара осуществляется следующими способами: охлаждением – охлаждение конденсированной фазы сплошными струями воды, охлаждение распыленными струями воды, охлаждение путем перемешивания горючих материалов; разбавлением – разбавление газовой и конденсированной фаз (твердой, жидкой) струями тонкораспыленной воды, разбавление горючих жидкостей водой, разбавлением негорючими газами или водяным паром; изоляцией слоем пены различной кратности, изоляция слоем огнетушащего порошка; химическим торможением реакции горения с помощью огнетушащих порошков или галоидопроизводных углеводородов. Вода – наиболее распространенное и достаточно эффективное огнетушащее средство.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы поставленные цели достигнуты. Для этого были решены следующие задачи:

- Разработаны принципиальные схемы устройств.
- Разработано две печатные платы
- Сформирован пакет выходной документации для производства (сборочные чертежи, перечни элементов, файлы GERBER).
- Написан и протестирован код для обеих МПС
- Рассчитана экономическая эффективность проекта

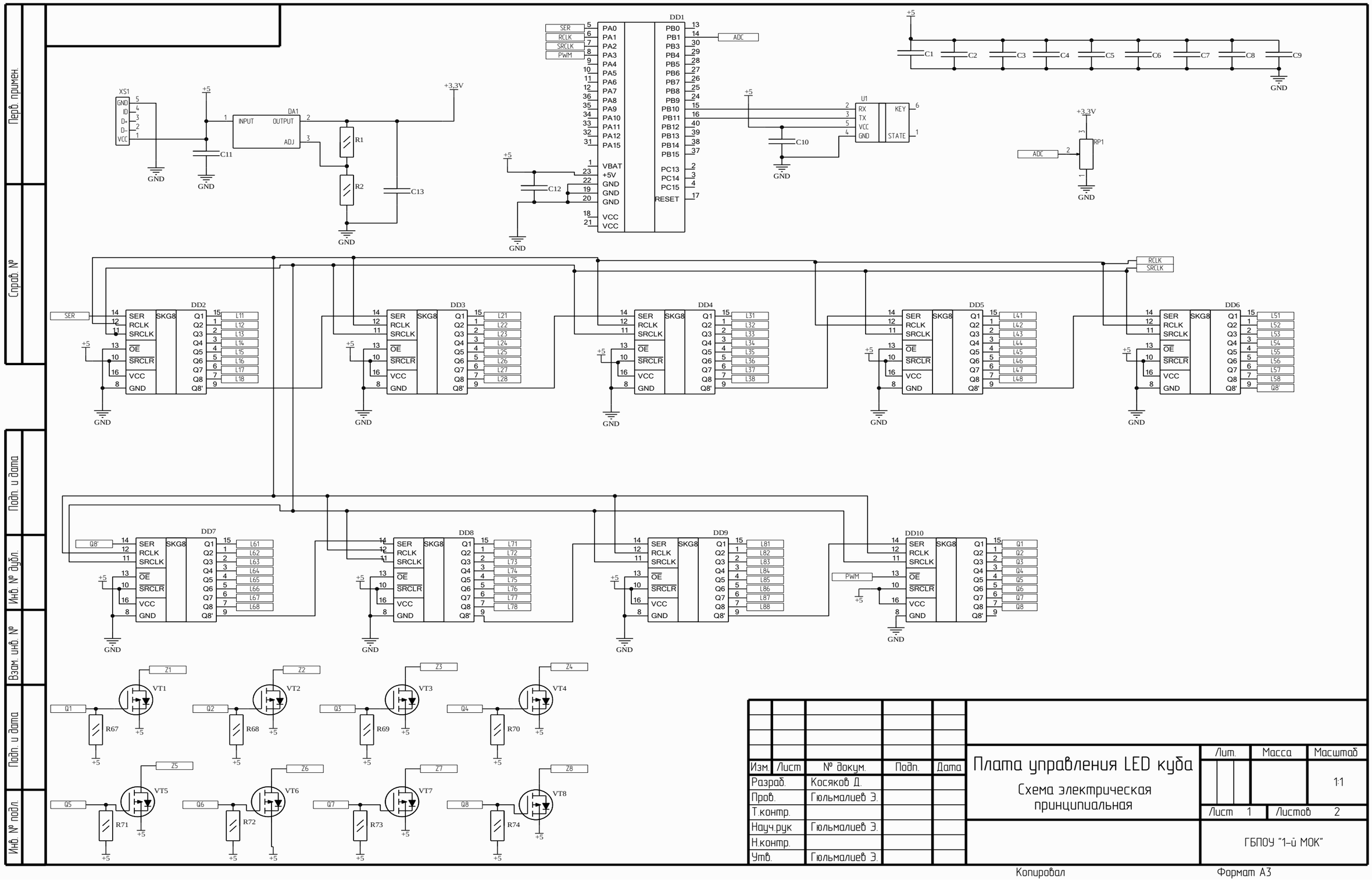
Данный проект можно считать завершенным, однако есть некоторые задачи, направленные на улучшение качества продукта:

- Поиск и исправление незначительных программных неисправностей
- Написание более подробной документации для добавления новых режимов другими программистами
- Улучшение принципа передачи данных для уменьшения дублируемого кода
- Добавление дополнительного функционала (например изменение строки для режима «Текст» с пульта ДУ, а не из кода программы)

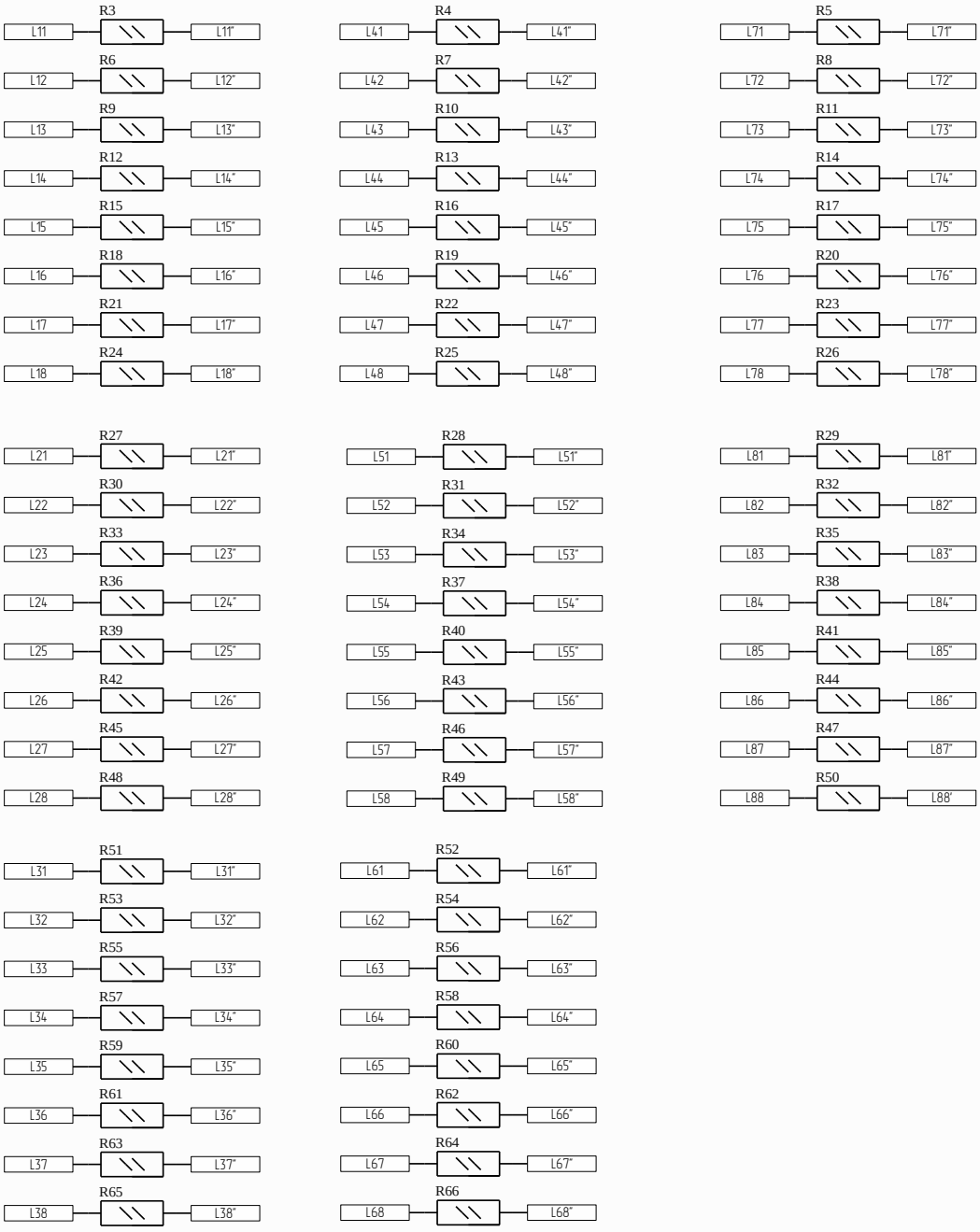
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ 2.701 (Схемы. Виды и типы. Общие требования к выполнению)
2. ГОСТ 2.702 (Правила выполнения электрических схем.)
3. ГОСТ 2.705 (Правила выполнения электрических схем обмоток и изделий с обмотками)
4. ГОСТ 2.708 (Правила выполнения электрических схем цифровой вычислительной техники)
5. ГОСТ 2.709 (Система обозначения цепей в электрических схемах)
6. ГОСТ 2.710 (Обозначения буквенно-цифровые в электрических схемах)
7. ГОСТ 2.728-74 (УГО резисторов и конденсаторов)
8. ГОСТ 2.730-73 (УГО полупроводниковых приборов)
9. ГОСТ 2.743.91 (УГО элементов цифровой техники)
10. История Arduino – сайт Arduino-tech
11. Описание семейства cortex – ST Microelectronics
12. Контроллеры с высокой вычислительной мощностью – ST Microelectronics
13. Линейка беспроводных контроллеров STM32 – ST Microelectronics
14. Линейка контроллеров STM32 с низкий энергопотреблением – ST Microelectronics
15. Основа для библиотеки дисплея – сайт GitHub
16. Расчет напряжения ШИМ – сайт electrosam
17. Reference manual для STM32F103xx – ST Microelectronics
18. Таймеры STM32 - сайт Istarik
19. Техническая документация на основные компоненты устройств – сайт Чип и Дип

ПРИЛОЖЕНИЕ А



ПРИЛОЖЕНИЕ Б



Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Косяков Д.		
Пров.		Гюльмалиев Э.		
Т.контр.				
Науч.рук.		Гюльмалиев Э.		
Н.контр.				
Утв.		Гюльмалиев Э.		

Плата управления LED куба

Схема электрическая принципиальная

Лит.	Масса	Масштаб
		1:1
Лист 2	Листов 2	

ГБПОУ «1-й МОК»

Копировал

Формат А4

ПРИЛОЖЕНИЕ В

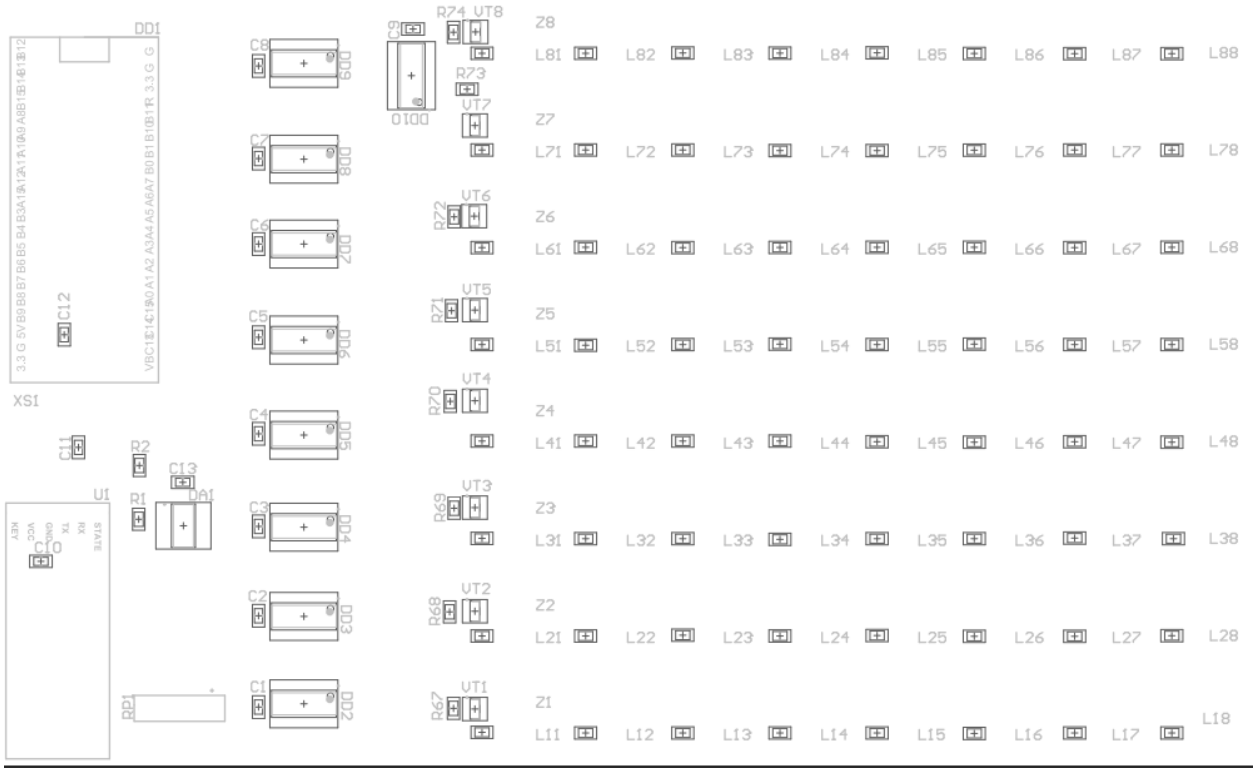
Позиционное обозначение	Наименование	Количество	Примечание
	Конденсаторы		импорт
C1...C11, C13	0.1 µF X7R 50B 10% 0805, GRM21BR71H104K	12	
C12	1 µF X7R 50B 10% 0805, GRM21BR71H104K	1	
	Микросхемы		импорт
DA1	LM317	1	
DD1	STM32F103C8T6	1	
DD2...DD10	74HC595	9	
U1	HC05	1	
	Резисторы		импорт
R1	0.125Вт 0805 240 Ом, 1% RC0805FR-07240RL	1	
R2	0.125Вт 0805 390 Ом, 1% RC0805FR-07240RL	1	
R3...R66	0.125Вт 0805 220 Ом, 1% RC0805FR-07240RL	64	
R67...R74	0.125Вт 0805 10 кОм, 1% RC0805FR-07240RL	8	
RP1	50 кОм	1	
	Транзисторы		
VT1...VT8	IRLML6401TRPBF 4.3 A sot23	8	импорт

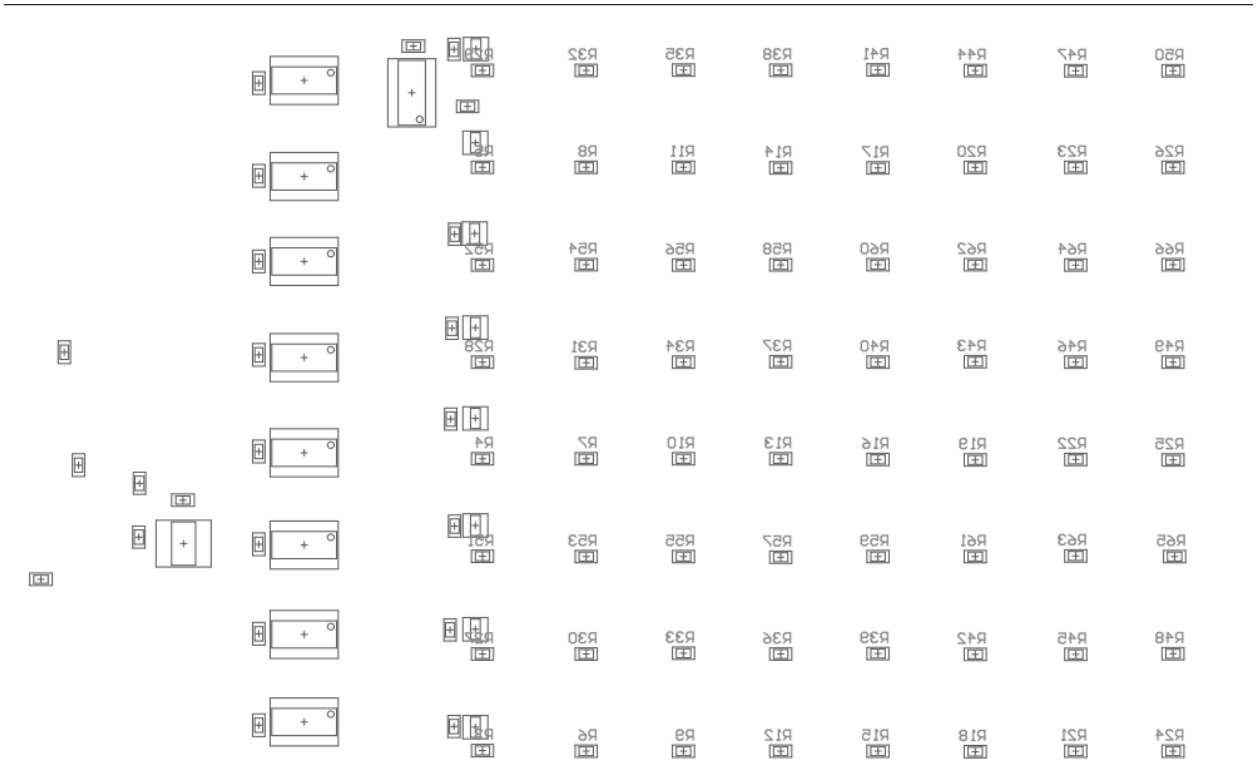
Изм.	Лист	№ докум.	Подпись	Дата	
Разраб.		Косяков Д.А			Перечень элементов платы управления LED кубом
Провер.					
Реценз.		Гюльмалиев Э.А.			
Н. Контр.					
Утверд.		Гюльмалиев Э.А.			

Лит.	Лист	Листов
	1	1

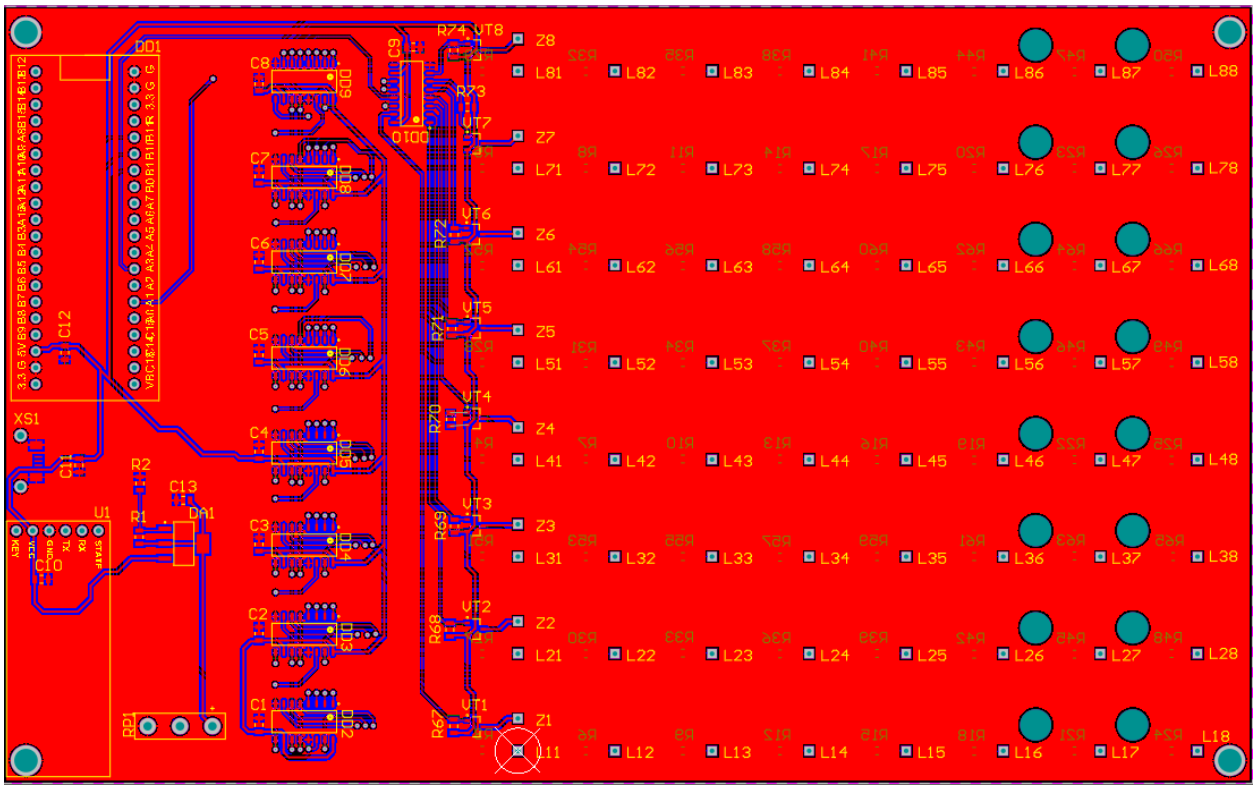
ГБПОУ «1-й МОК»

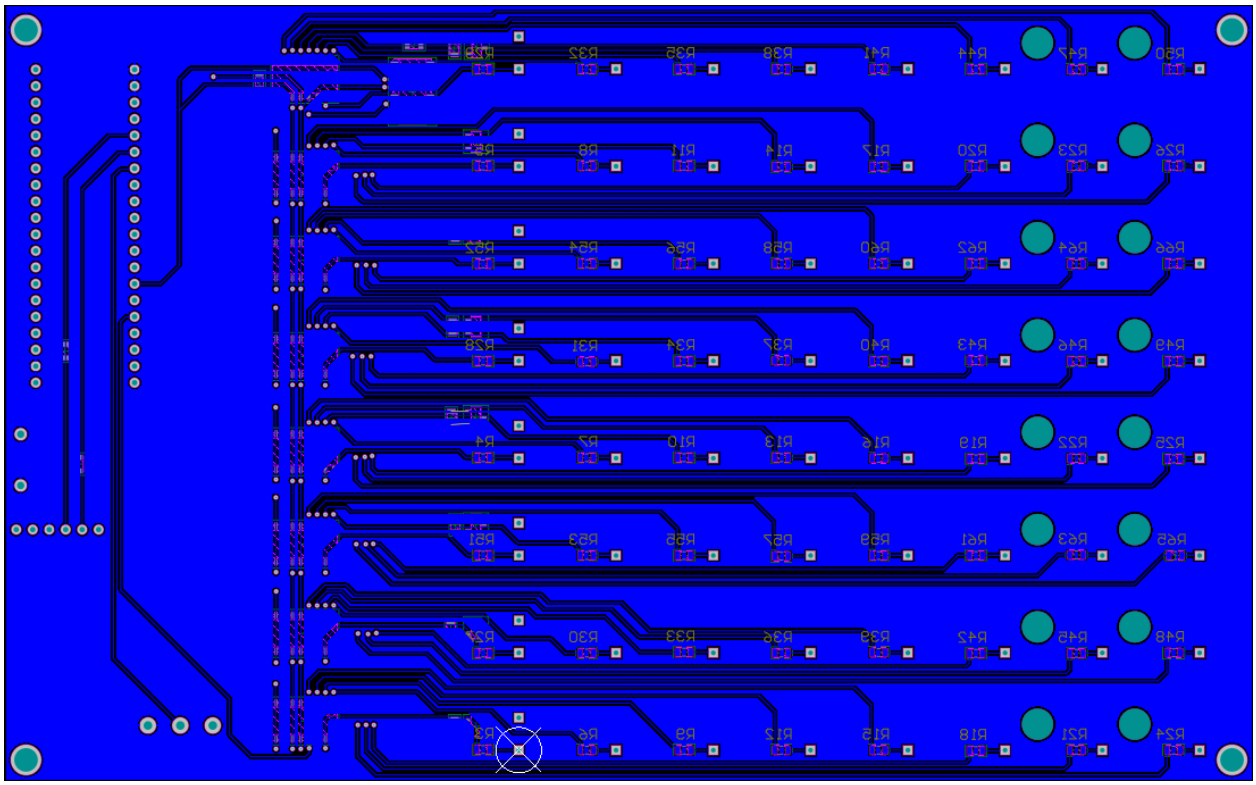
ПРИЛОЖЕНИЕ Г





ПРИЛОЖЕНИЕ Д





ПРИЛОЖЕНИЕ Е

Файл main.cpp

```
#include "main.h"

/* Private includes ----------------------------------------
------------------*/
/* USER CODE BEGIN Includes */
#include <cstdio>
#include <cstdlib>
#include "string.h"
#include "Menu.h"
/* USER CODE END Includes */
/* Private typedef -----------------------------------------
------------------*/
/* USER CODE BEGIN PTD */
typedef void (*menumode_t)();         //Тип указателя на
функцию режима для меню
/* USER CODE END PTD */
/* Private variables ---------------------------------------
------------------*/
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
UART_HandleTypeDef huart3;
/* USER CODE BEGIN PV */
Cube cube;
char str[100] = "EZ text";            //Строка для отладки
FlagUnion IsButtonPressed;
//[bit 16..9] | [bit 8] | [bit7] | [bit6] | [bit5] | [bit4]
| [bit3] | [bit2] | [bit1] | [bit0]
//Reset State    OK     RR       RU       RL       RB
LR       LU       LL       LB
//                EXTI8    EXTI7    EXTI6    EXTI5    EXTI4
EXTI3    EXTI2    EXTI1   EXTI0
uint8_t ModeSwitchingFlag = 0;
menumode_t ModePTR = ChangingSpeed;
/* USER CODE END PV */
/* Private function prototypes -----------------------------
------------------*/
void SystemClock_Config(void);
```

```c
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM2_Init(void);
static void MX_ADC2_Init(void);
static void MX_USART3_UART_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------
-----------------*/
/* USER CODE BEGIN 0 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)  //по
окончании измерения вызывается функция обратного вызова
{
    if(hadc->Instance == ADC2)                 //Если преры-
вание пришло от ADC2
    {
                                  //преобразование не нужно,
поскольку максимальные значения импульса и ацп равны
        TIM2 -> CCR4 = HAL_ADC_GetValue(&hadc2);    //Обраща-
емся к регистру CCRx, где x - номер канала таймера, для
управления скавжностью
    }
     if(hadc->Instance == ADC1)
    {
     srand(HAL_ADC_GetValue(&hadc1));      //по завершении
считывания значения с температурного датчика (ADC1), уста-
навливаем seed ряда случайных чисел
     HAL_ADC_DeInit(&hadc1);
                                                 //Отстрели-
ваем ацп (из колбека этого же ацп, да, именно так)
                                  //тк нам нужно на ацп1 про-
извести всего 1 измерение
    }
}//Останавливать АЦП не нужно, поскольку не включен режим
циклического измерения


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
         //Приведение полученных данных к более удобному
формату
```

```
        //Обнулять ничего не нужно поскольку если сделать
это тут,
        //то есть вероятность того, что флаг опустится до
его обработки
    if(huart == &huart3)
    {
        uint8_t TxVar = cube.getMode();
        HAL_UART_Transmit(&huart3, &TxVar, 1, 1000);
        HAL_UART_Receive_IT(&huart3, IsButtonPressed.IsBut-
tonPressedRxBuff, 2);
    }

}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

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

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

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

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

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */
```

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_TIM3_Init();
  MX_TIM2_Init();
  MX_ADC2_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */

//калибровка и запуск прерывания ацп
  HAL_TIM_Base_Start(&htim3);
  HAL_TIM_PWM_Start_IT(&htim2, TIM_CHANNEL_4);
  HAL_ADCEx_Calibration_Start(&hadc2);
  HAL_ADC_Start_IT(&hadc2);


uint16_t x = sizeof(cube);


  HAL_ADC_Start_IT(&hadc1);
  TIM3->EGR = TIM_EGR_UG;
  HAL_UART_Receive_IT(&huart3, IsButtonPressed.IsButton-
PressedRxBuff, 2);

  cube.setTextString(str, strlen(str));
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      if ((IsButtonPressed.IsButtonPressedRx >> OK) & 1)
      {
       IsButtonPressed.IsButtonPressedRx &= ~(1 << OK);
       ModeSwitchingFlag ^= 1;
       ModePTR = ModeSwitchingFlag ? SwitchingModes : Chang-
ingSpeed;
       cube.changeMode(); //TODO: в следующих версиях перера-
ботать переключение режимов
      }
      ModePTR();
```

```
      (cube.*cube.m_currentModePtr)(); //вызов метода через
указатель
      cube.printFrame();


    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
```

Файл Cube.h

```
#ifndef INC_CUBE_H_
#define INC_CUBE_H_


#define TOTAL_MODES 8
#include "stm32f1xx_hal.h"

#include "CubeFonts.h"
//#include "main.h"
#include "getRandomNumber.h"

enum Axis
{
     XAXIS = 0,
     YAXIS,
     ZAXIS
};

enum Direction
{
     POS_X = 0,
     NEG_X,
     POS_Y,
     NEG_Y,
     POS_Z,
     NEG_Z
};
enum Bright_Vector
{
     L2H = 0,           //Low to high
     H2L
};
```

```
enum Modes                    //TODO: обновить заголовочник в
файле для куба
{
     RAIN = 0,
     WALL,
     WOOP_WOOP,
     CUBE_JUMP,
     TEXT,
     LIGHTCUBE,
     WALKINGCUBE,
     CONST_CHANGE_BRIGHTNESS
};

enum Timers
{
     RAIN_TIME =         260,
     WALL_TIME =         220,
     WOOP_WOOP_TIME =    350,
     CUBE_JUMP_TIME =    200,
     TEXT_TIME =         300,
     CLOCK_TIME =        500, //У режима Lit нет времени, но
поле необходимо для удобной записи
     WALKING_TIME =          100,
     BRIGHT_TIME =       1
};

struct String                 //Структура для режима текста
{
     char *stringPtr;
     uint8_t lengh;
     uint8_t charCounter;         //номер текущего символа
     uint8_t charPosition;        //координата символа в кубе
по Y(нужно для формирования символа)
};

class Cube
{
     public:

     void printFrame();
                          //метод для отрисовки куба
```

```
    void setDot(uint8_t x, uint8_t y, uint8_t z, uint8_t
set_num);     //метод для установки определенного светодиода


    void changeMode();
    void incCurrentMode();
    void decCurrentMode();



    void text();                                        //метод
для бегущих букв
    void clearCube();                                   //поту-
шить все светодиоды
    void lightCube();                                   //зажечь
все светодиоды
    void walkingCube();
    //перемещающийся по осям куб
    void rain();                                        //Дождь
    void walk_and_flying_Wall();
    //Стена, перемещающаяся по случайным осям
    void woopWoop();                                    //Расши-
ряющийся и сужающийся куб
    void cubeJump();
    void constChangeBrightnes();

    void setTextString(char *text, uint8_t lengh);
    void setSpeed(int16_t timer);
    int32_t getSpeed();         //возможно временное решение
для изменения инкремента "на лету"

    typedef void (Cube::*ModePtr)();          //тип указа-
теля, который указывает на метод текущего режима, который
должен вызываться в обработчике таймера
    ModePtr m_currentModePtr = &Cube::lightCube;        //по
умолчанию указывает на режим светящегося куба

    Modes getMode();                        //геттер для ре-
жима
    private:

    void setWall(Axis axis, uint8_t i);
```

```
    void drawCube(uint8_t x, uint8_t y, uint8_t z, uint8_t
s);
    void shift(Direction dir);

    uint8_t m_OutFrame[8][8] =
         {
              {0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111111},
              {0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111111},
              {0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111111},
              {0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111111},
              {0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111111},
              {0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111111},
              {0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111111},
              {0b11111111, 0b11111111, 0b11111111,
0b11111111, 0b11111111, 0b11111111, 0b11111111, 0b11111111},
         };       //[z][x], y - номер бита (отсчет слева
направо)
    Modes m_currentMode = LIGHTCUBE;
    uint8_t getFont(uint8_t font, uint8_t layer);
    uint8_t m_IsModeSwitched = 1;        //Переменная-ста-
тус, показывающая факт того, что режим был переключен

    /*----------------------------Ходячий куб--------------
---------------*/
    int16_t m_coord[3];         //координаты
    int8_t m_vector[3];         //вектора
    /*----------------------------Ходячий куб--------------
---------------*/

    /*----------------------------Стена--------------------
------------*/
    Direction wallDirection;
    uint8_t  wallPosition = 0;
    uint8_t isLooped = 0;
    /*----------------------------Стена--------------------
------------*/
```

```
    /*---------------------------woopWoop-------------------
------------*/
    uint8_t cubeSize;
    uint8_t cubeExpanding = 0;
    /*---------------------------woopWoop-------------------
------------*/

    /*---------------------------Прыгающй куб---------------
------------*/
    uint8_t xPos;
    uint8_t yPos;
    uint8_t zPos;
    /*---------------------------Прыгающй куб---------------
------------*/

    String Text;                //Структура с текстом

    /*-----------------------таймеры-------------------------
------------*/
    uint32_t m_lastTrigger; //Время последнего "кадра" ре-
жима
    int32_t m_modeTimer = TEXT_TIME; //период кадров в ре-
жиме
    /*-----------------------таймеры-------------------------
------------*/

    /*-----------------------Меняющаяся яркость--------------
------------*/

    uint16_t brightness = 0;
    Bright_Vector vector = L2H;

    //uint8_t m_killADC = 0;     //Возможно не нужно, потом
надо убрать
    /*-----------------------Меняющаяся яркость--------------
------------*/
};



    #endif /* INC_CUBE_H_ */
```

Файл Cube.cpp

```
#include "Cube.h"
//#include "getRandomNumber.h"
#define RESETSERCLK GPIOA -> BSRR |= 1 << (2 + 16)
#define SETSERCLK GPIOA -> BSRR |= 1 << 2

#define RESETRCLK GPIOA -> BSRR |= 1 << (16 + 1)
#define SETRCLK GPIOA -> BSRR |= 1 << 1

void Cube::printFrame()
{
    for (uint8_t z = 0; z < 8; ++z)
    {
        uint8_t currentLayer = ~(128 >> (7 - z));

        RESETRCLK;

        for (uint8_t i = 0; i < 8; ++i)
            //Посылаем байт с текущим слоем в сдвиговый
регистр
        {
            RESETSERCLK;
            GPIOA -> BSRR |= 1 << (16 * !((currentLayer >>
i) & 1));
            SETSERCLK;
        }

        for (uint8_t x = 0; x < 8; ++x)
        {
            for (uint8_t y = 0; y < 8; ++y)
            {
                RESETSERCLK;
                GPIOA -> BSRR |= 1 << (16 * ((m_OutFrame[7
- z][x] >> y) & 1 ));
                SETSERCLK;
            }
        }


        SETRCLK;
        //HAL_Delay(1);
    }
}
```

```
void Cube::setSpeed(int16_t timer)
{
    m_modeTimer = timer;
}

int32_t Cube::getSpeed()
{
    return m_modeTimer;
}

Modes Cube::getMode()
{
    return m_currentMode;
}

void Cube::setDot(uint8_t x, uint8_t y, uint8_t z, uint8_t
set_num)
{
    m_OutFrame[7 - z][x] &= ~(128 >> y);
              //сбрасываем бит
    m_OutFrame[7 - z][x] |= (128 >> y) * set_num;
                   //устанавливаем 0 или 1 в зависимости от
set_num
}



void Cube::changeMode()                    //метод для пере-
ключения режима в зависимости от m_currentMode
{
     clearCube();
     m_IsModeSwitched = 1;
     m_lastTrigger = HAL_GetTick();

     ADC2->CR1 |= 1 << 5; //включаем ацп
                          //всегда устанавливаем бит в 1

                  //А обнуляем только если текущий режим

                  //CONST_CHANGE_BRIGHTNESS

     switch(m_currentMode)
     {
```

```
        case CONST_CHANGE_BRIGHTNESS:
            m_modeTimer = BRIGHT_TIME;
            ADC2->CR1 &= ~(1 << 5); //выключаем ацп, если
измерения не нужны                          //обнуление
бита EOCIE(End Of Conversion Int Enable)

                //для того чтобы ацп не перебивал режим
            m_currentModePtr = &Cube::constChangeBright-
nes;
            break;

        case RAIN:
            m_modeTimer = RAIN_TIME;
            m_currentModePtr = &Cube::rain;
            break;

        case WALL:
            m_modeTimer = WALL_TIME;
            m_currentModePtr = &Cube::walk_and_fly-
ing_Wall;
            break;

        case WOOP_WOOP:
            m_modeTimer = WOOP_WOOP_TIME;
            m_currentModePtr = &Cube::woopWoop;
            break;

        case CUBE_JUMP:
            m_modeTimer = CUBE_JUMP_TIME;
            m_currentModePtr = &Cube::cubeJump;
            break;

        case TEXT:
            m_modeTimer = TEXT_TIME;
            m_currentModePtr = &Cube::text;
            break;

        case LIGHTCUBE:
            m_currentModePtr = &Cube::lightCube;
            break;

        case WALKINGCUBE:
            m_modeTimer = WALKING_TIME;
```

```
                m_currentModePtr = &Cube::walkingCube;
                break;
     }
}

void Cube::incCurrentMode()
{
     m_currentMode =
static_cast<Modes>(static_cast<uint8_t>(m_currentMode) + 1);
    if (m_currentMode > TOTAL_MODES)
    {
     m_currentMode = RAIN;
    }
}

void Cube::decCurrentMode()
{
     m_currentMode =
static_cast<Modes>(static_cast<uint8_t>(m_currentMode) - 1);
    if (m_currentMode > TOTAL_MODES)
    {
     m_currentMode = static_cast<Modes>(TOTAL_MODES - 1);
          //защита от OutOfBounds
    }
}

void Cube::clearCube()
     //очистка куба
{
    for(uint8_t x = 0; x < 8; ++x)
    {
     for(uint8_t z = 0; z < 8; ++z)
     {
          m_OutFrame[z][x] = 0;
     }
    }
}

void Cube::lightCube()
     //Горящий куб
{
    if(m_IsModeSwitched)
    {
```

```
    for(uint8_t x = 0; x < 8; ++x)
    {
        for(uint8_t z = 0; z < 8; ++z)
        {
            m_OutFrame[z][x] = 0b11111111;
        }
    }
    m_IsModeSwitched = 0;
   }
}

void Cube::constChangeBrightnes()
{
    /*----------------------Setup----------------------------
*/
    if (m_IsModeSwitched)
    {
        lightCube();
        brightness = 0;
        vector = L2H;
    }
    /*----------------------Setup----------------------------
*/
    /*-----------------------Main loop-----------------------
*/
    if (HAL_GetTick() - m_lastTrigger > m_modeTimer)
    {
        m_lastTrigger = HAL_GetTick();
        switch(vector)
        {
            case L2H:
            if(++brightness == 4095)
            {
                vector = H2L;
            }
            case H2L:
            if(--brightness == 0)
            {
                vector = L2H;
            }
        }
        TIM2->CCR4 = brightness;
    }
```

```
    /*-----------------------Main loop-----------------------
*/
}
void Cube::text()
{
    /*---------------------Setup---------------------------
*/
    if (m_IsModeSwitched)
    {
        if(Text.stringPtr == nullptr)
        {
            return;                 //если строка null, вы-
ходим
        }
        m_IsModeSwitched = 0;
        clearCube();
        Text.charPosition = -1;
        Text.charCounter = 0;
    }
    /*---------------------Setup---------------------------
*/
    /*-----------------------Main loop-----------------------
*/
    if (HAL_GetTick() - m_lastTrigger > m_modeTimer)
    {
        m_lastTrigger = HAL_GetTick();

        shift(POS_X);
        if(++Text.charPosition == 7)
        {
            if(++Text.charCounter > Text.lengh - 1)
            {
                Text.charCounter = 0;
            }
            Text.charPosition = 0;
        }
        if(Text.charPosition == 0)
        {
            for(uint8_t i = 0; i < 8; ++i)
            {
                m_OutFrame[i][0] =
getFont(Text.stringPtr[Text.charCounter], i);
            }
```

```
          }
     }
     /*-----------------------Main loop----------------------
*/
}
void Cube::setTextString(char *text, uint8_t lengh)
{
     Text.stringPtr = text;
     Text.lengh = lengh;
}

void Cube::walkingCube()
     //Летающий куб внутри пространства
{
     /*---------------------Setup----------------------------
*/
     if (m_IsModeSwitched)
     {
          clearCube();
          m_IsModeSwitched = 0;
          m_modeTimer = WALKING_TIME;
          for (uint8_t i = 0; i < 3; ++i)
          {
               m_coord[i] = 300;
     //300, чтобы не использовать долгие вычисления с плаваю-
щей точкой
               m_vector[i] = getRandomNumber(3, 7) * 15; //B
f103 нет блока FPU
          }
          m_lastTrigger = HAL_GetTick();
     }
     /*----------------------Setup----------------------------
*/
     /*-----------------------Main loop----------------------
*/
     if (HAL_GetTick() - m_lastTrigger > m_modeTimer)
     {
          m_lastTrigger = HAL_GetTick();

          for(uint8_t i = 0; i < 3; ++i)
          {
               m_coord[i] += m_vector[i];
```

```
            if (m_coord[i] <= 1)                    //Защита
от отрицательных координат и векторов
            {
                m_coord[i] = 1;
                m_vector[i] = -m_vector[i];
                m_vector[i] += getRandomNumber(0, 5) - 3;
            }

            if (m_coord[i] >= (700 - 100))
            {
                m_coord[i] = (700 - 100);
                m_vector[i] = -m_vector[i];
                m_vector[i] += getRandomNumber(0, 5) - 3;
            }
        }
        clearCube();
        int8_t thisX = m_coord[0] / 100;
        int8_t thisY = m_coord[1] / 100;
        int8_t thisZ = m_coord[2] / 100;

        setDot(thisX,      thisY,        thisZ,        1);
        setDot(thisX + 1,  thisY,        thisZ,        1);
        setDot(thisX,      thisY + 1,    thisZ,        1);
        setDot(thisX,      thisY,        thisZ + 1,    1);
        setDot(thisX + 1,  thisY + 1,    thisZ,        1);
        setDot(thisX,      thisY + 1,    thisZ + 1,    1);
        setDot(thisX + 1,  thisY,        thisZ + 1,    1);
        setDot(thisX + 1,  thisY + 1,    thisZ + 1,    1);
    }
    /*----------------------Main loop----------------------
*/
}

void Cube::rain()
        //Режим дождя
{
    /*--------------------Setup---------------------------
*/
    if (m_IsModeSwitched)
    {
        clearCube();
        m_IsModeSwitched = 0;
        m_lastTrigger = HAL_GetTick();
```

```
    }
    /*---------------------Setup---------------------------
*/

    /*-----------------------Main loop---------------------
*/
    if (HAL_GetTick() - m_lastTrigger > m_modeTimer)
    {
        m_lastTrigger = HAL_GetTick();
        shift(NEG_Y);
        uint8_t numDrops = getRandomNumber(0, 4);
        for (uint8_t i = 0; i < numDrops; ++i)
        {
            setDot(getRandomNumber(0, 7), getRandom-
Number(0, 7), 7, 1);
        }

    }
    /*-----------------------Main loop---------------------
*/
}


void Cube::walk_and_flying_Wall()
    //режим, в котором есть двигающаяся по случайной оси
"стена"
{

    /*---------------------Setup---------------------------
*/
    if (m_IsModeSwitched)
    {

        clearCube();
        uint8_t axis = getRandomNumber(0, 2);
        wallPosition = getRandomNumber(0, 1) * 7;
        setWall(static_cast<Axis>(axis), wallPosition);
        switch(static_cast<Axis>(axis))
        {
            case XAXIS:
                if(wallPosition == 0)
                {
                    wallDirection = POS_X;
```

```
                    }
                    else
                    {
                        wallDirection = NEG_X;
                    }
                    break;

                case YAXIS:
                    if(wallPosition == 0)
                    {
                        wallDirection = POS_Y;
                    }
                    else
                    {
                        wallDirection = NEG_Y;
                    }
                    break;

                case ZAXIS:
                    if(wallPosition == 0)
                    {
                        wallDirection = POS_Z;
                    }
                    else
                    {
                        wallDirection = NEG_Z;
                    }
                    break;
            }
            isLooped = 0;
            m_IsModeSwitched = 0;
            m_lastTrigger = HAL_GetTick();
        }
        /*----------------------Setup----------------------------
*/
        /*-----------------------Main loop-----------------------
*/
        if (HAL_GetTick() - m_lastTrigger > m_modeTimer)
        {
            m_lastTrigger = HAL_GetTick();
            shift(wallDirection);
            if (wallDirection % 2 == 0)
            {
```

```
            if (++wallPosition == 7)
            {
                if (isLooped)
                {
                    m_IsModeSwitched = 1;
    //когда достигло края, перезагружаем режим
                }
                else
                {
                    wallDirection = static_cast<Direc-
tion>(static_cast<uint8_t>(wallDirection) + 1);  //Это про-
сто работает
                    isLooped = 1;
                }
            }
        }
        else
        {
            if (--wallPosition == 0)
            {
                if (isLooped)
                {
                    m_IsModeSwitched = 1;
                }
                else
                {
                    wallDirection = static_cast<Direc-
tion>(static_cast<uint8_t>(wallDirection) - 1);
                    isLooped = 1;
                }
            }
        }
    }
    /*-----------------------Main loop----------------------
*/
}



void Cube::woopWoop()
{
    /*---------------------Setup---------------------------
*/
```

```
    if (m_IsModeSwitched)
    {
        m_modeTimer = WOOP_WOOP_TIME;
        clearCube();
        cubeSize = 2;
        cubeExpanding = 1;
        m_IsModeSwitched = 0;
    }
    /*---------------------Setup----------------------------
*/

    /*-----------------------Main loop----------------------
*/
    if (HAL_GetTick() - m_lastTrigger > m_modeTimer)
    {
        m_lastTrigger = HAL_GetTick();
        /*-----------переключение из сжимающегося в расши-
рающийся(или обратно)-----------*/
        if (cubeExpanding)
        {
            cubeSize += 2;
            if (cubeSize == 8)
            {
                cubeExpanding = 0;
            }
        }
        else
        {
            cubeSize -= 2;
            if (cubeSize == 2)
            {
                cubeExpanding = 1;
            }
        }
        /*-----------переключение из сжимающегося в расши-
рающийся(или обратно)-----------*/

        clearCube();
        drawCube(4 - cubeSize / 2, 4 - cubeSize / 2, 4 -
cubeSize / 2, cubeSize);
    }
    /*-----------------------Main loop----------------------
*/
```

```
}

void Cube::cubeJump()
{
    /*----------------------Setup----------------------------
*/
    if (m_IsModeSwitched)
    {
        m_IsModeSwitched = 0;
        clearCube();
        xPos = getRandomNumber(0, 1) * 7;
        yPos = getRandomNumber(0, 1) * 7;
        zPos = getRandomNumber(0, 1) * 7;
        cubeSize = 8;
        cubeExpanding = 0;
    }
    /*----------------------Setup----------------------------
*/

    /*-----------------------Main loop-----------------------
*/
    if (HAL_GetTick() - m_lastTrigger > m_modeTimer)
    {
        m_lastTrigger = HAL_GetTick();
        clearCube();
        /*--------------------------Определение координат
угла--------*/
        if (xPos == 0 && yPos == 0 && zPos == 0)
        {
            drawCube(xPos, yPos, zPos, cubeSize);
        }
        else if (xPos == 7 && yPos == 7 && zPos == 7)
        {
            drawCube(xPos + 1 - cubeSize, yPos + 1 - cu-
beSize, zPos + 1 - cubeSize, cubeSize);
        }
        else if (xPos == 7 && yPos == 0 && zPos == 0)
        {
            drawCube(xPos + 1 - cubeSize, yPos, zPos, cu-
beSize);
        }
        else if (xPos == 0 && yPos == 7 && zPos == 0)
        {
```

```
            drawCube(xPos, yPos + 1 - cubeSize, zPos, cu-
beSize);
        }
        else if (xPos == 0 && yPos == 0 && zPos == 7)
        {
            drawCube(xPos, yPos, zPos + 1 - cubeSize, cu-
beSize);
        }
        else if (xPos == 7 && yPos == 7 && zPos == 0)
        {
            drawCube(xPos + 1 - cubeSize, yPos + 1 - cu-
beSize, zPos, cubeSize);
        }
        else if (xPos == 0 && yPos == 7 && zPos == 7)
        {
            drawCube(xPos, yPos + 1 - cubeSize, zPos + 1 -
cubeSize, cubeSize);
        }
        else if (xPos == 7 && yPos == 0 && zPos == 7)
        {
            drawCube(xPos + 1 - cubeSize, yPos, zPos + 1 -
cubeSize, cubeSize);
        }
        /*---------------------------Определение координат
угла--------*/

        if(cubeExpanding)
        {
             if (++cubeSize == 8)
             {
                  cubeExpanding = 0;
                  xPos = getRandomNumber(0, 1) * 7;
                  yPos = getRandomNumber(0, 1) * 7;
                  zPos = getRandomNumber(0, 1) * 7;
             }
        }
        else
        {
             if (--cubeSize == 1)
             {
                  cubeExpanding = 1;
             }
        }
```

```
    }
    /*-----------------------Main loop----------------------
*/
}



void Cube::drawCube(uint8_t x, uint8_t y, uint8_t z, uint8_t
s)
{
    for (uint8_t i = 0; i < s; ++i)
    {
        setDot(x, y + i, z, 1);
        setDot(x + i, y, z, 1);
        setDot(x, y, z + i, 1);
        setDot(x + s - 1, y + i, z + s - 1, 1);
        setDot(x + i, y + s - 1, z + s - 1, 1);
        setDot(x + s - 1, y + s - 1, z + i, 1);
        setDot(x + s - 1, y + i, z, 1);
        setDot(x, y + i, z + s - 1, 1);
        setDot(x + i, y + s - 1, z, 1);
        setDot(x + i, y, z + s - 1, 1);
        setDot(x + s - 1, y, z + i, 1);
        setDot(x, y + s - 1, z + i, 1);
    }
}

void Cube::setWall(Axis axis, uint8_t i)            //в зави-
симости от оси и точки на этой оси, закрашиваем поверхность
{
    for(uint8_t j = 0; j < 8; ++j)
    {
        for(uint8_t k = 0; k < 8; ++k)
        {
            switch(axis)
            {
                case XAXIS:
                    setDot(i, j, k, 1);
                    break;
                case YAXIS:
                    setDot(j, i, k, 1);
                    break;
                case ZAXIS:
```

```
                        setDot(j, k, i, 1);
                }
            }
        }
}

void Cube::shift(Direction dir)
{
    switch(dir)
    {
        case POS_X:
                        //Каждый столбец начиная со восьмого
копирует то, что

                        //находится в левом столбце
            for (uint8_t x = 7; x > 0; --x)
            {
                for (uint8_t z = 0; z < 8; ++z)
                {
                    m_OutFrame[z][x] = m_OutFrame[z][x -
1];
                }
            }
            for (uint8_t i = 0; i < 8; ++i)
                //Затем все что в самом левом столбце за-
тирается
            {
                m_OutFrame[i][0] = 0;
            }
            break;
        case NEG_X:
            for (uint8_t x = 0; x < 7; ++x)
                //Каждый столбец начиная с нулевого копи-
рует то, что
                                    //находится в правом столбце
            {
                for (uint8_t z = 0; z < 8; ++z)
                {
                    m_OutFrame[z][x] = m_OutFrame[z][x +
1];
                }
            }
```

```
                for (uint8_t i = 0; i < 8; ++i)
                    //Затем все что в самом правом столбце за-
тирается
                {
                    m_OutFrame[i][7] = 0;
                }
                break;
            case POS_Y:
                for (uint8_t x = 0; x < 8; ++x)
                    //Каждая ячейка сдвигатся на 1 вправо
                {
                    for (uint8_t z = 0; z < 8; ++z)
                    {
                        m_OutFrame[z][x] >>= 1;
                    }
                }
                break;
            case NEG_Y:
                for (uint8_t x = 0; x < 8; ++x)
                    //Каждая ячейка сдвигатся на 1 влево
                {
                    for (uint8_t z = 0; z < 8; ++z)
                    {
                        m_OutFrame[z][x] <<= 1;
                    }
                }
                break;
            case POS_Z:
                for (uint8_t x = 0; x < 8; ++x)
                    //Каждая строка, начиная с нулевой копи-
рует то, что
                {
                        //находится в следующей
                    for (uint8_t z = 0; z < 7; ++z)
                    {
                        m_OutFrame[z][x] = m_OutFrame[z +
1][x];
                    }
                }
                for (uint8_t i = 0; i < 8; ++i)
                    //Затем все что в самой нижней строке за-
тирается
                {
```

```
                    m_OutFrame[7][i] = 0;
               }
               break;


          case NEG_Z:
                         //Каждая строка, начиная с седьмой
копирует то, что
               for (uint8_t x = 0; x < 8; ++x)
                    //находится в предыдущей
               {
                    for (uint8_t z = 7; z > 0; --z)
                    {
                         m_OutFrame[z][x] = m_OutFrame[z -
1][x];
                    }
               }
               for (uint8_t i = 0; i < 8; ++i)
                    //Затем все что в самом правом столбце за-
тирается
               {
                    m_OutFrame[0][i] = 0;
               }
               break;
     }
     }
```

Файл getRandomNumber.cpp

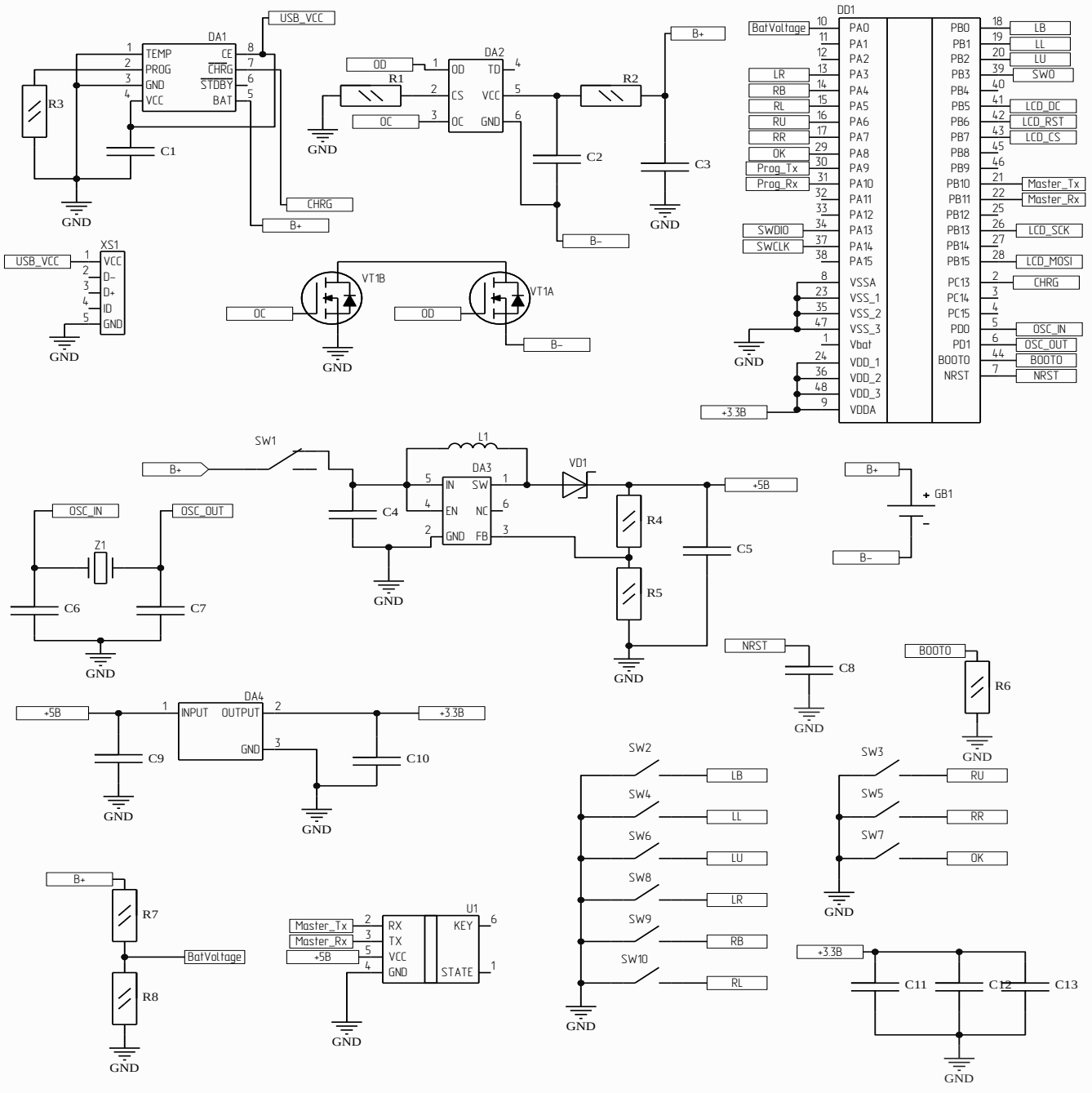
```
#include "main.h"
// Генерируем рандомное число между значениями min и max
// Предполагается, что функцию srand() уже вызывали
int getRandomNumber(int min, int max)
{
    // Равномерно распределяем рандомное число в диапазоне
    return (rand() % (max - min + 1)) + min;
}
```

Файл Menu.cpp

```
#include "Menu.h"

void SwitchingModes()
{
    if ((IsButtonPressed.IsButtonPressedRx >> LB) & 1)
```

```
    {
     cube.incCurrentMode();
     IsButtonPressed.IsButtonPressedRx &= ~(1 << LB);
    }
    if ((IsButtonPressed.IsButtonPressedRx >> LU) & 1)
    {
     cube.decCurrentMode();
     IsButtonPressed.IsButtonPressedRx &= ~(1 << LU);
    }
}
void ChangingSpeed()
{
    if ((IsButtonPressed.IsButtonPressedRx >> RB) & 1)
    {
     cube.setSpeed(cube.getSpeed() + 10);
     if (cube.getSpeed() > 1000)
     {
         cube.setSpeed(1000);
     }
     IsButtonPressed.IsButtonPressedRx &= ~(1 << RB);
    }
    if ((IsButtonPressed.IsButtonPressedRx >> RU) & 1)
    {
     cube.setSpeed(cube.getSpeed() - 10);
     if (cube.getSpeed() < 0)
     {
         cube.setSpeed(0);
     }
     IsButtonPressed.IsButtonPressedRx &= ~(1 << RU);
    }
    if ((IsButtonPressed.IsButtonPressedRx >> LL) & 1)
    {
     cube.changeMode();
     uint8_t TxVar = cube.getMode();
     //HAL_UART_Transmit(&huart3, &TxVar, 1, 0xFF);
     IsButtonPressed.IsButtonPressedRx &= ~(1 << LL);
    }
     }
```

ПРИЛОЖЕНИЕ Ж



Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Косяков Д.		
Пров.		Гюльмалиев Э.		
Т.контр.				
Науч. рук		Гюльмалиев Э.		
Н.контр.				
Утв.		Гюльмалиев Э.		

Плата пульта ДУ

Схема электрическая принципиальная

Лит.	Масса	Масштаб
		1:1
Лист 1	Листов 2	

ГБПОУ "1-й МОК"

Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

Взам. инв. №

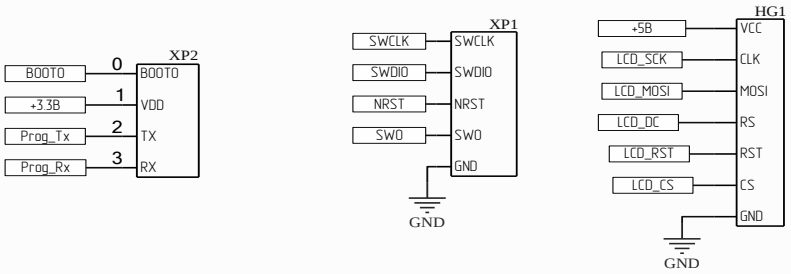
Подп. и дата

Инв. № подл.

Копировал

Формат А4

ПРИЛОЖЕНИЕ 3



Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Косяков Д.		
Пров.		Гюльмалиев Э.		
Т.контр.				
Науч.рук		Гюльмалиев Э.		
Н.контр.				
Утв.		Гюльмалиев Э.		

Плата

Схема электрическая принципиальная

Лит.	Масса	Масштаб

Лист 2 Листов 2

ГБПОУ "1-й МОК"

Копировал Формат А4

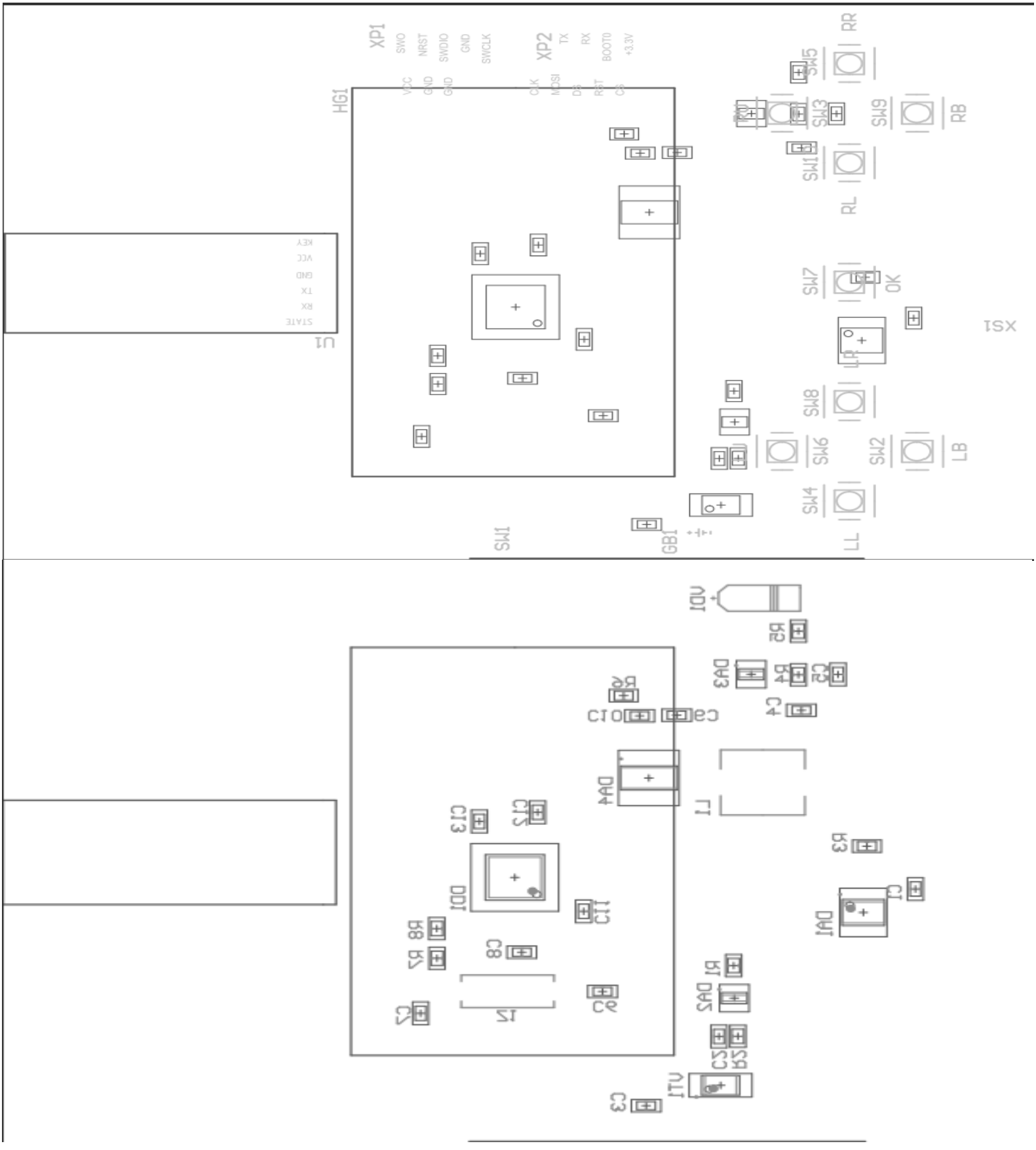
ПРИЛОЖЕНИЕ И

Позиционное обозначение	Наименование	Количество	Примечание
	Конденсаторы		импорт
C1, C2, C8, C11...C13	0.1 µF X7R 50B 10% 0805, GRM21BR71H104K	6	
C3, C9, C10	10 µF X7R 50B 10% 0805, GRM21BR71H104K	1	
C4, C5,	22 µF X7R 50B 10% 0805, GRM21BR71H104K	2	
C6, C7	22 pF X7R 50B 10% 0805, GRM21BR71H104K	2	
	Микросхемы		импорт
DA1	TP4056	1	
DA2	DW01A	1	
DA3	MT3608	1	
DA4	NCP117	1	
DD1	STM32F103C8T6	1	
U1	HC05	1	
	Катушки индуктивности		
L1	B82472G6223M000, 22 µF , 1.45 A, 7X7	1	
	Резисторы		импорт
R1	0.125Вт 0805 1 кОм, 1% RC0805FR-07240RL	1	
R2	0.125Вт 0805 100 Ом, 1% RC0805FR-07240RL	1	
R3	0.125Вт 0805 1.2 кОм, 1% RC0805FR-07240RL	1	
R4	0.125Вт 0805 15 кОм, 1% RC0805FR-07240RL	1	
R5	0.125Вт 0805 2 кОм, 1% RC0805FR-07240RL	1	
R6	0.125Вт 0805 100 кОм, 1% RC0805FR-07240RL	1	
R7, R8	0.125Вт 0805 47 кОм, 1% RC0805FR-07240RL	2	
	Переключатели		импорт
SW1	SMTS-102-2C3, Тумблер ON-ON (1.5A 250VAC) SPDT 3P	1	
SW2...SW10	Кнопка тактовая 6x6 мм	9	
	Диоды		импорт
VD1	SS34 Диод Шоттки	1	
	Транзисторы		
VT1	FS8205A Dual N-Channel Power MOSFET 6A	1	импорт

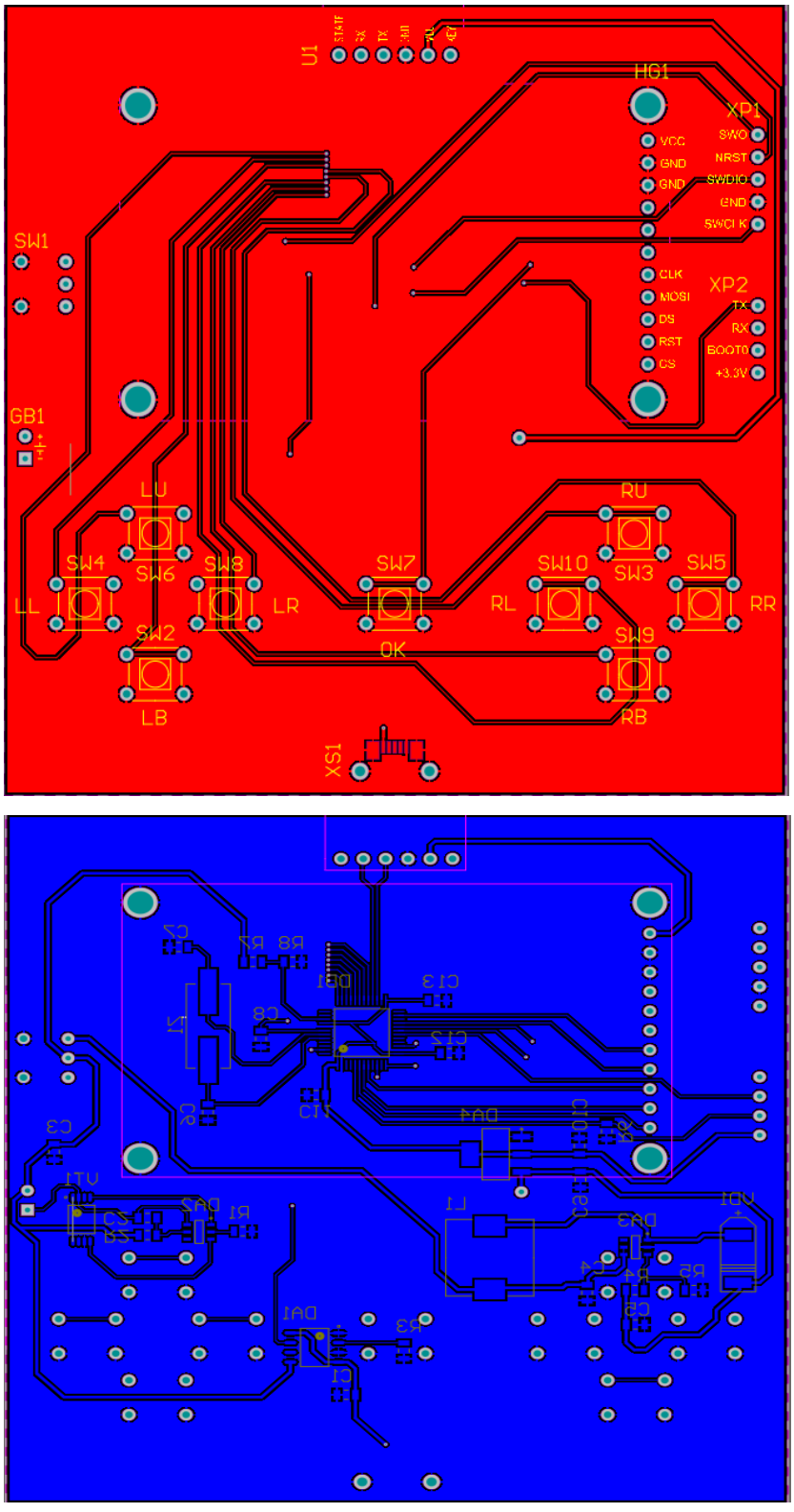
Изм.	Лист	№ докум.	Подпись	Дата	
Разраб.		Косяков Д.А			Перечень элементов платы управления LED кубом
Провер.		Гюльмалиев Э.А.			
Реценз.					
Н. Контр.					
Утверд.		Гюльмалиев Э.А.			

Лит.	Лист	Листов
	1	1

ПРИЛОЖЕНИЕ К



ПРИЛОЖЕНИЕ Л



ПРИЛОЖЕНИЕ М

Файл main.c

```
#include "main.h"

/* Private includes ----------------------------------------
------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
#include "lightning.h"
#include "BluetoothImage.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------
------------------*/
/* USER CODE BEGIN PTD */
typedef void (*menumode_t)();         //Тип указателя на
функцию режима для меню
/* USER CODE END PTD */

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

SPI_HandleTypeDef hspi2;
DMA_HandleTypeDef hdma_spi2_tx;

TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;

UART_HandleTypeDef huart3;
extern uint16_t IsButtonsPressed;                    //флаг,
который показывает, что нажатие кнопки нужно обработать
//[bit 16..9] | [bit 8] | [bit7] | [bit6] | [bit5] | [bit4]
| [bit3] | [bit2] | [bit1] | [bit0]
//Reset State    OK     RR       RU       RL       RB
LR       LU       LL       LB
//                EXTI8    EXTI7    EXTI6    EXTI5    EXTI4
EXTI3    EXTI2    EXTI1   EXTI0
```

```c
uint8_t RecievedMode = 0;                       //Получаемый
режим от куба

extern FontDef Font_7x10;
extern FontDef Font_11x18;
extern FontDef Font_16x26;


extern const uint16_t Timers_cube[TOTAL_MODES];
extern int16_t Speed;
extern enum Modes CurrentMode;

uint8_t ChargePercent = 0;            //Заряд в процентах
char ChargePercentOutString[50];
extern uint8_t IsModeSwithced;
uint8_t IsModeRxd = 0;    //Флаг для принятия ответа
uint8_t DrawBTbmp = 0;
uint8_t SynchroFlag = 0;  //Флаг для синхронизации
uint8_t ADC_Flag = 0;     //Если не ввести флаг на обработку
ацп, то не исключены зависания
uint8_t MenuMode = 0; //Если 0 - текущий режим и частота(в
Гц), 1 - "карусель" выбора режима
/* USER CODE END PV */

/* Private function prototypes ----------------------------
------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_SPI2_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);

/* Private user code --------------------------------------
------------------*/
/* USER CODE BEGIN 0 */

#define MIN_Percent_charge_mV 1650
#define Dif_Between_100_And_0_Percent_charge_mv 464
/*
 * Константы выше подобраны опытным путем
```

```
 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
     if(hadc->Instance == ADC1)
     {
         HAL_TIM_Base_Stop(&htim3);
         ADC_Flag = 1;       //В бесконечном цикле происходит
отрисовка заряда в процентах
     }
}

void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi)
{
     // снимаем Chip Select только после окончания передачи
DMA, не в коде программы
     // т.к. DMA работает независимо от CPU. Если это сде-
лать в коде программы, то не исключены
      // артефакты
     if (hspi->Instance == SPI2)
     {   // //проверяем какой SPI вызвал колбек

      HAL_GPIO_WritePin(CSX_PORT, CSX_PIN, GPIO_PIN_SET);
     }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart == &huart3)
    {
     IsModeRxd = 1;
     if(SynchroFlag)
     {//TODO: сместить режимы на единичку
         SynchroFlag = 0;
         CurrentMode = RecievedMode; //Перезапись режима
         Speed = Timers_cube[RecievedMode]; //Сброс тайме-
ров(со стороны основной МПС произошло тоже самое)
     }
     TIM4->EGR = TIM_EGR_UG;
     HAL_UART_Receive_IT(&huart3, &RecievedMode, 1);
    }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
```

```
{
    if(htim->Instance == TIM4)
    {
     HAL_TIM_Base_Stop_IT(&htim4);
     DrawBTbmp = IsModeRxd;
     IsModeRxd = 0;

    }
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
menumode_t CurrentMode = ShowingCurrentMode;

  /* USER CODE END 1 */

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

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

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

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

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
```

```
  MX_ADC1_Init();
  MX_SPI2_Init();
  MX_USART3_UART_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  /* USER CODE BEGIN 2 */
  __HAL_TIM_CLEAR_FLAG(&htim4, TIM_SR_UIF); //очищаем флаг
события таймера, чтобы прерывание не сработало раньше вре-
мени


//ILI9225_Unselect();
  lcd_init();
  fill_rectangle(0, 0, 220, 176, COLOR_RED);
  fill_rectangle(0, 164, 220, 176, COLOR_DARKRED);
  HAL_TIM_Base_Start(&htim3);

  HAL_ADCEx_Calibration_Start(&hadc1);
  HAL_ADC_Start_IT(&hadc1);
  TIM3->EGR = TIM_EGR_UG;                //Пренудительно
генерируем событие, чтобы сразу запустить ацп

  //draw_bitmap(183, 165, 7, 10, lightning_bmp);
  HAL_UART_Receive_IT(&huart3, &RecievedMode, 1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
       Post_Processing_Buttons();//отработка антидребезга

     if ((IsButtonsPressed >> OK) & 1)
     {
         MenuMode ^= 1;          //инвертируем флаг по нажа-
тию на кнопку OK
         IsButtonsPressed &= ~(1 << OK);

         IsModeSwithced = 1;
         fill_rectangle(0, 0, 220, 164, COLOR_RED);
```

```
        if (MenuMode)                //В зависимости от ре-
жима, заставляем указатель режима указывать на нужный нам
режим
        {
         CurrentMode = SwitchingModes;
        }
        else
        {
         CurrentMode = ShowingCurrentMode;
        }
//            EXTI->IMR |= 1 << OK;
    }

    CurrentMode();//Вызываем нужный режим по указателю


    if(DrawBTbmp)
    {
        draw_bitmap(175, 164, 7, 12, BL_Bmp); //Если ответ
есть, нарисовать иконку

    }
    else
    {
        fill_rectangle(175, 164, 189, 176, COLOR_DARKRED);
//иначе затереть
    }

    if (ADC_Flag)
    {

        uint16_t mVoltage = ((336800U / 4096) *
HAL_ADC_GetValue(&hadc1))
             / 100;
        uint8_t ChargePercent = ((mVoltage -
MIN_Percent_charge_mV) * 100)
             / Dif_Between_100_And_0_Percent_charge_mv;

        sprintf(ChargePercentOutString, "%d%c ", ChargePer-
cent, 37);
        ADC_Flag = 0;
        HAL_TIM_Base_Start(&htim3);
```

```
        draw_string(190, 164, ChargePercentOutString,
Font_7x10,
                COLOR_WHITE, COLOR_DARKRED);
    }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Файл stm32f1xx_it.c

```
#include "main.h"
#include "stm32f1xx_it.h"

#define Delay 120  //Задержка для антидребезга

void Post_Processing_Buttons();

/* USER CODE BEGIN 0 */
extern UART_HandleTypeDef huart3;
extern TIM_HandleTypeDef htim4;
extern uint8_t RecievedMode;
extern uint8_t SynchroFlag;
volatile uint16_t buttFlags = 0;          //Полуслово флагов
//volatile uint16_t buttStatuses = 0;          //Полуслово
состояний UPD: не нужно
 uint32_t IntTimeIRQ[9];              //Массив из моментов
последних нажатий (срабатываний прерываний)
volatile uint16_t IsButtonsPressed = 0;        //флаг, кото-
рый показывает, что нажатие кнопки нужно обработать

void Post_Processing_Buttons()
{
    IRQn_Type NVIC_ITS[9] =
        {
            EXTI0_IRQn, EXTI1_IRQn, EXTI2_IRQn, EXTI3_IRQn,
EXTI4_IRQn,
            EXTI9_5_IRQn, EXTI9_5_IRQn, EXTI9_5_IRQn,
EXTI9_5_IRQn
        };
    for(uint8_t ButtNo = 0; ButtNo < 9; ++ButtNo)
    {
        if (((buttFlags >> ButtNo) & 1) && HAL_GetTick() -
IntTimeIRQ[ButtNo] > Delay)
```

```
        {
                                    //Ничего не делаем, пока не
пройдет время задержки
            //Включаем прерывание
            __HAL_GPIO_EXTI_CLEAR_IT(1 << ButtNo);
            NVIC_ClearPendingIRQ(NVIC_ITS[ButtNo]);
            EXTI->IMR |= 1 << ButtNo;
            buttFlags &= ~(1 << ButtNo);
        }
    }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    EXTI->IMR &= ~(GPIO_Pin);
    //NVIC_ClearPendingIRQ();
    IsButtonsPressed |= GPIO_Pin;
    uint8_t TxBuff[2] = {(IsButtonsPressed & 0xff), (IsBut-
tonsPressed >> 8 & 0xFF)};//важно чтобы младшие биты шли
справа
    HAL_TIM_Base_Start_IT(&htim4);
    HAL_UART_Transmit_IT(&huart3, TxBuff, 2);
    //HAL_UART_Receive_IT(&huart3, &RecievedMode, 1);
    buttFlags |= GPIO_Pin;
    if((IsButtonsPressed >> LL) & 1) //вызов синхронизации
    {
        IsButtonsPressed &= ~(1 << LL);
        SynchroFlag = 1;
    }
}
```

Файл Menu.c

```
#include "Menu.h"
#include "ILI9225.h"
#include "main.h"
extern FontDef Font_7x10;
extern FontDef Font_11x18;
extern FontDef Font_16x26;
extern uint16_t IsButtonsPressed;
uint8_t IsModeSwithced = 1;
enum Modes CurrentMode = WALKINGCUBE;  //Режим по умолчанию
- ходячий куб
const uint16_t Timers_cube[TOTAL_MODES] =
```

```
    {
        260, 220, 350, 200, 300, 500, 100, 8
    };

uint16_t Speed = Timers_cube[WALKINGCUBE];
char ModesStrings[][30] =
    {
        "Rain", "Wall", "Woop-woop", "Jumping cube",
"Text", "Lit cube", "Walking cube", "Fading"
    };

void SwitchingModes()
{

    if (IsModeSwithced)
    {
     IsModeSwithced = 0;

     for (int8_t j = TOTAL_MODES/2; j > -(TOTAL_MODES/2); --
j)
     {
         char str1[13] = ""; //строка для вывода
         char str2[13] = ""; //вспомогательная строка
         uint8_t i;
         int8_t SubCurrentMode = (int8_t)CurrentMode + j < 0
? (TOTAL_MODES + (int8_t)CurrentMode + j) % TOTAL_MODES :
((int8_t)CurrentMode + j) % TOTAL_MODES;
         for (i = 0; i < ((13 - strlen(ModesStrings[SubCur-
rentMode])) / 2); ++i)
         {
          str1[i] = ' ';
          str2[i] = ' ';
         }
         str1[i + 1] = '\0';
         strcat(str1, ModesStrings[SubCurrentMode]);
         strcat(str1, str2);
         draw_string(60, 83 + 10 * -j, str1, Font_7x10,
COLOR_WHITE, COLOR_RED);
     }
     draw_char(50, 83, '>', Font_7x10, COLOR_WHITE,
COLOR_RED);
     draw_char(153, 83, '<', Font_7x10, COLOR_WHITE,
COLOR_RED);
```

```
     draw_bitmap(0, 103, 7, 10, Up_PTR);
     draw_string(8, 103, "LU", Font_7x10, COLOR_WHITE,
COLOR_RED);
     draw_bitmap(0, 63, 7, 10, Down_PTR);
     draw_string(8, 63, "LB", Font_7x10, COLOR_WHITE,
COLOR_RED);
    }
    if ((IsButtonsPressed >> LB) & 1)
    {
     IsButtonsPressed &= ~(1 << LB);
     CurrentMode = ((int8_t)CurrentMode + 1) % TOTAL_MODES;
//учет переполнения сверху
     Speed = Timers_cube[CurrentMode];
     IsModeSwithced = 1;
    }
    if ((IsButtonsPressed >> LU) & 1)
    {
     IsButtonsPressed &= ~(1 << LU);
     CurrentMode = (int8_t) CurrentMode - 1 < 0 ?
                 TOTAL_MODES : (int8_t) CurrentMode - 1;
//учет переполнения снизу
     Speed = Timers_cube[CurrentMode];
     IsModeSwithced = 1;                    //Перерисовываем
экран
    }
}


void ShowingCurrentMode()
{
    if (IsModeSwithced)
    {
     IsModeSwithced = 0;
     char str[] = "Current mode:     Speed:";
     draw_string(20, 83, str, Font_7x10, COLOR_WHITE,
COLOR_RED);
     sprintf(str, "Press LL for synchro");
     draw_string(20, 20, str, Font_7x10, COLOR_WHITE,
COLOR_RED);

    }
```

```
    /*---------------Выравнивание по центру(Режим)----------
-----------*/
    char str[30] = "";
    char str1[20] = ""; //вспомогательная строка для выра
нивания
    uint8_t i;
    for (i = 0; i < ((13 - strlen(ModesStrings[Current-
Mode])) / 2); ++i)
    {
     str[i] = ' ';
     str1[i] = ' ';
    }
    str[i + 1] = '\0';
    strcat(str, ModesStrings[CurrentMode]);
    strcat(str, str1);
    draw_string(20, 73, str, Font_7x10, COLOR_WHITE,
COLOR_RED);
    /*---------------Выравнивание по центру(Режим)----------
-----------*/

    /*---------------Отображение скорости(без выравнивания)-
-----------------*/
    if (CurrentMode == LIGHTCUBE)
    {
     sprintf(str, "N/A");
    }
    else
    {
     sprintf(str, "%d     ", Speed);
    }
    draw_string(153, 73, str, Font_7x10, COLOR_WHITE,
COLOR_RED);
    /*---------------Отображение скорости(без выравнивания)-
-----------------*/

    /*---------------Обработка кнопок(изменение скорости)---
---------------*/
    if ((IsButtonsPressed >> RB) & 1)
    {
     IsButtonsPressed &= ~(1 << RB);
     Speed += 10;
     if (Speed > 1000)
     {
```

```
        Speed = 1000;
     }
    }
    if ((IsButtonsPressed >> RU) & 1)
    {
     IsButtonsPressed &= ~(1 << RU);
     Speed = (int16_t)Speed - 10 < 0? 0 : Speed - 10;
    }
    /*----------------Обработка кнопок(изменение скорости)---
----------------*/
     }
```

Выпускная квалификационная работа выполнена мной самостоятельно.

Использованные в работе материалы из опубликованной научной литературы и других источников имеют ссылки на них.

Библиография: _____19_____ наименований.

Количество страниц: __66 и приложения на 50 страницах_

Отпечатано в 1 (одном) экземпляре, копия работы на электронном носителе.

«_____» июня 2021 года

Студент: _____Косяков Денис Александрович_____

Нормоконтроль

_____

_____

_____

_____

_____

_____

_____

_____

«__» _____ 2021 года

Нормоконтролер: _____