Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ» (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.03 «Прикладная информатика»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «Микропроцессорные системы»
на тему:

Устройство для имитации освещения планеты

Студент ИУ6–75Б _____ П.В. Землянский
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ Б.И. Бычков
(Подпись, дата) (И.О.Фамилия)

2022 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ6
(Индекс)
_____ А.В.Пролетарский
(И.О.Фамилия)
«2» сентября 2022 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Микропроцессорные системы

Студент группы ИУ6-75Б

Землянский Петр Валерьевич
(Фамилия, имя, отчество)

Тема курсовой работы: Устройство для имитации освещения планеты

Направленность курсовой работы – учебная

Источник тематики – кафедра

График выполнения работы: 25% к 4 нед., 50% к 8 нед., 75% к 12 нед., 100% к 16 нед.

Техническое задание:

Разработать на основе микроконтроллера AVR устройство для имитации освещения планеты. Устройство должно на макете Земли, разделенном на 24 сектора согласно часовым поясам, демонстрировать световой цикл дня от рассвета до заката. Смену освещения организовать в двух режимах: по часам реального времени и в ускоренном режиме (1 час соответствует 1 секунде). Текущее время выводить на дисплей. Макет должен иметь подсветку городов в ночной период (с 22:00 до 05:00). Режим работы выбирает пользователь с пульта оператора.

Разработать схему, алгоритмы и программу. Отладить проект на макете. Оценить потребляемую мощность.

Оформление курсовой работы:

1. Расчетно-пояснительная записка на 30 листах формата А4.
2. Перечень графического материала:
   а) схема электрическая функциональная;
   б) схема электрическая принципиальная.

Дата выдачи задания «02» сентября 2022 г. Дата защиты: «20» декабря 2022 г.

Руководитель курсовой работы _____ Б.И. Бычков
(Подпись, дата) (И.О.Фамилия)

Студент _____ П.В. Землянский
(Подпись, дата) (И.О.Фамилия)

РЕФЕРАТ

Записка 40 с., 34 рис., 8 табл., 11 источн., 4 прил.

МИКРОКОНТРОЛЛЕР, УСТРОЙСТВО, ОСВЕЩЕНИЕ, ИМИТАЦИЯ, СИГНАЛЫ, UART, I2C, AVR, ARDUINO.

Объектом разработки является устройство для имитации освещения планеты.

Цель работы – создание устройства с полным комплектом конструкторской документации и программным обеспечением для микроконтроллера семейства AVR.

Результатом проектирования является комплект конструкторской документации для изготовления устройства, исходные коды для программирования памяти микроконтроллера.

Устройство должно обладать следующими техническими характеристиками и возможностями:

– в режиме реального времени отображать освещение планеты;

– иметь возможность работы демонстрационного режима;

– обладать пультом оператора для выбора режима работы устройства;

– обеспечивать возможность вывода информации об устройстве – на текстовый дисплей;

Для написания программного обеспечения был использован язык C (компилятор AVR-GCC), разработка велась в среде Arduino IDE, а симуляция проводилась в среде Proteus ISIS 8, а так же был собран макет.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МС – Микросхема

МК – Микроконтроллер

ТЗ – Техническое задание

УГО – Условное графическое обозначение

I2C – (Inter-Integrated Circuit) Последовательная ассиметричная шина

SPI – (Serial Peripheral Interface) Последовательный периферийный интерфейс

UART – (Universal asynchronous receiver/transmitter) Универсальный асинхронный приемопередатчик

ВВЕДЕНИЕ

В данной работе произведена разработка устройства для имитации освещения планеты на основе микроконтроллеров семейства AVR.

Устройства предназначено для наглядного представления освещения планеты в зависимости от часового пояса в режиме реального времени. Также имеется возможность графического просмотра данных об устройстве, работы в режиме представления без освещения городов, без освещения солнцем, настройке яркости и в ускоренном режиме. Актуальность устройства обусловлена необходимостью усовершенствования гранд-макета России, а также в познавательных целях.

Преимуществом устройство является наглядность и отсутствие аналогов.

1 Конструкторская часть

Исходя из требований, изложенных в техническом задании, можно сделать вывод, что задачей работы устройства является поэтапное выполнение следующих шагов:

1) выводить световой цикл дня на макете планеты, разделенном на 24 часовых пояса.
2) возможность работы устройства в режиме смены часа равной 1 секунде;
3) выводить подсвету городов для ночной части цикла;
4) выводить текущее время на текстовый дисплей;
5) возможность управления работой программы через пульт оператора.

1.1 Анализ требований и принципа работы устройства

Согласно техническому заданию, требуется разработать устройство для имитации освещения планеты. Итоговое устройство должно на макете земли, разделенном на 24 сектора согласно часовым поясам, демонстрировать световой цикл дня от рассвета до заката.

Разработанная структурная схема представлена на рисунке 1.

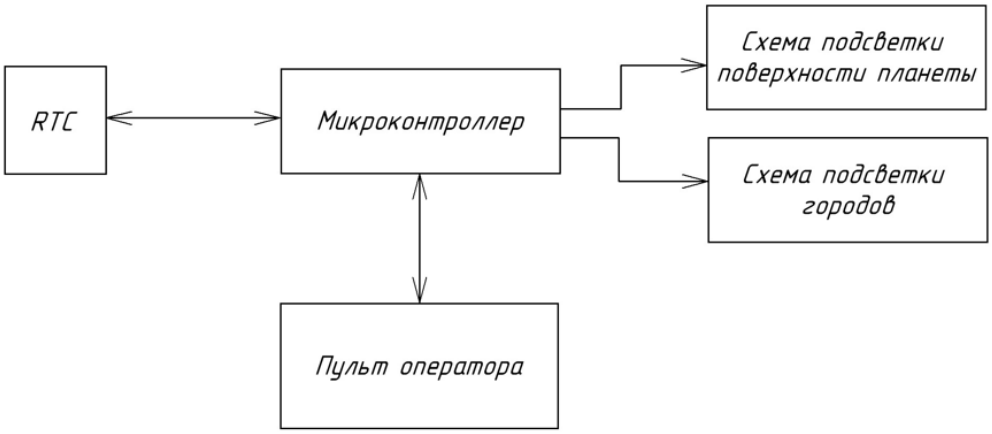


Рисунок 1 – Структурная схема устройства

Блоки схемы подсветки поверхности планеты и схемы подсветки городов необходимы для отображения освещения планеты солнцем и ночными фонарями соответственно. Блок RTC служит для точного подсчета времени, а также автономной работе в режиме реального времени за счет отдельного питания. Данные времени при компиляции программы заносятся в блок RTC из микроконтроллера один раз, после чего считываются из RTC регулярно. Затем информация, полученная с RTC, с помощью микроконтроллера преобразуется по функции перехода в формат rgb цветов и выводится на блок схемы подсветки поверхности планеты, а также по функции перехода для подсветки городов выводится на блок подсветки городов. Пульт оператора позволяет управлять режимом работы устройства с помощью текстового дисплея и набора кнопок.

1.2 Разработка и описание функциональной схемы

В этом подразделе приводится функциональное описание работы устройства.

1.2.1 Описание платы Arduino UNO3 и микроконтроллера ATmega328P

По заданию требуется использовать микроконтроллер семейства AVR.

Среди всего многообразия микроконтроллеров AVR на рынке для данной работы была выбрана плата Arduino UNO3 на базе микроконтроллера ATmega328P за доступность и готовое решение.

Arduino UNO3 [1] – это плата микроконтроллера с открытым исходным кодом, основанная на микроконтроллере ATmega328P, разработанная Arduino.cc. Плата оснащена наборами цифровых и аналоговых входов/выходов (I/O) контакты, которые могут быть подключены к различным платам расширения и другим схемам. Плата имеет 14 контактов цифрового ввода/вывода (шесть из которых могут работать с ШИМ), 6 контактов аналогового ввода/вывода и программируется с помощью Arduino IDE (интегрированная среда разработки) через USB-кабель типа B. Он может питаться от USB-кабеля. Принципиальная схема и схема расположения выводов платы представлена на рисунках 2 и 3 соответственно.

ATmega328P [2] – 8-разрядный микроконтроллер, основанный на AVR RISC архитектуре. ATmega328З имеет 23 линии ввода-вывода общего назначения, 32 рабочих регистра общего назначения, 3 гибких таймер/счетчика с режимами сравнения, внутренние и внешние прерывания, последовательный программируемый USART, I2C, последовательный порт SPI и программируемый сторожевой таймер с внутренним генератором. Устройство работает от 1,8 до 5,5 вольт. Устройство достигает пропускной способности, приближающейся к 1 миллиону инструкций в секунду на 1 МГц. Структурная схема и схема расположения выводов микросхемы МК представлена на рисунках 2 и 3 соответственно.

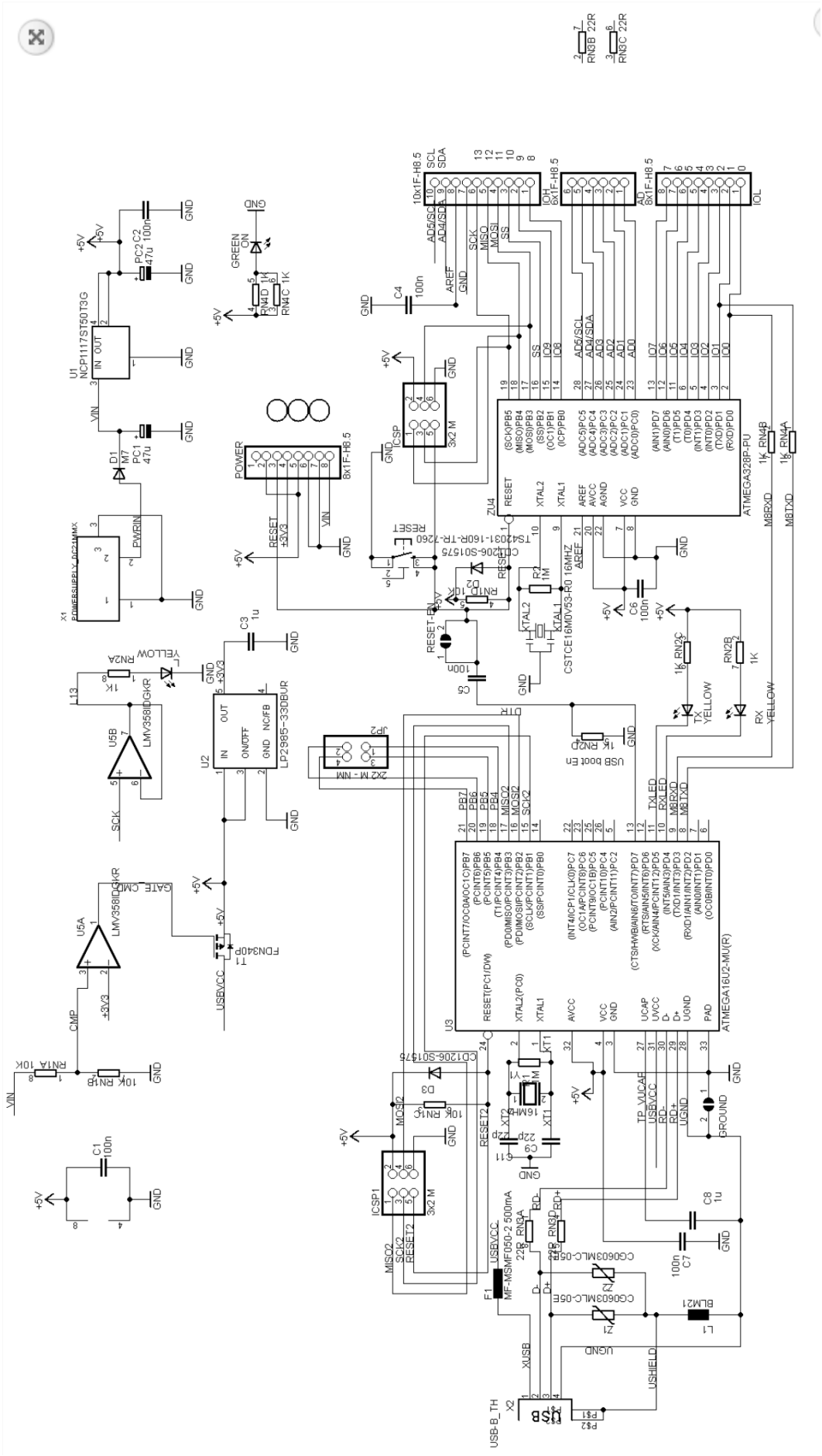


Рисунок 2 – принципиальная схема Arduino UNO3

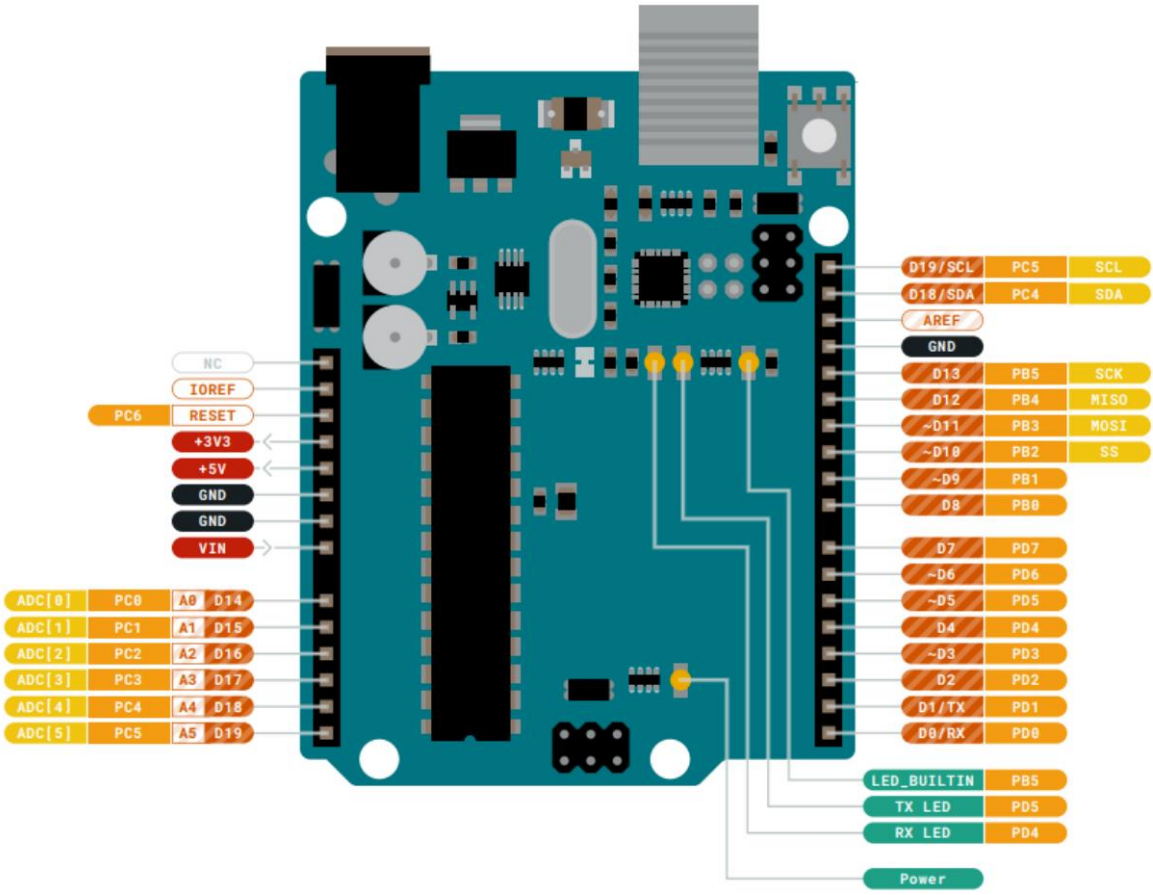


Рисунок 3 – Схема расположения выводов платы Arduino UNO3

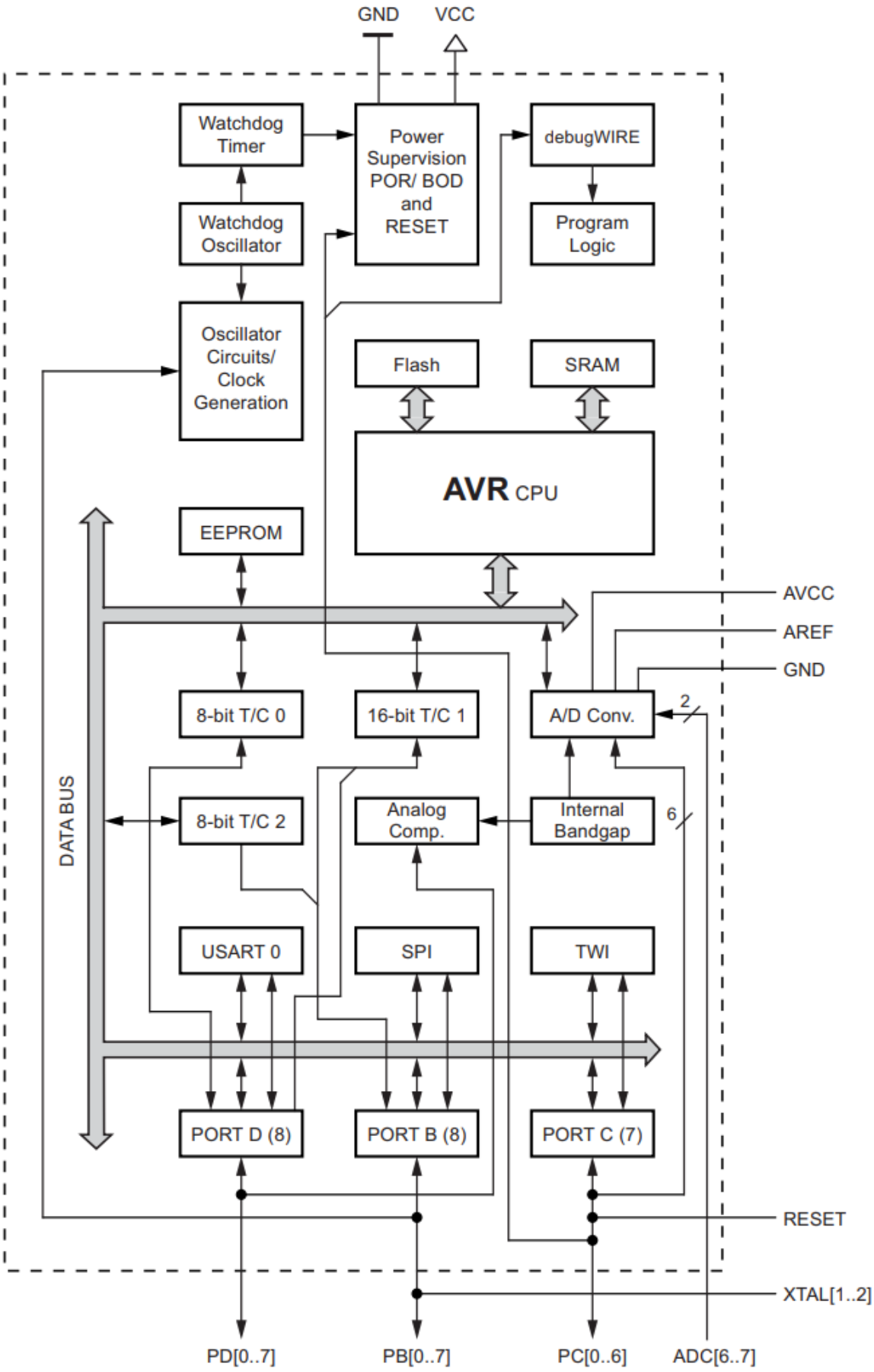


Рисунок 4 – Структурная схема МК ATmega328P

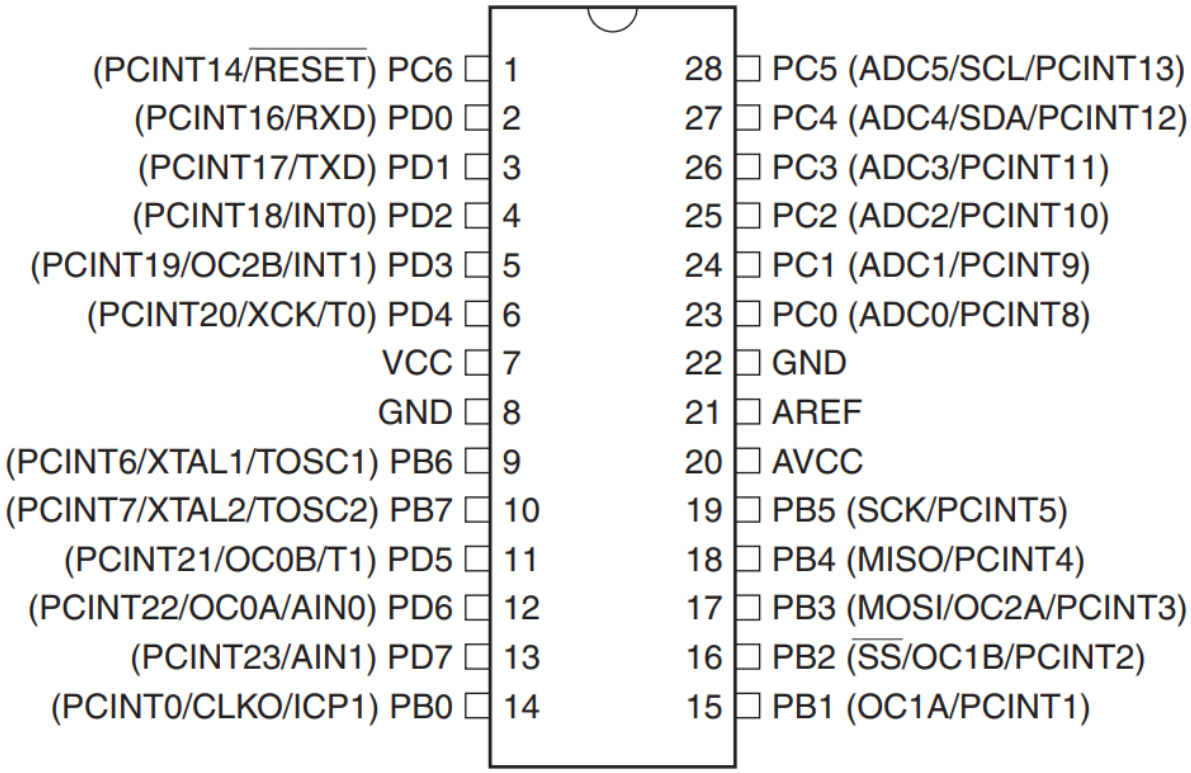


Рисунок 5 – Схема расположения выводов МК ATmega238P

Данный МК обладает следующими характеристиками:

1) Архитектура:

– разрядность – 8 бит;

– рабочая частота – от 1 до 20 МГц;

– количество инструкций (команд) – 131.

2) Память:

– 32 Кбайт внутрисистемно программируемой flash-памяти;

– 1 Кбайт EEPROM;

– 2 Кбайт встроенной SRAM (с возможностью организации внешней области памяти размером до 64 Кбайт).

3) Периферийные устройства:

– Два 8-разрядный таймер-счетчик с отдельным предделителем и режимом компаратора;

– Один 16-разрядный таймер-счетчик с отдельным предделителем, режимом компаратора и режимом захвата фронтов;

– Шесть каналов ШИМ (широтно-импульсная модуляция);

– Программируемый последовательный UART;

– Последовательный интерфейс SPI, работающий в режимах master и slave;

– Последовательный интерфейс I2C.

4) Порты и корпуса (рисунок 5):

– 14 цифровых выводов МК;

– 6 аналоговых выводов МК;

5) Напряжение: 1.8–5.5В.

В микроконтроллерах AVR семейства Mega реализована Гарвардская архитектура, в соответствии с которой разделены не только адресные пространства памяти программ и памяти данных, но также и шины доступа к ним. Способы адресации и доступа к этим областям памяти также различны. Такая структура позволяет центральному процессору работать одновременно как с памятью программ, так и с памятью данных, что существенно увеличивает производительность. Каждая из областей памяти (ОЗУ и EEPROM) также расположена в своем адресном пространстве. Карта распределения памяти МК ATmega328P представлена на рисунке 6.

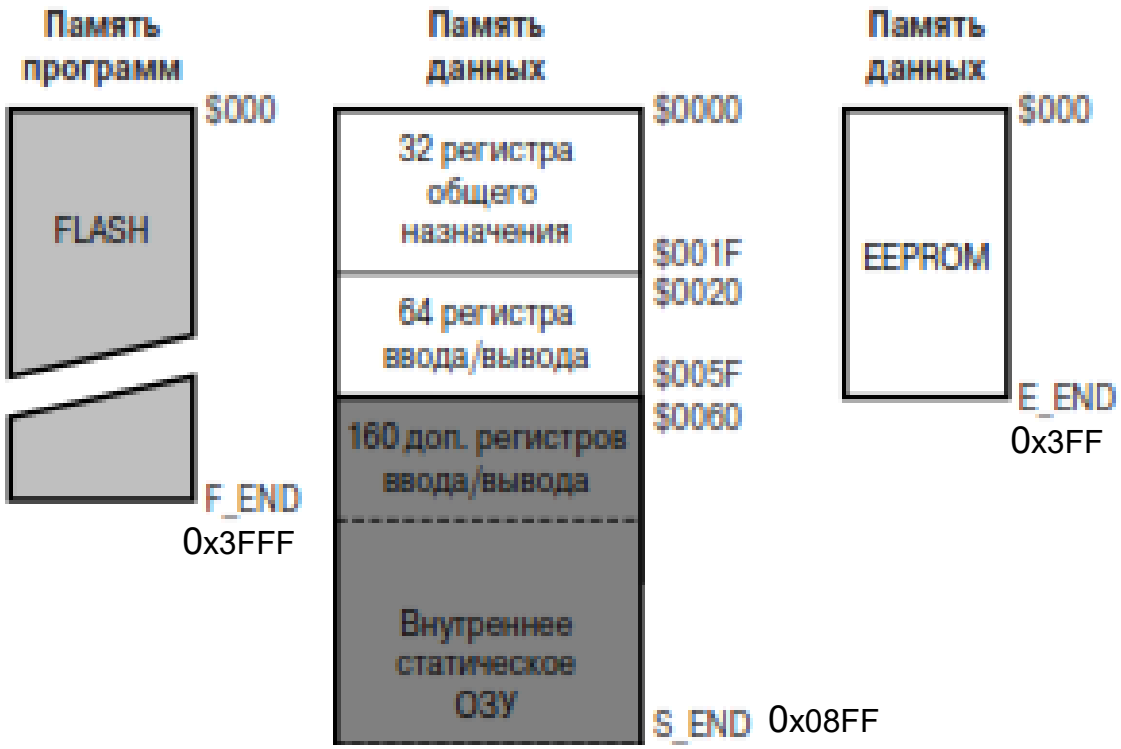


Рисунок 6 – Карта распределения памяти МК ATmega328P

1.2.1.2 Используемые компоненты МК

Описание компонентов, задействованных в работе микроконтроллера:

Порт C – порт вывода, используется для передачи данных, полученных с кнопок пульта оператора;

– *PC0:PC3* – выходы микроконтроллера по отношению к кнопкам SB3:SB0 соответственно.

Порт B – порт ввода/вывода использующийся для взаимодействия со схемой подсветки городов:

– *PB2,PB*4 – выходы микроконтроллера по отношению к DS и SH_CP регистра сдвига соответственно.

Порт D – порт ввода/вывода:

– *PD0:PD1* – выводы микроконтроллера используемые для программирования МК;

– *PD2* – вывод микроконтроллера по отношению к ST_CP регистра сдвига.

Указатель стека – указатель, хранящий значение вершины стека, используется при вызове подпрограмм и работе с локальными временными переменными;

Программный счетчик – регистр процессора, который указывает, какую команду нужно выполнять следующей;

Регистры общего назначения – регистры, предназначенные для хранения операндов арифметико-логических инструкций, а также адресов или отдельных компонентов адресов ячеек памяти;

SRAM – статическая память с произвольным доступом, используется для хранения объявленный переменных;

Память Flash – память программ, которая хранит код;

Регистр команд – содержит исполняемую в текущий момент (или следующий) команду, то есть команду, адресуемую счетчиком команд;

Декодер – блок, выделяющий код операции и операнды команды, а затем вызывающий микропрограмму, которая исполняет данную команду;

АЛУ – блок процессора, который под управлением устройства управления служит для выполнения арифметических и логических преобразований над данными;

Регистр SREG – регистр состояния, содержит набор флагов, показывающих текущее состояние работы микроконтроллера;

Логика программирования – устанавливает логику того, как программа будет вшита в МК;

Таймеры/счетчики – средство микропроцессора, служащее для измерения времени, реализации задержек, формирование ШИМ сигналов и количества входящих импульсов – используется для задержек;

Прерывания – механизм, который позволяет микроконтроллеру реагировать на внешние события – используется для сигнализации выбора канала внешних аналоговых сигналов;

Генератор – используем тактирование от внешнего кварца;

TWI – модуль микроконтроллера с последовательным интерфейсом I2C;

DebugWire – модуль отладки программы во время исполнения;

UART – модуль микроконтроллера для работы с последовательным интерфейсом – используется для программирования МК.

1.2.2 Схема подсветки поверхности земли

Схема подсветки поверхности земли должна обеспечивать подсветку карты, разделенной на 24 секции, от заката до рассвета. Соответственно иметь возможность изменять передаваемый цвет.

Подходящим вариантом является адресная светодиодная RGB лента WS2812B [3]. WS2812B необходимо напряжение питания 5В, а также МС потребляет 20мА на один светодиод. УГО WS2812B представлена на рисунке 7. Назначение выводов описано в таблице 2.

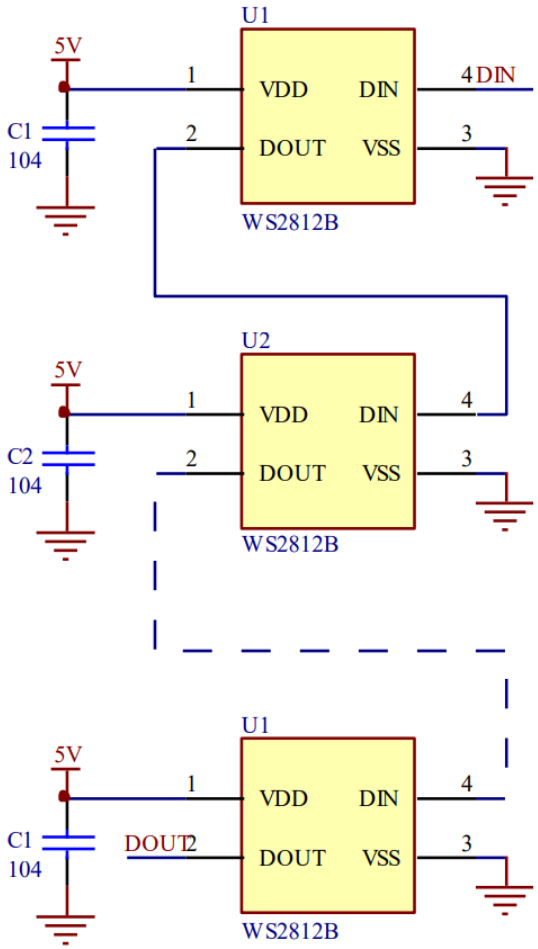


Рисунок 7 – УГО WS2812

Таблица 1 – Назначение выводов WS2812

Вывод	Описание	Применение
VDD	источник питания (напряжение)	подключен к питанию 5В
DIN	вход данных	подключен к PB1 МК/DOUT предыдущего
VSS	источник питания (земля)	подключен к земле
DOUT	выход данных	подключен к DIN следующего

WS2812 передает сигналы по одной линии, следовательно это последовательный протокол передачи данных. Схема работы протокола представлена на рисунке 8.

Data transfer time(TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50μs	

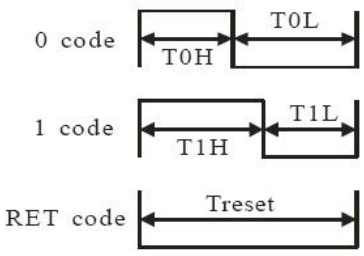


Рисунок 8 – Протокол передачи данных в WS2812

Хоть светодиоды ws2812b называют адресными, но по факту они не адресуются. Данные передаются по цепочке друг за другом, по принципу вошел - вышел. Управляющие пакеты светодиодам передаются по 3 байта или пакет из 24 бит, по одному байту на каждый цвет одного RGB светодиода. Первым в пакете передается байт данных для зеленого цвета, следующий байт для красного, и третий байт пакета предназначен для синего цвета. Направление передачи бит идет - от старшего к младшему. После каждого пакета должна быть выдержана пауза 50 мкс. Пауза больше 100 мкс воспринимается как окончание передачи. Длительность передачи одного бита ноля или единицы, должны быть равны 1.25 мкс. Бит логической единицы или ноля в свою очередь формируется из двух импульсов. Единица должна быть - высокий уровень длительностью 0.8 мкс и низкий уровень длительность 0.45 мкс. Логический ноль будет выглядеть следующим образом - высокий уровень длительностью 0.4 мкс и низкий уровень длительностью 0.85 мкс. Допускаются небольшие отклонения до 150 нс на импульс. На изображении все наглядно изображено.

1.2.3 Схема подсветки городов

Схема подсветки городов должна обеспечивать точечную подсветку городов в ночное время с 22.00 до 05.00. Подсветка городов имитирует работу фонарей в городе в ночное время суток, следовательно, подсветка должна включаться, в пределах одного часового пояса, одновременно.

Для этой задачи был выбран регистр сдвига 74HC595 [4]. УГО регистра представлено на рисунке 9. Назначение выводов описано в таблице 3.

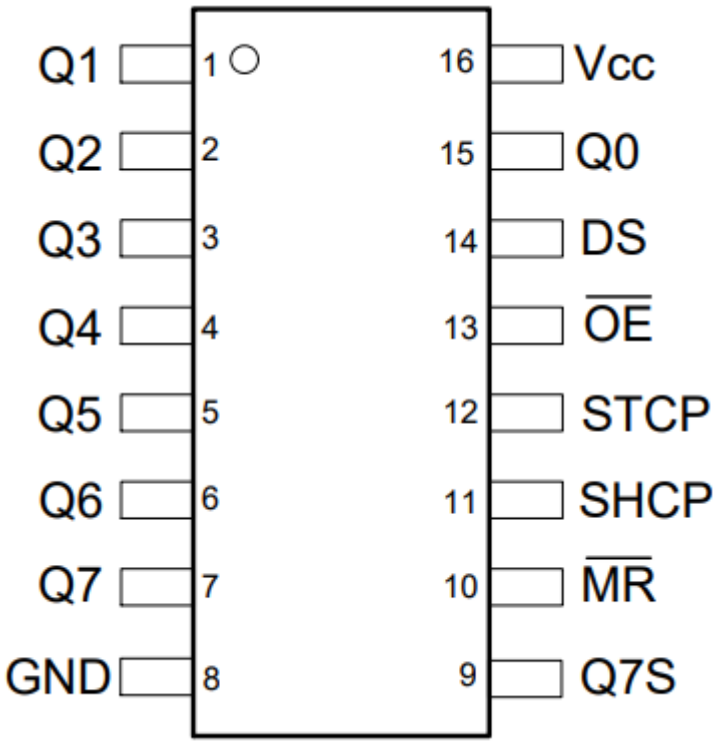


Рисунок 9 – УГО 74HC595

Таблица 3 – Назначение выводов 74HC595

Вывод	Описание	Применение
$Q0 - Q7$	цифровой вывод	подключен к диодам
STCP	тактовый вход для записи в выходной регистр	подключен к PD2 МК
SHCP	тактовый вход для записи во входной регистр	подключен к PB4 МК
Q7S	выход последовательных данных	Подключен к DS следующего регистра
$\overline{OE}$	инверсный вход разрешения вывода	подключен к земле
DS	вход последовательных данных	подключен к PB2 МК/Q7S предыдущего
$\overline{MR}$	инверсный вход сброса	подключен к питанию
GND	источник питания (земля)	подключен к земле
Vcc	источник питания (напряжение)	подключен к питанию

1.2.4 RTC

По заданию требуется использовать часы реального времени, для определения времени на устройстве.

DS1307 [5] – часы реального времени работающие по I2C.

УГО микросхемы изображено на рисунке 10. Назначение выводов DS1307 описано в таблице 4.

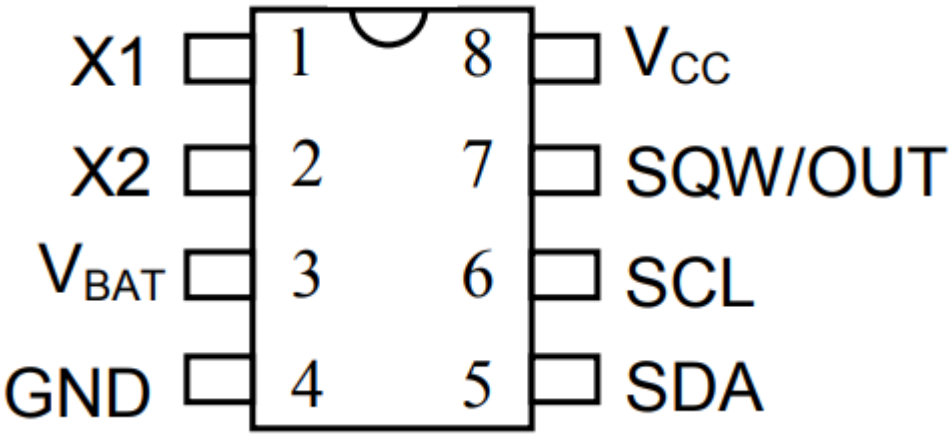


Рисунок 10 – УГО DS1307

Таблица 4 – Назначение выводов 74157

Вывод	Описание	Применение
X1	вход кварцевого резонатора	подключен к XTAL1
X2	вход кварцевого резонатора	подключен к XTAL2
SQW/OUT	выходной драйвер	подключены к земле

(Продолжение таблицы 4)

SCL	тактовый вход I2C	подключен к PC5 МК
SDA	вход данных I2C	подключен к PC4 МК
V_{BAT}	вход питания батареей +3В	подключен к батарее
VCC	источник питания (напряжение)	подключен к питанию
GND	источник питания (земля)	подключен к земле

1.2.4.1 Описание интерфейса I2C

I2C или (Inter-Integrated Circuit) последовательная ассиметричная шина для связи между интегральными схемами внутри электронных приборов. Использует две двунаправленные линии связи (SDA и SCL), применяется для соединения низкоскоростных периферийных компонентов с процессорами и микроконтроллерами.

Данные передаются по двум проводам — провод данных и провод тактов. Есть ведущий(master) и ведомый (slave), такты генерирует master, ведомый лишь поддакивает при приеме байта. Всего на одной двупроводной шине может быть до 127 устройств. Схема подключения устройств показана на рисунке 11.

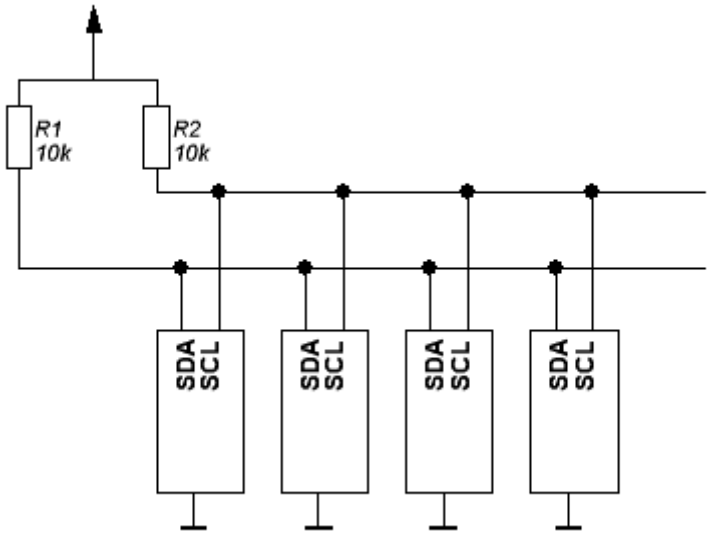


Рисунок 11 – схема подключения устройств по I2C

Передача/Прием сигналов осуществляется прижиманием линии в 0, в 1 устанавливается по умолчанию, за счет подтягивающих резисторов. Вся передача данных состоит из стартовой посылки, битов и стоповой посылки. Порядок изменения уровня на шинах задает тип посылки.

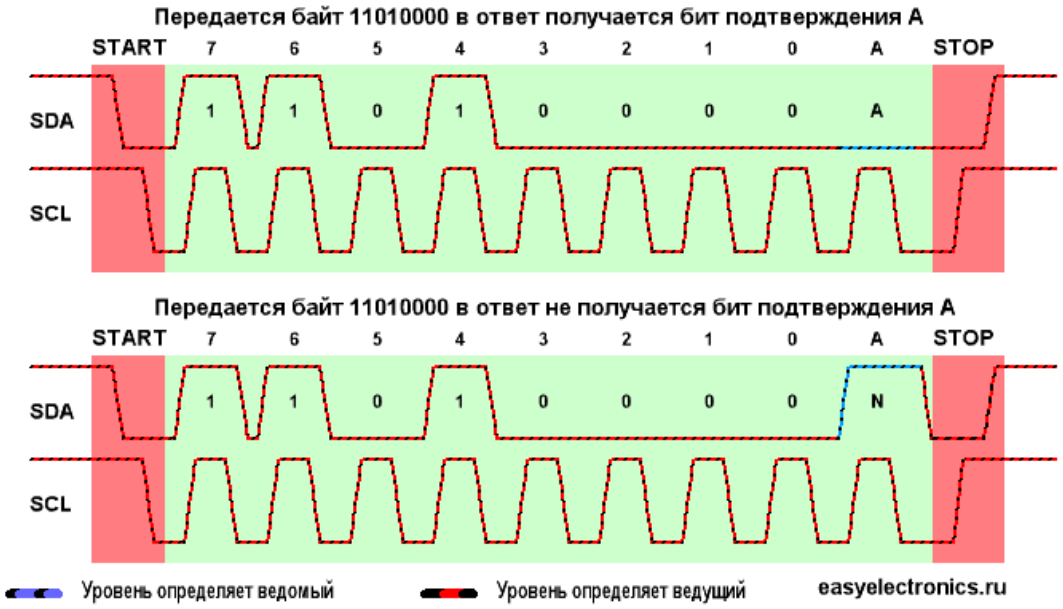


Рисунок 12 – схема передачи данных по I2C

После старта передача одного бита данных идет по тактовому импульсу. То есть когда линия SCL в нуле master или slave выставляют бит на SDA (прижимают — если 0 или не прижимают — если 1 линию SDA) после чего SCL отпускается и master/slave считывают бит. Таким образом, у нас протокол совершенно не зависит от временных интервалов, только от тактовых битов. Данные отправляются пакетами по 9 бит (8 бит данные и 1 бит подтверждения). После прихода стартового бита отправляется байт данных адреса отправки, после чего ожидается ответ от slave, о начале приема данных. Полный пакет представлен на рисунке 13.

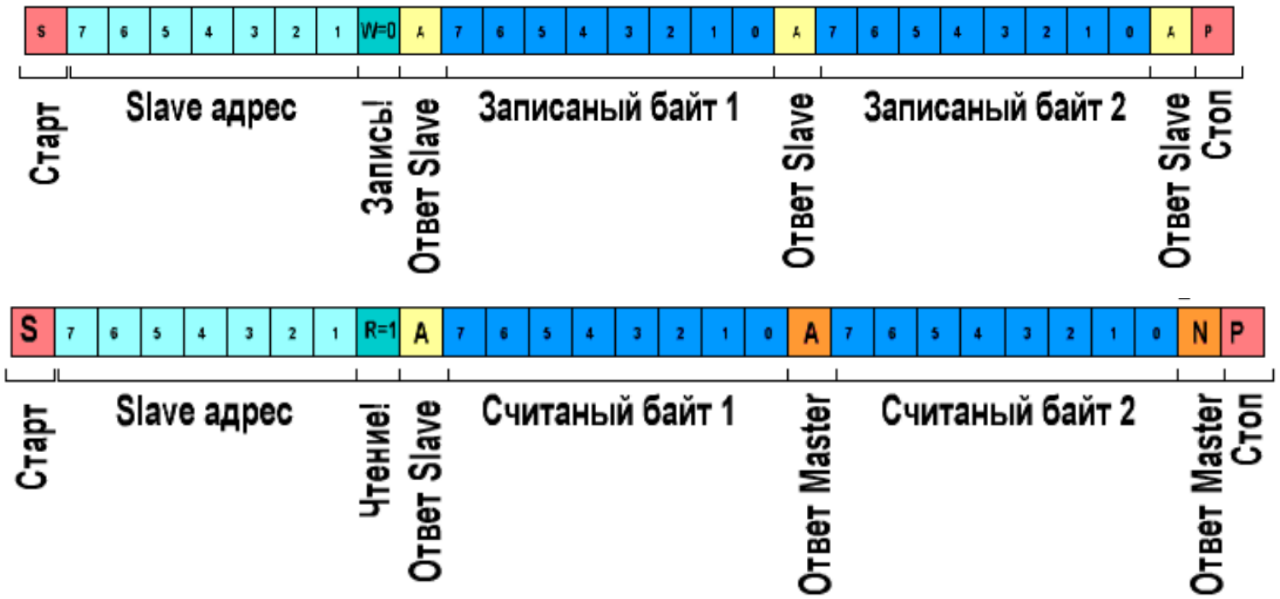


Рисунок 13 – схема передачи данных по I2C при записи и чтении

1.2.5 Схема пульта оператора

По заданию требуется использовать пульт оператора. В качестве пульта оператора используется схема управления меню и текстовый дисплей.

1.2.5.1 Текстовый дисплей

Для вывода меню состоящего из режимов понадобится четырехстрочный символьный дисплей (1-я строка для объявления потенциального выбора, 2-я и 3-я – для альтернативного выбора, 4-я для вывода времени). Используется последовательная запись данных через I2C, так как она экономит количество используемых портов несколько раз. Соответственно нужен расширитель портов I2C и контроллер, на базе которого построен дисплей, должен иметь возможность принимать данные параллельной записью (8 бит). В качестве расширителя портов подходит PCF8574 [6]. В качестве контроллера подходит HD44780 – принимает данные по параллельной шине и просто программируется. LCD дисплеем на базе этого микроконтроллера является LM044L [7].

УГО расширителя портов и текстового дисплея изображены на рисунках 14 и 15 соответственно, назначение выводов описаны в таблицах 5 и 6 соответственно.

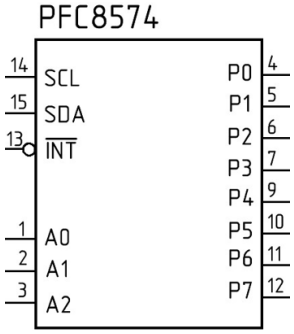


Рисунок 14 – УГО микросхемы PCF8574

Таблица 5 – Назначение выводов PCF8574

Вывод	Описание	Применение
P0 – P3	биты вывода	подключены к выводам RS,RW,E LM044L
P4 – P7	биты вывода	подключены к выводам D4 – D7 LM044L
A0 – A2	адресные биты	подключен к земле
$\overline{\text{INT}}$	инверсный вывод прерывания	подключен к земле
SCL	тактовый вход I2C	подключен к PC5 МК
SDA	вход данных I2C	подключен к PC4 МК

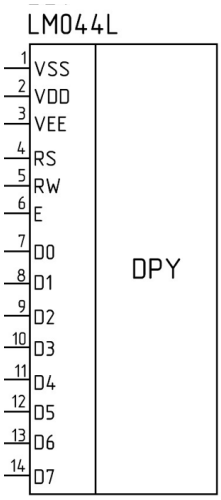


Рисунок 15 – УГО микросхемы LM044L

Таблица 6 – Назначение выводов LM044L

Вывод	Описание	Применение
D0 – D7	биты интерфейса	подключены к порту A МК-обработчика
RS	выбор регистра	подключен к PE1 МК-обработчика
RW	0/1 – запись/чтение	подключен к земле
E	сигнал включение (строб)	подключен к PE0 МК-обработчика
VSS	источник питания (земля)	подключен к земле
VDD	источник питания (напряжение)	подключен к питанию
VEE	управление контрастностью	подключен к переменному резистору

1.2.6 Схема управления меню

В качестве схемы управления меню выступает набор кнопок. Работа со схемой управления меню состоит из поэтапного выбора из предложенного меню путем нажатия определенной кнопки. Функционал кнопок представлен в таблице 7. При запуске программы на дисплей выводится начальное сообщение «Меню». Далее, сообщения выводятся после ввода пользователем. Команды и их результаты подробно описаны в таблице 6. Структура меню представлена на рисунке 16.

Таблица 7 – функционал кнопок пульта управления

Обозначение	Название	Результат ввода
SB0	вверх	Перемещение по текущему уровню меню на 1 единицу вверх

(продолжение таблицы 7)

SB1	Вниз	Перемещение по текущему уровню меню на 1 единицу вниз
SB2	Влево	Перемещение на один уровень выше
SB3	Вправо	Перемещение на один уровень ниже/выбор конечного пункта

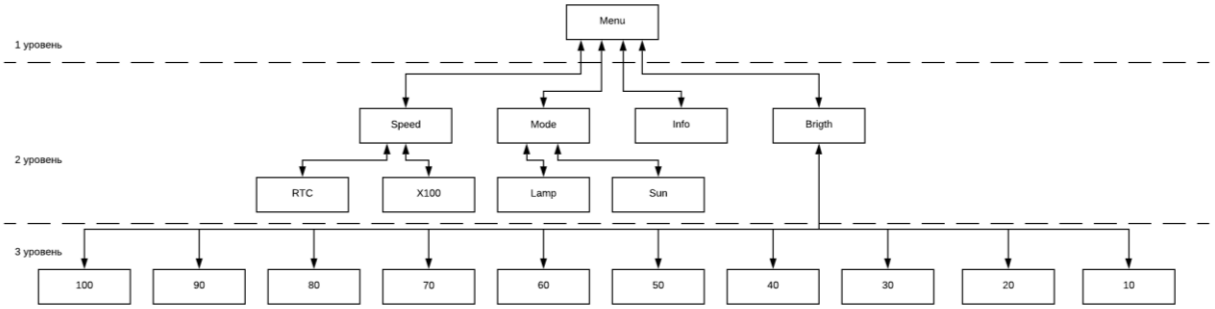


Рисунок 16 – структурная схема меню

Управляя кнопками вверх, вниз можно перемещаться в пределах одного уровня меню. Управляя кнопками влево, вправо можно переходить от одного уровня к другому.

1.2.7 Построение функциональной схемы

Функциональная схема разрабатываемого устройства, разработанная на основе структурной схемы с использованием выбранных компонентов, изображена на рисунке 17 и представлена в приложении Б.

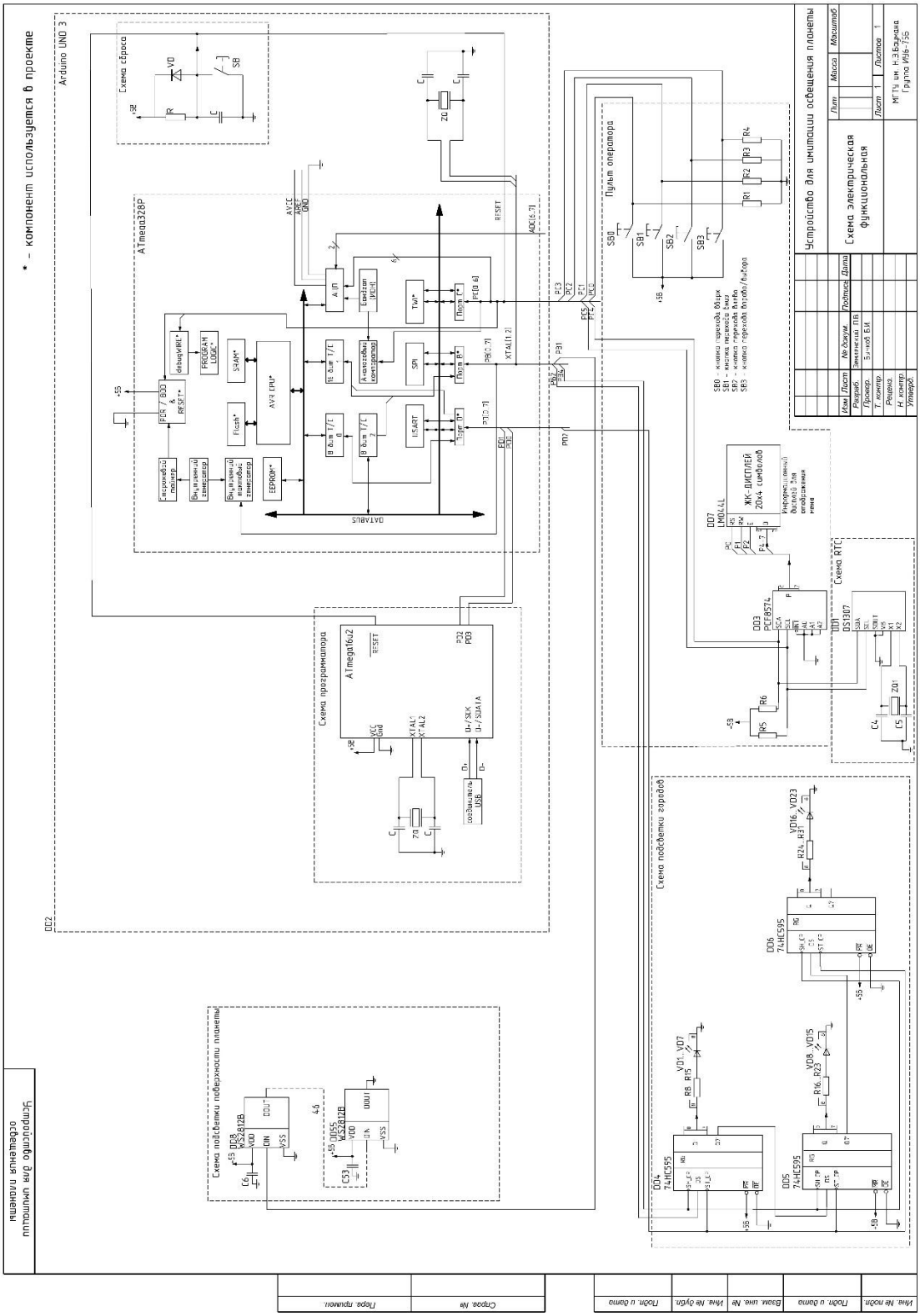


Рисунок 17 – Функциональная схема устройства

1.3 Разработка и описание принципиальной схемы

В этом подразделе произведен анализ и детализация функциональной схемы, результатом которых стала принципиальная схема.

1.3.1 Разъемы для подключения внешних устройств

В качестве соединителя для программирования микроконтроллера и получения питания у Arduino UNO3 используется порт USB TYPE B, условное обозначение на схеме XP1(рисунок 18);

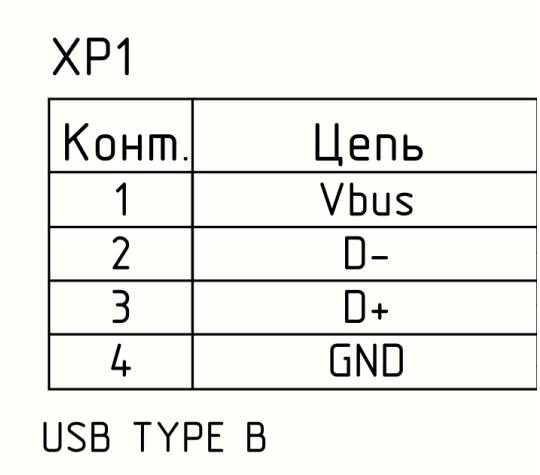


Рисунок 18 – Разъем USB TYPE B

Цепи «D+» «D-» отвечают за передачу данных, вместе образуя симметричное соединение. Данные через порт USB заносятся в программатор, после чего по протоколу UART, программа заносится в ATmega328P.

Для связи микроконтроллера и программатора используется последовательный порт UART, работающий по стандартному протоколу передачу. Протокол передачи данных на уровне последовательности логических значений у UART иллюстрируется следующим примером диаграммы передачи байта (рисунок 19).

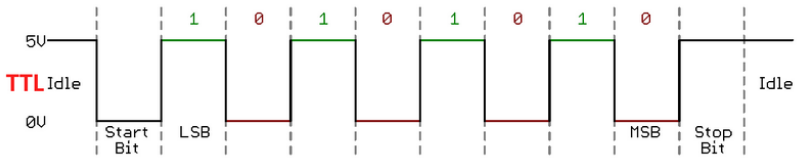


Рисунок 19 – Передача байта по UART-стандарту

1.3.2 Подключение цепи питания

Источники питания на 12В встречаются и используются чаще чем источники на 5В, как в бытовых, так и в рабочих целях. Однако большинство элементов схемы работают от напряжения 5В. Также не все источники питания на 5В способны обеспечить постоянные 5В без просадок по напряжению. Поэтому, на схему лучше подавать напряжение 12В.

Следовательно, необходимо добавить на схему устройство, которое снижает входные 12В до необходимых 5В.

Популярным и дешевым решением является установка линейного стабилизатора напряжения. На данной схеме используется стабилизатор LM7805. Для поддержки нормального режима работы стабилизатора необходимо шунтировать полярными конденсаторами вход и выход стабилизатора (рисунок 20).

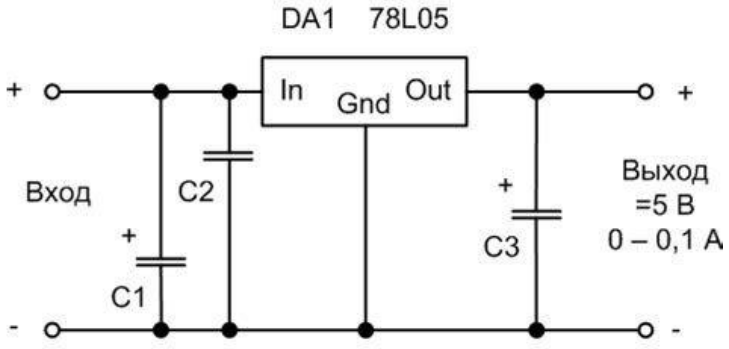


Рисунок 20 – Пример подключение линейного стабилизатора

Заметим, что стабилизатор напряжения на является трансформатором и лишняя мощность рассеивается на радиаторе устройства. При перегреве (+145 °C) устройства уходит в защитный режим на небольшой промежуток времени для того, чтобы снизить температуру. В схемах с большим потреблением (более 500мА в постоянном режиме) следует установить линейному стабилизатору напряжение дополнительной радиатор, или заменить стабилизатор на импульсный понижающий преобразователь (buck-конвертер) [8].

Для подключения питания к схеме, будет использовано гнездо питания DS-201, имеющий диаметр центрального проводника – 2 мм и диаметр Jack-а – 2 мм. Условное обозначение на схеме – XS1 (рисунок 21).

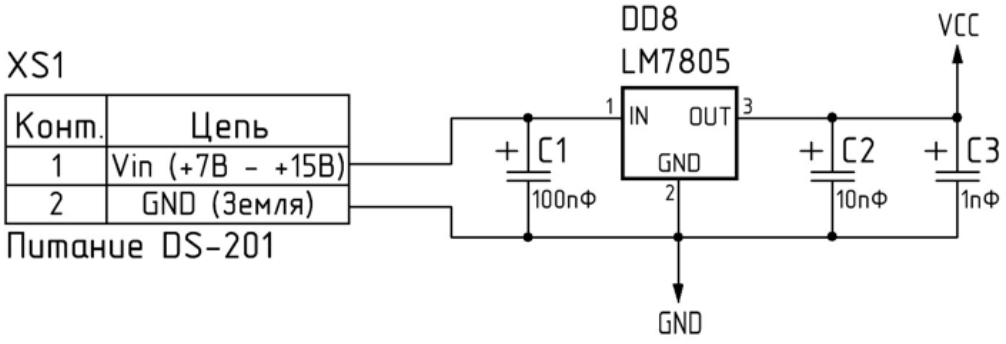


Рисунок 21 – Гнездо DS-201

1.3.3 Генератор тактовых импульсов

Для работы RTC необходим внешний тактовый генератор. Он подключается к RTC через входы X1 и X2. Частота резонатора равна 32 КГц для соответствующей тактовой частоты RTC .

Схема резонаторов для подключения к RTC изображена на рисунке 22.

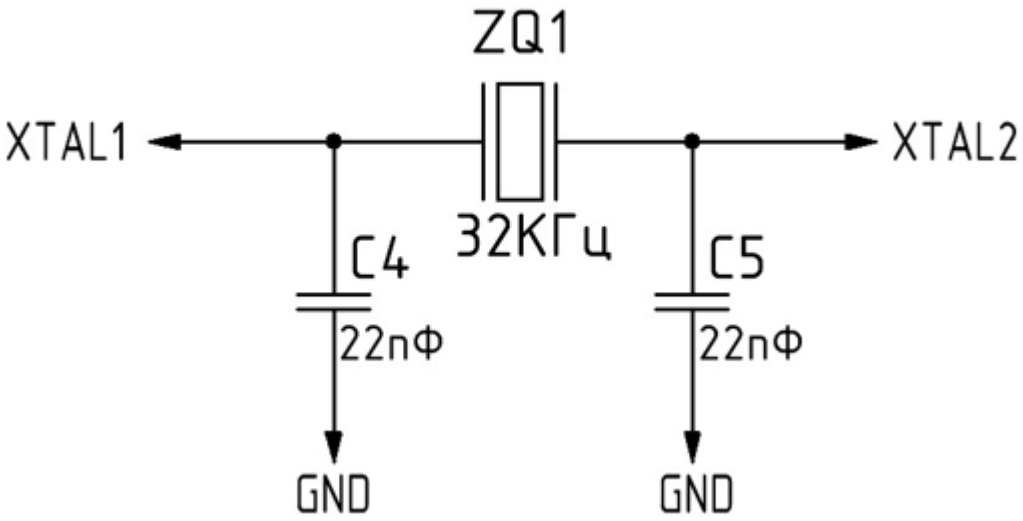


Рисунок 22 – Схема генератора тактовых импульсов

1.3.4 Построение принципиальной схемы

Принципиальная схема разрабатываемого устройства, разработанная на основе выбранных компонентов, изображена на рисунке 23 и представлена в приложении В.

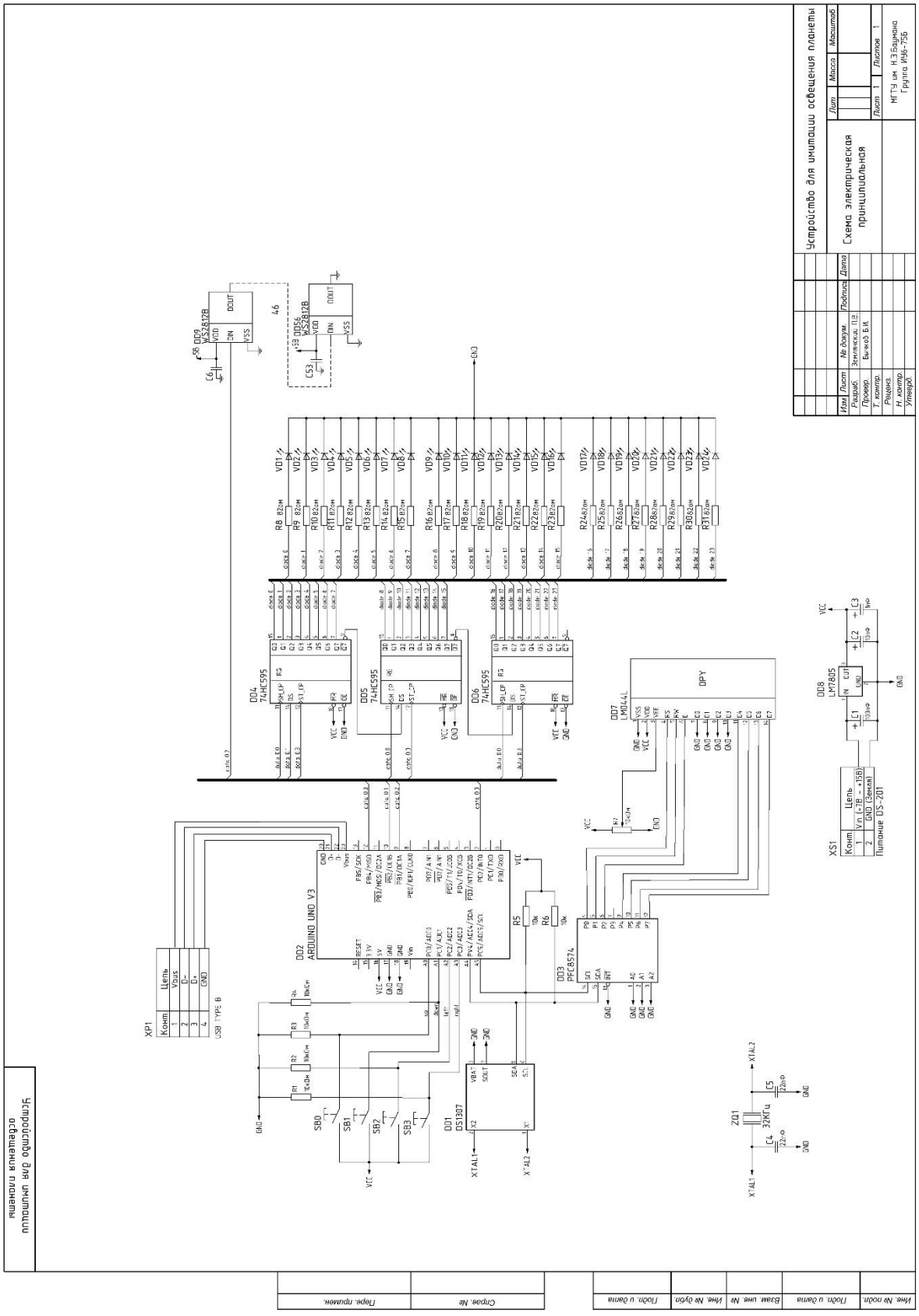


Рисунок 23 –Принципиальная схема устройства

1.4 Расчет потребляемой мощности проектируемого устройства

Рассчитаем мощность, потребляемую разрабатываемым устройством. В качестве питания подается напряжение $U_{cc} = 5\text{В}$. Мощность рассчитывается по формуле:

$$P = U_{cc} * I_{cc}$$

где U_{cc} – напряжение питания, равное 5В;

I_{cc} – ток потребления микросхемы, указанный для каждой схемы в ее документации.

Таким образом значение мощности получается в результате перемножения этих двух значений для всех микросхем кроме линейного стабилизатора.

Результат расчетов для микросхем представлен в таблице 8.

Таблица 8 – Расчет мощности, потребляемый микросхемами

Тип ИМС	Количество ИМС, шт	Максимальный ток потребления микросхемы, мА	Максимальная мощность, потребляемая элементом, мВт	Суммарная мощность, мВт
Arduino UNO3	1	100	500	500
DS1307	1	1.5	7.5	7.5
74HC595	3	0.008	0.04	0.12
LM016L	1	3	15	15
PCF8574	1	100	500	500
WS2812	48	10	50	2400
25AA160B	1	6	30	30
			Итого:	3452.62

Максимальная суммарная мощность, потребляемая микросхемами $P_{\text{МИКР}}$ равна 3452.62 мВт.

Рассмотрим теперь рассеивание мощности линейного стабилизатора. Иллюстрация представлена на рисунке 24.

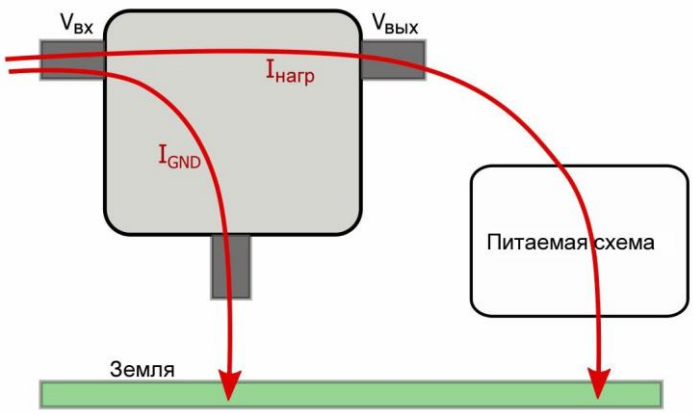


Рисунок 24 – Определение мощности, рассеиваемой линейным стабилизатором

На этой иллюстрации показаны два пути протекания тока в линейном стабилизаторе. Путь от входного вывода непосредственно к земле называется током на землю (I_{GND}), а путь

от входного вывода к земле через питаемую цепь – ток нагрузки ($I_{\text{НАГР}}$). Внутреннее рассеивание мощности в результате протекания этих двух токов будет равно [9]:

$$P_{I_{GND}} = I_{GND} \times V_{\text{ВХ}}$$

$$P_{I_{\text{НАГР}}} = I_{\text{НАГР}} \times (V_{\text{ВХ}} - V_{\text{ВЫХ}})$$

Таким образом, общая рассеиваемая мощность внутри стабилизатора будет равна:

$$P_{\text{СТАБ}} = (I_{GND} \times V_{\text{ВХ}}) + (I_{\text{НАГР}} \times (V_{\text{ВХ}} - V_{\text{ВЫХ}}))$$

$V_{\text{ВХ}}$ возьмем максимально возможное для элемента, равное 15В; $V_{\text{ВЫХ}}$ – равным 5В; $I_{\text{НАГР}}$ равна сумме максимальных значений тока всех остальных микросхем ($4 + 1 + 48 + 3 + 2 + 10 + 6 = 74$мА), I_{GND} дана в документации линейного стабилизатора и равна 8ма.

Получим $P_{\text{СТАБ}} = (8 \times 15) + \big(74 \times (15 - 5)\big) = 120 + 740 = 860$мВт.

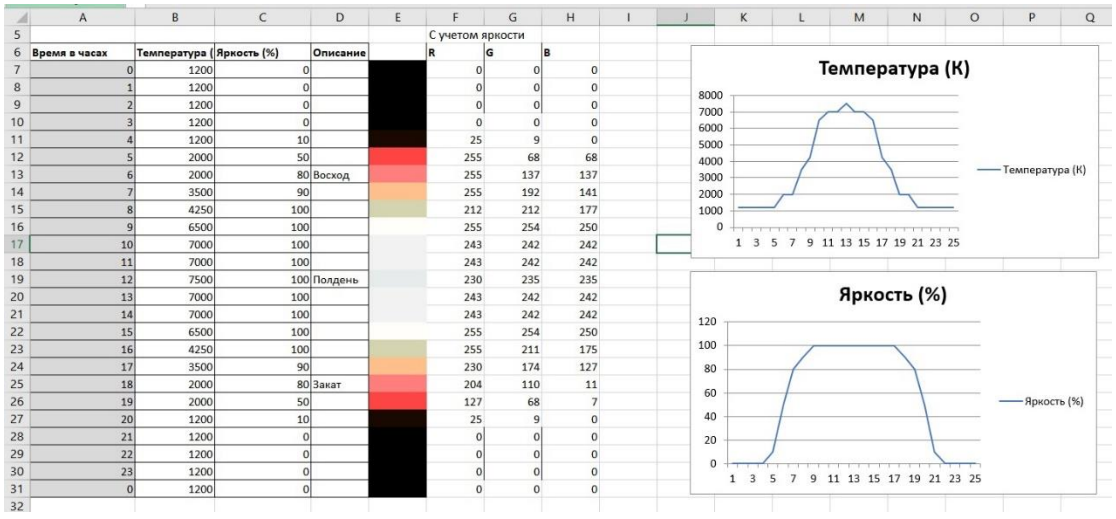
Тогда суммарная общая мощность, потребляемая устройством, будет равна $P_{\text{ОБЩ}} = P_{\text{МИКС}} + P_{\text{СТАБ}} = 3452.62 + 860 = 4315.62$ мВт.

1.5 Разработка алгоритмов

Была разработана программа для программирования микроконтроллера в среде Arduino IDE(среде для разработки под платы Arduino). Рассмотрим алгоритм работы программы.

1.5.1 Алгоритм обработки RTC

Алгоритм заключается в получении данных из RTC и преобразовании времени в цвет. Схема перехода времени в температуру представлена на рисунке 25.



	Время в часах	Температура (	Яркость (%)	Описание	R	G	B
					С учетом яркости		
7	0	1200	0		0	0	0
8	1	1200	0		0	0	0
9	2	1200	0		0	0	0
10	3	1200	0		0	0	0
11	4	1200	10		25	9	0
12	5	2000	50		255	68	68
13	6	2000	80	Восход	255	137	137
14	7	3500	90		255	192	141
15	8	4250	100		212	212	177
16	9	6500	100		255	254	250
17	10	7000	100		243	242	242
18	11	7000	100		243	242	242
19	12	7500	100	Полдень	230	235	235
20	13	7000	100		243	242	242
21	14	7000	100		243	242	242
22	15	6500	100		255	254	250
23	16	4250	100		255	211	175
24	17	3500	90		230	174	127
25	18	2000	80	Закат	204	110	11
26	19	2000	50		127	68	7
27	20	1200	10		25	9	0
28	21	1200	0		0	0	0
29	22	1200	0		0	0	0
30	23	1200	0		0	0	0
31	0	1200	0		0	0	0

Рисунок 25 – схема перехода от времени к температуре и RGB формату

График функции перехода от времени к RGB представлен на рисунке 26.

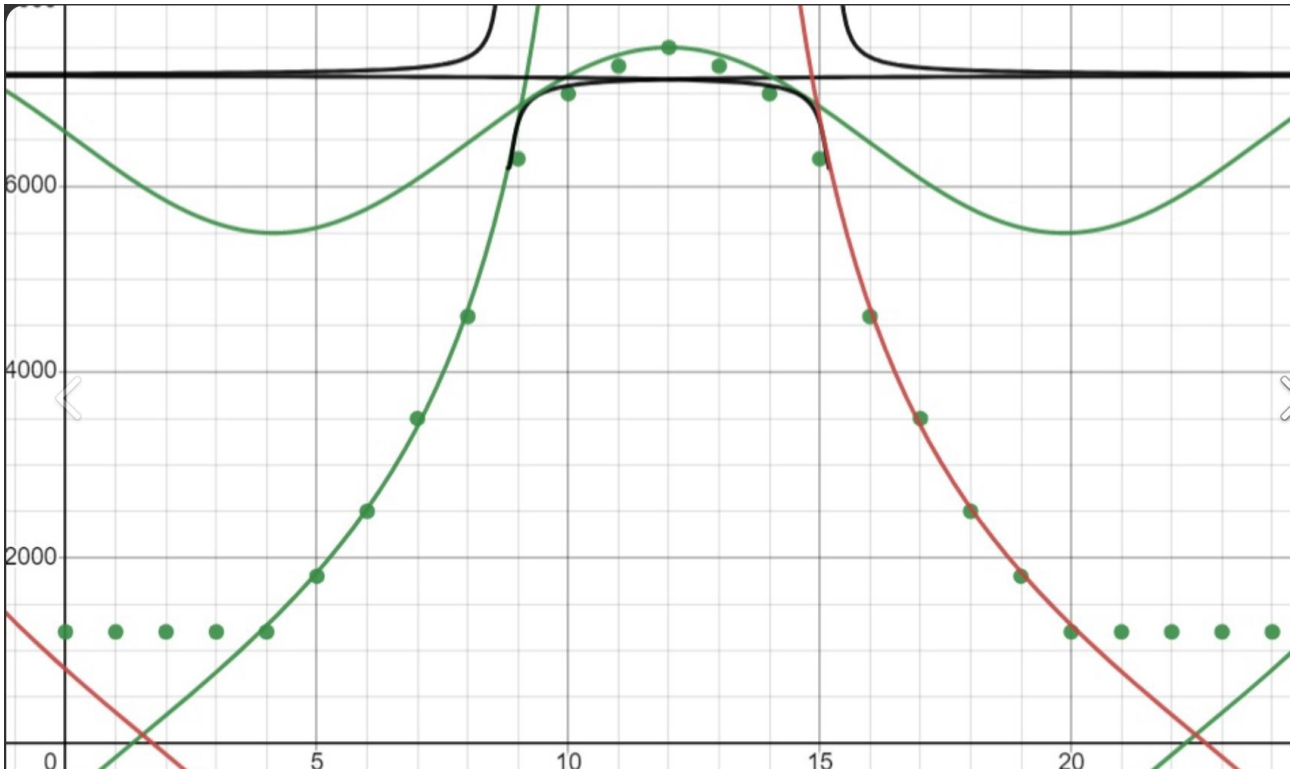


Рисунок 26 – график перехода от времени к температуре

Температура по функции перехода представленной в листинге ниже преобразуется в формат RGB, после чего подается на адресную ленту WS2812.

```
float Red(float temp){
  float reds;
  if (temp <= 66 ) {  reds = 255; }
  else {
    reds = temp - 60 ;
    reds = 329.698727446 * (pow(reds,(-0.1332047592)));
    if (reds < 0){reds = 0;}
    else if (reds > 255){reds = 255;};
  }
  return reds;
}

float Green(float temp){
  float gr=0;
  if (temp <= 66 ) {
    gr = temp ;
    gr = 99.4708025861 *log(gr);
    gr = gr - 161.1195681661;
    if (gr < 0){gr = 0;}
    else if (gr > 255){gr = 255;};
   }
  else {
    gr = temp - 60 ;
    gr = 288.1221695283 * (pow(gr,(-0.0755148492)));
    if (gr < 0){gr = 0;}
    else if (gr > 255){gr = 255;};
```

```
  }
  return gr;

}
float Blue(float temp) {
   float bl=0;
  if (temp >= 66 ) {  bl = 255; }
  else {
    if (temp <=19 ) { bl =0;}
    else {
      bl =  temp - 10;
      bl =  138.5177312231 * log(bl) - 305.0447927307;
      if (bl < 0) {bl = 0;}
      else if(bl > 255){bl = 255;}
    }
  }
  return bl;
}
```

Схема алгоритма обработки данных с RTC изображена на рисунке 27.

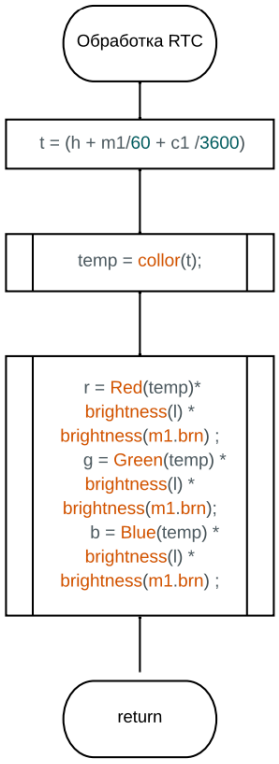


Рисунок 27 – Схема алгоритма обработки RTC

Алгоритм работает следующим образом: время полученное с RTC преобразуется в формат часов, после чего отправляется на функцию «color» где преобразуется в температуру в кельвинах. Далее температура отправляется на функции RGB где преобразуется в цвета ргб.

1.5.2 Алгоритм пульта оператора

Алгоритм состоит из обработки кнопок и формировании меню, выводимого на текстовый дисплей.

Структурная схема меню представлена на рисунке 16. Функционал кнопок описан в таблице 7. Схема алгоритма обработки пульта оператора представлена на рисунке

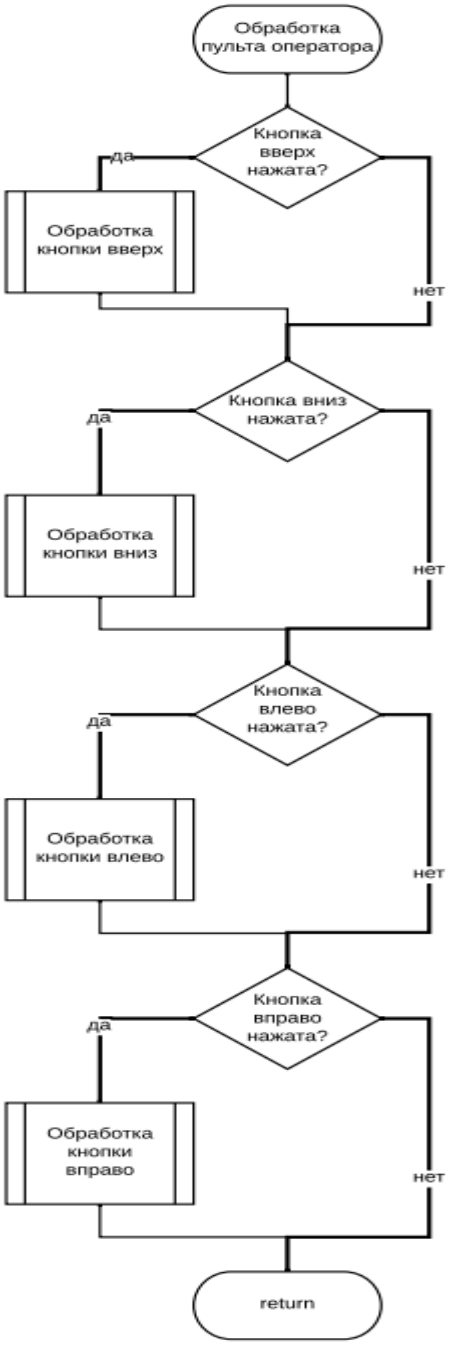


Рисунок 28 – Схема алгоритма обработки пульта оператора

Листинг обработки кнопок представлен ниже.

```
void button(int type){
   switch (type){
     case 0: //up
```

```
          if ((menu[0]==1) and (menu[1]==0)){
                    }
                else {
                    if(menu[1] > 1 and menu[2]==0) {
                        if (menu[1] > 1){
                            menu[1] = menu[1] - 1;
                        } }
                      else if (menu[2] > 1 ) {
                                        menu[2] = menu[2] - 1;
                                    } else if (menu[2] == 1) {
                                        menu[2]= menu[2] + 1;} else {menu[1]=
menu[1] + 3;}
              };
              if ((menu[1] == 4) and (menu[2]=1)) {
                if (brn < 10){brn = brn + 1;} else {brn = 1;}}
       break;
         case 1: //down
         if ((menu[0]==1) and (menu[1]==0)){
                  }
              else {
                  if(menu[2] ==0 ) {
                      if (menu[1] < 4){
                          menu[1] = menu[1] + 1;
                      } else { menu[1]= menu[1] - 3;}
                  }  else if (menu[2] < 2) {
                                      menu[2] = menu[2] + 1;
                                  } else { menu[2]= menu[2] - 1;}
            };
            if ((menu[1]== 4) and (menu[2] == 1)) {
                if (brn > 1){brn = brn - 1;} else {brn = 10;}}
             break;
         case 2: //left
           if (menu[1] ==0){ } else {if (menu[2] == 0){
              menu[1] = 0;

        } else if (menu[3] == 0) {
              menu[2] = 0;

      };}
    break;
         case 3: //right
           if(menu[1] ==0){
              menu[1] = menu[1] + 1;
        } else if (menu[2] == 0){
           menu[2]= menu[2]+1;
        } else if((menu[1] ==1) and (menu[2] == 1)){
           speed_bool = false;
           //режим реального времени
           //выбор действие (изменить что-то) из меню
        } else if((menu[1]==1) and( menu[2] == 2)){
           speed_bool = true;
```

```
            //режим ускоренный x100
        } else if((menu[1]==2) and(menu[2] == 1)){
            mode_lamp_bool = ! mode_lamp_bool;
            //режим без ламп
        } else if((menu[1]==2 ) and (menu[2]== 2)){
            mode_sun_bool = ! mode_sun_bool;
            //режим без солнца
        } else if((menu[1]==4) and (menu[2] == 1)){
            //яркость 100%
            brn = 10;
        } else if((menu[1]== 4) and (menu[2] == 2)){
          //яркость 50%
            brn = 7;
        }
        break;
      }
    };
```

1.5.3 Алгоритм основной программы

Алгоритм представляет из себя настройку периферии и вывод информации на тектовый дисплей. Схема алгоритма представлена на рисунке 29.

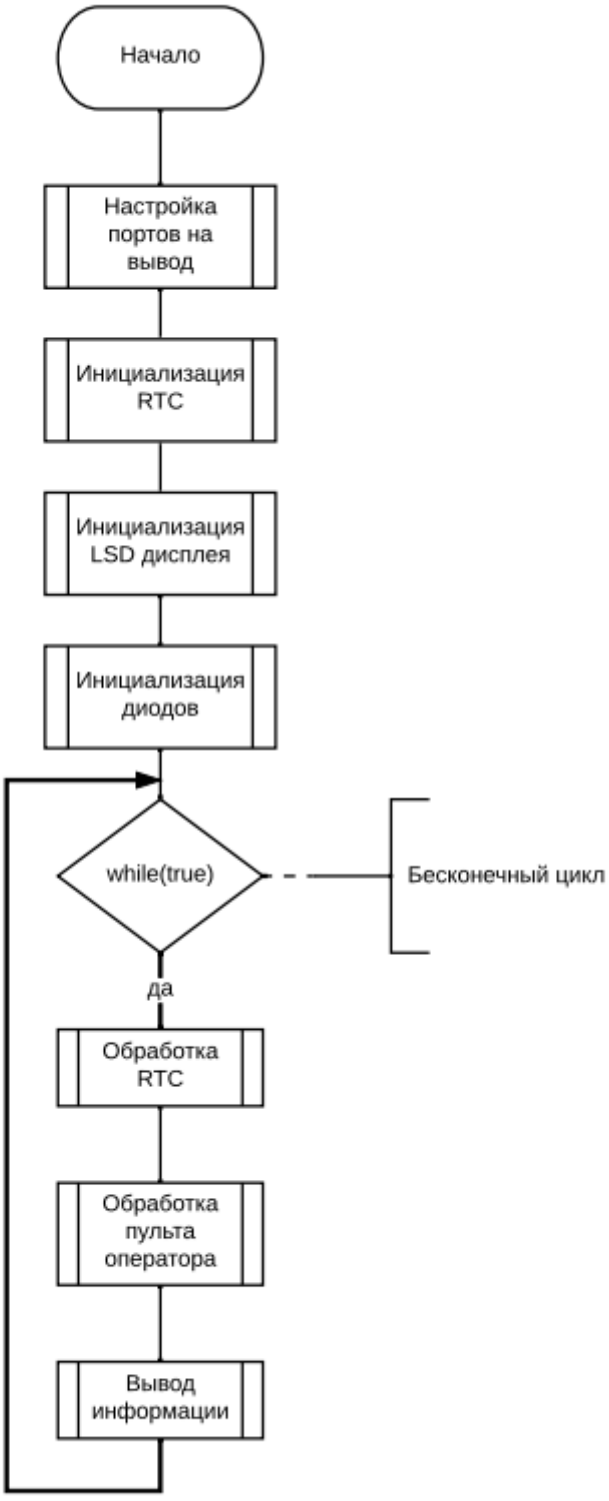


Рисунок 29 – Схема алгоритма основной программы

Суть основной программы в настройке портов на вывод, инициализации дисплея, часов реального времени и блока подсветки городов.

2 Технологическая часть

2.1 Описание систем разработки

Для проектирования и отладки устройства были использованы следующие среды и инструменты:

1. AVR GCC – оптимизирующий компилятор программ, написанных на языках C/C++ для AVR.

2. Proteus 8 Professional – многофункциональная программа для автоматизированного проектирования электронных схем.

2.2 Отладка и тестирование устройства

Разработано, отлажено и протестировано устройство в Proteus. Схема, собранная в программе, изображена на рисунке 30.

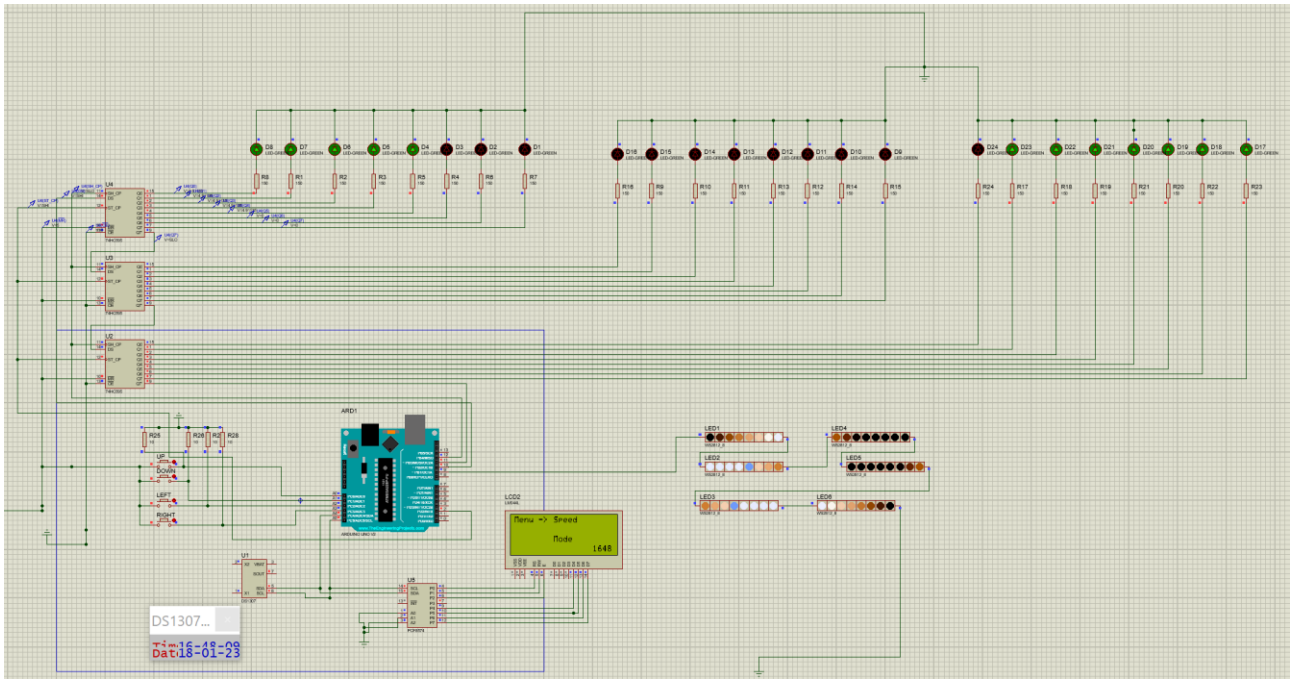


Рисунок 30 – Схема Proteus

Количество задействованной устройством памяти программ представлено на рисунке 31.

```
"C:\\Users\\Mi\\AppData\\Local\\Arduino15\\packages\\arduino\\tools\\avr-gcc\\7.3.0-atmel3.6.1-arduino7/bin/avr-size" -A "C:\\Users\
Скетч использует 17392 байт (53%) памяти устройства. Всего доступно 32256 байт.
Глобальные переменные используют 1068 байт (52%) динамической памяти, оставляя 980 байт для локальных переменных. Максимум: 2048 ба
```

Рисунок 31 – Количество используемой памяти

2.3 Сборка макета устройства

Процесс сборки макета устройства изображен на рисунке 32. Тестирование макета изображено на рисунках 33 и 34.

Рисунок 32 – Процесс сборки макета

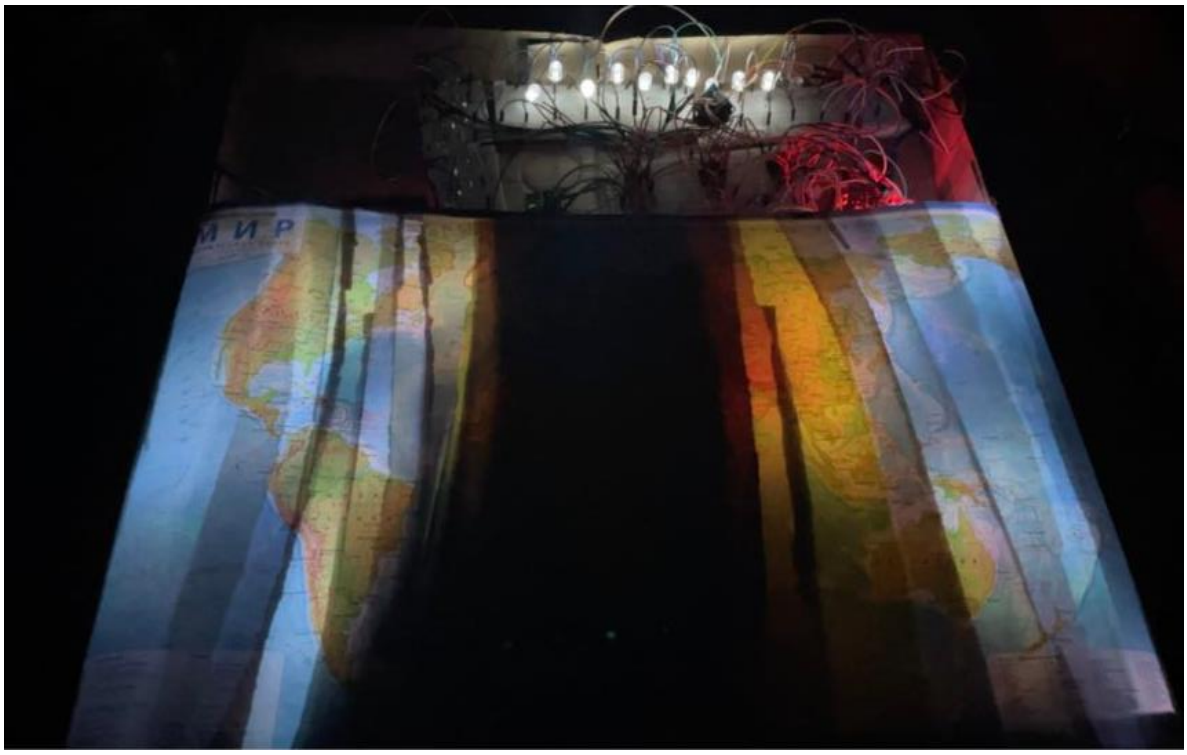


Рисунок 33 – Тестирование макета

Рисунок 34 – Тестирование макета

2.4 Способы программирования памяти МК

Загрузка прошивки в МК производится обычно двумя способами:

– «напрямую» во flash-память микроконтроллера при помощи ISP (In System Programming) внутрисистемного программатора;

– при помощи загрузчика (bootloader), который располагается в конце flash-памяти МК, принимает программный код по протоколу TTL (UART) и записывает его во flash-память.

Загрузчик располагается в самом конце flash-памяти МК и позволяет записывать прошивку, отправляемую через UART. Загрузчик стартует при подаче питания на МК, ждет некоторое время попытки переслать код прошивки по UART, затем передает управление основной программе. И так происходит каждый раз при старте МК.

Он и был использован для программирования данного устройства.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта получены функциональное и принципиальное описание устройства, разработаны алгоритмы и исходные коды программ функционирования устройства на языке С для программирования микроконтроллера.

Устройство представляет из себя систему, состоящую из основного микроконтроллера, подключенного на плату Arduino UNO и различного рода периферии, подключенной к плате. Устройство позволяет выполнить имитацию освещения планеты в таких режимах как: реального времени, ускоренный, без подсветки городов, без подсветки солнцем, измененной яркостью. Устройство обладает пультом оператора, состоящим из текстового дисплея и набора кнопок, для управления устройством.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Arduino UNO3 Datasheet [Электронный ресурс]. – URL: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf (дата обращения 25.10.2022)

2. ATmega328P Datasheet [Электронный ресурс]. – URL: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf (дата обращения 25.10.2022)

3. WS2812 Datasheet [Электронный ресурс]. – URL: https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf (дата обращения 35.10.2022)

4. 74HC595 Datasheet [Электронный ресурс]. – URL: https://www.diodes.com/assets/Datasheets/74HC595.pdf (дата обращения 25.10.2022)

5. DS1307 Datasheet [Электронный ресурс]. – URL: https://iarduino.ru/lib/5835a02639c67ee31e2466d588b01a6b.pdf (дата обращения 30.08.2022)

6. PCF8574 Datasheet [Электронный ресурс]. – URL: https://www.aurel32.net/elec/pcf8574.pdf (дата обращения 30.08.2022)

7. LM016L Datasheet [Электронный ресурс]. – URL: https://www.farnell.com/datasheets/305035.pdf (дата обращения 25.10.2022)

8. Линейные интегральные стабилизаторы напряжения [Электронный ресурс]. – URL: https://power-electronics.info/liner-voltage-regulators.html (дата обращения 30.08.2022)

9. Тепловое проектирование для линейных стабилизаторов напряжения [Электронный ресурс]. – URL: https://radioprog.ru/post/711 (дата обращения 30.08.2022)

10. Хартов В.Я. Микроконтроллеры AVR. Практикум для начинающих: учебное пособие. – 2-е изд., испр. и доп. – М.: Издательство: МГТУ им. Н.Э. Баумана, 2012. – 280с.

11. ГОСТ 7.32–2017. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – [введен в действие Приказом Росстандарта от 24.10.2017 N 1494-ст]

ПРИЛОЖЕНИЕ А

Исходный текст программы

Листинг файла main.ino

```
#include <Adafruit_NeoPixel.h>
#include <iarduino_RTC.h>
iarduino_RTC time(RTC_DS1307);   // создаем обьект rtc = time
#include <LiquidCrystal_I2C.h>
#include <math.h>
#include "params.h"
#include <Wire.h>

const int lamp[24][3]= {
   //таблица переходов освещения городов в зависимости от часового пояса
       {3, 255, 128}, //00000011 11111111 10000000
       {7, 255, 0},   //00000111 11111111 00000000
       {15, 254, 0},  //00001111 11111110 00000000
       {31, 252, 0},  //00011111 11111100 00000000
       {63,248, 0},   //00111111 11111000 00000000
       {127,240,0},   //01111111 11110000 00000000
       {255,224,0},   //11111111 11100000 00000000
       {255,192,1},   //11111111 11000000 00000001
       {255,128,3},   //11111111 10000000 00000011
       {255,0,7},     //11111111 00000000 00000111
       {254,0,15},    //11111110 00000000 00001111
       {252,0,31},    //11111100 00000000 00011111
       {248,0,63},    //11111000 00000000 00111111
       {240,0,127},   //11110000 00000000 01111111
       {224,0,255},   //11100000 00000000 11111111
       {192,1,255},   //11000000 00000001 11111111
       {128,3,255},   //10000000 00000011 11111111
       {0,7,255},     //00000000 00000111 11111111
       {0,15,254},    //00000000 00001111 11111110
       {0,31,252},    //00000000 00011111 11111100
       {0,63,248},    //00000000 00111111 11111000
       {0,127,240},   //00000000 01111111 11110000
       {0,255,224},   //00000000 11111111 11100000
       {1,255,192}    //00000001 11111111 11000000
};
// номер пина отведенный под ws2812
const int NeoPin = 9; //PB1
// количество светодиодов
const int NeoCount = 48;
//Пин подключен к ST_CP входу 74HC595
const int latchPin = 2;
//Пин подключен к SH_CP входу 74HC595
const int clockPin = 12;
//Пин подключен к DS входу 74HC595
const int dataPin = 10;

```

```
47. // constexpr uint8_t PIN_RS = 4;
48. // constexpr uint8_t PIN_EN = 3;
49. // constexpr uint8_t PIN_DB4 = 8;
50. // constexpr uint8_t PIN_DB5 = 7;
51. // constexpr uint8_t PIN_DB6 = 6;
52. // constexpr uint8_t PIN_DB7 = 5;
53.
54.  int r,g,b;
55.  int c = -1;
56.  int brn;
57.  int data=0;
58.  bool bup;
59.  bool bdown;
60.  bool bleft;
61.  bool bright;
62. bool flag_up, flag_right, flag_left, flag_down = false;
63.
64. LiquidCrystal_I2C lcd(0x27,20,4); // 0x20 адрес в протеусе 0x27 на плате
65.
66. //NEO GRB = //последовательность цветов при передаче данных G,R,B
67. // NEO_KHZ800 //Передача данных на частоте 800 кГц (для лент на
    светодиодах WS2812)
68.
69. Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NeoCount, NeoPin, NEO_GRB +
    NEO_KHZ800); //
70. //0x68 адрес rtc в протеусе
71.
72. struct MENUS { // основная структура, в которой сохраняются текущие
    настройки
73.       int brn=10; //яркость %/10;
74.        int menu[4]={1,0,0,0}; // уровни меню.
75.       bool speed_bool= false; // скоростной режим false= RTC; true = смена
    раз в секунду
76.       bool mode_sun_bool = true; // режим работы с подсветкой ws2812
77.       bool mode_lamp_bool = true; // режим без работы регистров (подсветки
    городов)
78.       bool info_bool = false;  //вывод информации
79.       char components[47]= "A.Uno,DS1307,LM044L,WS2812B,74HC595";
80.        // информация об устройстве
81.
82.
83.       void button(int type){ // функция обработки кнопок по уровням
84.         switch (type){
85.           case 0: //up
86.            if ((menu[0]==1) and (menu[1]==0)){
87.                       }
88.                  else {
89.                      if(menu[1] > 1 and menu[2]==0) {
90.                          if (menu[1] > 1){
91.                              menu[1] = menu[1] - 1;
92.                          } }
```

```
93.                         else if (menu[2] > 1 ) {
94.                                             menu[2] = menu[2] - 1;
95.                                         } else if (menu[2] == 1) {
96.                                             menu[2]= menu[2] + 1;} else
   {menu[1]= menu[1] + 3;}
97.                 };
98.                 if ((menu[1] == 4) and (menu[2]=1)) {
99.                   if (brn < 10){brn = brn + 1;} else {brn = 1;}}
100.        break;
101.          case 1: //down
102.          if ((menu[0]==1) and (menu[1]==0)){
103.                    }
104.                else {
105.                    if(menu[2] ==0 ) {
106.                        if (menu[1] < 4){
107.                            menu[1] = menu[1] + 1;
108.                        } else { menu[1]= menu[1] - 3;}
109.                    }  else if (menu[2] < 2) {
110.                                        menu[2] = menu[2] + 1;
111.                                    } else { menu[2]= menu[2] - 1;}
112.             };
113.             if ((menu[1]== 4) and (menu[2] == 1)) {
114.                 if (brn > 1){brn = brn - 1;} else {brn = 10;}}
115.              break;
116.          case 2: //left
117.            if (menu[1] ==0){ } else {if (menu[2] == 0){
118.               menu[1] = 0;
119.
120.         } else if (menu[3] == 0) {
121.               menu[2] = 0;
122.
123.       };}
124.     break;
125.          case 3: //right
126.            if(menu[1] ==0){
127.               menu[1] = menu[1] + 1;
128.          } else if (menu[2] == 0){
129.             menu[2]= menu[2]+1;
130.          } else if((menu[1] ==1) and (menu[2] == 1)){
131.             speed_bool = false;
132.             //режим реального времени
133.             //выбор действие (изменить что-то) из меню
134.          } else if((menu[1]==1) and( menu[2] == 2)){
135.             speed_bool = true;
136.             //режим ускоренный x100
137.          } else if((menu[1]==2) and(menu[2] == 1)){
138.             mode_lamp_bool = ! mode_lamp_bool;
139.             //режим без ламп
140.          } else if((menu[1]==2 ) and (menu[2]== 2)){
141.             mode_sun_bool = ! mode_sun_bool;
142.             //режим без солнца
```

```
143.          } else if((menu[1]==4) and (menu[2] == 1)){
144.             //яркость 100%
145.             brn = 10;
146.          } else if((menu[1]== 4) and (menu[2] == 2)){
147.            //яркость 50%
148.             brn = 7;
149.          }
150.          break;
151.        }
152.      };
153.
154.};
155.
156.MENUS m1;
157.
158.void setup() {
159.  // основная программа
160.  // настройка пинов подсветки городов на вывод;
161.  pinMode(latchPin, OUTPUT);
162.  pinMode(clockPin, OUTPUT);
163.  pinMode(dataPin, OUTPUT);
164.
165.  pixels.begin(); // подготовка для вывода данных
166.  pixels.show(); // Устанавливаем все светодиоды в состояние "Выключено"
167.  const char* strM="JanFebMarAprMayJunJulAugSepOctNovDec";  // Определяем
   массив всех вариантов текстового представления текущего месяца.
168.  const char* sysT=__TIME__;                                  // Получаем
   время компиляции скетча в формате "SS:MM:HH".
169.  const char* sysD=__DATE__;                                  // Получаем
   дату  компиляции скетча в формате "MMM:DD:YYYY", где MMM - текстовое
   представление текущего месяца, например: Jul.
170.  //  Парсим полученные значения sysT и sysD в массив i:
171.   // Определяем массив «i» из 6 элементов типа int, содержащий следующие
   значения: секунды, минуты, часы, день, месяц и год компиляции скетча.
172.const int i[6] {(sysT[6]-48)*10+(sysT[7]-48), (sysT[3]-48)*10+(sysT[4]-
   48), (sysT[0]-48)*10+(sysT[1]-48), (sysD[4]-48)*10+(sysD[5]-48),
   ((int)memmem(strM,36,sysD,3)+3-(int)&strM[0])/3, (sysD[9]-
   48)*10+(sysD[10]-48)};
173.  time.begin();
174.  time.settime(i[0],i[1],i[2],i[3],i[4],i[5]);
175.  //time.settime(0,8,4,22,12,22,4); // Устанавливаем время: 0 сек, 8 мин,
   4 час, 22, октября, 2022 года, четверг
176.  lcd.begin(); //подготовка для вывода данных
177.  //lcd.begin(16,2);
178.  lcd.backlight(); //включаем подсветку дисплея
179.  lcd.setCursor(1,1); //установка курсора на 1.1. при нумерации с 0.0.
180.  lcd.print("Menu");
181.  }
182.//}
183.void menu_type(MENUS m1,int k[4]){
184.  if (k[1]== 0 ){
```

```
185.       lcd.clear();
186.       lcd.print("Menu");
187.     //1000;
188.   }
189.
190.     if(k[1]==1 and k[2]==0){
191.         lcd.clear();
192.       lcd.print("Menu -> Speed");
193.       lcd.print("               Mode");
194.       //1100;
195.     }
196.
197.     if(k[1]==2 and k[2]==0 ) {
198.       lcd.clear();
199.       lcd.print("Menu -> Mode");
200.       lcd.print("                Info");
201.       //1200;
202.       }
203.      if(k[1]==3 and k[2]==0 ){
204.         lcd.clear();
205.         lcd.print("Menu    Mode");
206.         lcd.print("             -> Info");
207.       //1300
208.       }
209.       if(k[1]==4 and k[2]==0){
210.         lcd.clear();
211.         lcd.print("Menu    Info");
212.         lcd.print("             -> Bright");
213.         //1400
214.
215.       }
216.
217.       if(k[1]==1 and k[2]==1){
218.       //1110
219.       lcd.clear();
220.       lcd.print("Speed -> RTC");
221.       lcd.print("                X100");
222.       }
223.       if (k[1] ==1 and k[2]==2){
224.         //1120
225.         lcd.clear();
226.         lcd.print("Speed    RTC");
227.         lcd.print("              -> X100");
228.       }
229.
230.       if (k[1]==2 and k[2]==1){
231.         // 1210
232.         lcd.clear();
233.         lcd.print("Mode -> Lamp");
234.         lcd.print("                Sun");
235.       }
```

```
236.
237.     if (k[1]==2 and k[2]==2){
238.       // 1220
239.       lcd.clear();
240.       lcd.print("Mode    Lamp");
241.       lcd.print("              -> Sun");
242.
243.     }
244.     if (k[1]==3 and k[2]==1){
245.         // 1310
246.         lcd.clear();
247.         lcd.print(m1.components);
248.         //lcd.print();
249.     }
250.
251.     if (k[1]==4 and k[2]==1){
252.         //1410
253.       int buf_down =m1.brn -1;
254.       if (buf_down <2) {buf_down = 10;};
255.       lcd.clear();
256.       lcd.print("Brightness -> ");
257.       lcd.print(m1.brn * 10);
258.       lcd.print("                  ");
259.       lcd.print(buf_down * 10);
260.
261.     }
262.
263.}
264.
265.float times( int h, int m, int c){
266.    float t=0;
267.     float m1 = m;
268.     float  c1 = c;
269.     t = (h + m1/60 + c1 /3600) ;
270.     return t;
271.}
272.void loop() {
273.     Wire.begin(); // старт i2c
274.     Wire.beginTransmission(0x68); // начинаем обмен с DS1307 с i2c
   адресом 0x68
275.     time.gettime(); //получаем данные с RTC через I2C
276.      int h = time.Hours;
277.      int m = time.minutes ;
278.      int s = time.seconds ;
279.      lcd.setCursor(16,3); //выводим текущее время на экран
280.      lcd.print(h);
281.      lcd.print(":");
282.      lcd.setCursor(18,3);
283.      lcd.print(m);
284.      lcd.setCursor(0,0);
285.      h = (h + 10) % 24;
```

```
286.      if (m1.speed_bool)
287.      { h = (s+12) % 24;// ускоренный режим теста
288.        s = m ;
289.       }
290.     float time = times(h,m,s); // приводом время к формату времени часа
291.     for( int i =0 ; i< NeoCount; i++){
292.       float temp;
293.       float l= times((h+i)%24, m, s); // приводим время к формату часа в
   i-том часовом поясе
294.       temp = collor(l); // приводим формат времени к формату температуры
   по формуле перехода
295.
296.       //по формуле перехода из температуры получаем цвет в rgb и умножаем
   его на яркость.
297.       r = Red(temp)* brightness(l) * brightness(m1.brn) ; //по формуле
   перехода из температуры получаем цвет в rgb и умножаем его на яркость.
298.       g = Green(temp) * brightness(l) * brightness(m1.brn);
299.       b = Blue(temp) * brightness(l) * brightness(m1.brn) ;
300.       if (m1.mode_sun_bool){
301.         if (i<24){
302.             pixels.setPixelColor(i, pixels.Color(r,g,b)); // выяснить как
   устроено внутри и по какому интерфейсу работает диодная лента
303.       } else {
304.             pixels.setPixelColor(NeoCount-i+23, pixels.Color(r,g,b)); //
   выяснить как устроено внутри и по какому интерфейсу работает диодная
   лента
305.       }
306.      }else {
307.             pixels.setPixelColor(i, pixels.Color(0,0,0)); // выяснить
   как
308.      }
309. }
310.  pixels.show();
311.  int nesting =0;
312.  // rainbow(30);
313.  bup = digitalRead(A0);//but_up_pin);
314.  bdown = digitalRead(A1);//but_down_pin);
315.  bleft = digitalRead(A2);//but_left_pin);
316.  bright = digitalRead(A3);//but_right_pin);
317.  if (bup && !flag_up) {
318.    flag_up = true;
319.    m1.button(0);
320.    menu_type(m1, m1.menu);
321.  };
322.  if (!bup && flag_up) {
323.    flag_up = false;
324.  };
325.  if (bdown && !flag_down) {
326.    flag_down = true;
327.    m1.button(1);
328.    menu_type(m1, m1.menu);
```

```
329.
330.   };
331.   if (!bdown && flag_down) {
332.     flag_down = false;
333.   };
334.   if (bleft && !flag_left) {
335.     flag_left = true;
336.     m1.button(2);
337.     menu_type(m1, m1.menu);
338.   };
339.   if (!bleft && flag_left) {
340.     flag_left = false;
341.   };
342.   if (bright && !flag_right) {
343.     flag_right = true;
344.
345.     m1.button(3);
346.     menu_type(m1, m1.menu);
347.   };
348.   if (!bright && flag right) {
349.     flag_right = false;
350.   };
351.
352.     digitalWrite(latchPin, LOW);
353.   int d1 = 0;
354.   int d2 = 0;
355.   int d3 = 0;
356.
357.// передаем последовательно на вход данных
358.   if (!m1.mode_lamp_bool){
359.       shiftOut(dataPin, clockPin, MSBFIRST, d1);
360.       shiftOut(dataPin, clockPin, MSBFIRST, d2);
361.       shiftOut(dataPin, clockPin, MSBFIRST, d3);
362.
363.   } else {
364.       shiftOut(dataPin, clockPin, MSBFIRST, lamp[h][1]);
365.       shiftOut(dataPin, clockPin, MSBFIRST, lamp[h][2]);
366.       shiftOut(dataPin, clockPin, MSBFIRST, lamp[h][3]);
367.   }
368.         //"защелкиваем" регистр, устанавливаем значения на выходах
369.       digitalWrite(latchPin, HIGH);
370.}
371.
372.float brightness(float temp){ //переделать
373.   float f=0;
374.   f = 0 - pow(((0.2 *temp) -2.4),4) + 10;
375.   if (f<0){ f =0;};
376.   f = f / 10;
377.   return f;
378.}
379.
```

```
380.float collor(float data){
381.    float f=0;
382.     if (data < 8.9) {
383.       f= 30*tan(0.15*data -0.2);
384.     } else if (data > 15.01) {
385.       f= 30*tan(-0.15*data +0.2);
386.     } else if (data <= 9.4) {
387.       f = 10*(pow(M_E,((-1)/(7*(data - 8.8)))+6.2));
388.     } else if (data >=14.7){
389.       f = 10*(pow(M_E, ((1)/(7*(data-15.2)))+6.2));
390.     } else {f = 10*(sin(0.4*data-3.23)+6.5);}
391.         return f;
392.}
393.
394.float Red(float temp){
395.  float reds;
396.  if (temp <= 66 ) {  reds = 255; }
397.  else {
398.    reds = temp - 60 ;
399.    reds = 329.698727446 * (pow(reds,(-0.1332047592)));
400.    if (reds < 0){reds = 0;}
401.    else if (reds > 255){reds = 255;};
402.  }
403.  return reds;
404.}
405.
406.float Green(float temp){
407.  float gr=0;
408.  if (temp <= 66 ) {
409.    gr = temp ;
410.    gr = 99.4708025861 *log(gr);
411.    gr = gr - 161.1195681661;
412.    if (gr < 0){gr = 0;}
413.    else if (gr > 255){gr = 255;};
414.   }
415.  else {
416.    gr = temp - 60 ;
417.    gr = 288.1221695283 * (pow(gr,(-0.0755148492)));
418.    if (gr < 0){gr = 0;}
419.    else if (gr > 255){gr = 255;};
420.  }
421.  return gr;
422.
423.}
424.float Blue(float temp) {
425.   float bl=0;
426.  if (temp >= 66 ) {  bl = 255; }
427.  else {
428.    if (temp <=19 ) { bl =0;}
429.    else {
430.      bl =  temp - 10;
```

```
431.       bl =  138.5177312231 * log(bl) - 305.0447927307;
432.       if (bl < 0) {bl = 0;}
433.       else if(bl > 255){bl = 255;}
434.     }
435.   }
436.   return bl;
437. }
438.
439.
```

Листинг файла используемой библиотеки iarduino_RTC.h

```
1. //  Библиотека для работы с часами реального времени: (на чипе DS1302)
   https://iarduino.ru/shop/Expansion-payments/rtc-modul-ds1302.html
2. //                                                    (на чипе DS1307)
   https://iarduino.ru/shop/Expansion-payments/kroshechnye-rtc-modul-
   realnogo-vremeni.html
3. //                                                                htt
   ps://iarduino.ru/shop/Expansion-payments/chasy-realnogo-vremeni-rtc-trema-
   modul.html
4. //                                                    (на чипе DS3231)
   https://iarduino.ru/shop/Expansion-payments/chasy-realnogo-vremeni-
   ds3231.html
5. //                                                                htt
   ps://iarduino.ru/shop/Expansion-payments/chasy-realnogo-vremeni-rtc-trema-
   modul-v2-0.html
6. //                                                    (на чипе RX8025)
7. //  Версия: 2.0.0
8. //  Последнюю версию библиотеки Вы можете скачать по ссылке:
   https://iarduino.ru/file/235.html
9. //  Подробное описание функции бибилиотеки доступно по ссылке:
   https://wiki.iarduino.ru/page/chasy-realnogo-vremeni-rtc-trema-modul/
10. //  Библиотека является собственностью интернет магазина iarduino.ru и
   может свободно использоваться и распространяться!
11. //  При публикации устройств или скетчей с использованием данной
   библиотеки, как целиком, так и её частей,
12. //  в том числе и в некоммерческих целях, просим Вас опубликовать ссылку:
   https://iarduino.ru
13. //  Автор библиотеки: Панькин Павел
14. //  Если у Вас возникли технические вопросы, напишите нам:
   shop@iarduino.ru
15.
16. #ifndef iarduino_RTC_h                                              //
17. #define iarduino_RTC_h                                              //
18.                                                          //
19. #define RTC_UNDEFINED
   0                                          //  Модуль часов реального
   времени не определён
20.                                                          //
21. #if defined(ARDUINO) && (ARDUINO >=
   100)                                    //
22. #include <Arduino.h>                                          //
```

```
23. #else                                                          //
24. #include <WProgram.h>                                                 //
25. #endif                                                           //
26.                                                                //
27. #include
    "memorysaver.h"                                                //  Подключаем
    файл «хранитель памяти»                    (внутри файла есть комментарий
    поясняющий как сэкономить мапять)
28.                                                                //
29. class
    iarduino_RTC_BASE{                                              //  Объявляем
    полиморфный класс
30.   public:                                                      //
31.     virtual
    void  begin       (void);                                  //  Объявляем
    функцию инициализации модуля                (без параметров)
32.     virtual uint8_t funcReadTimeIndex
    (uint8_t);                              //  Объявляем функцию чтения 1
    значения из регистров даты и времени   (0-секунды / 1-минуты / 2-часы / 3-
    день / 4-месяц / 5-год / 6-день недели)
33.     virtual void  funcWriteTimeIndex  (uint8_t,
    uint8_t);                         //  Объявляем функцию записи 1 значения
    в  регистры  даты и времени   (0-секунды / 1-минуты / 2-часы / 3-день / 4-
    месяц / 5-год / 6-день недели, значение)
34. };                                                             //
35.                                                                //
36. #include
    "iarduino_RTC_DS1302.h"                                         //  Подключае
    м файл iarduino_RTC_DS1302.h
37. #include
    "iarduino_RTC_DS1307.h"                                         //  Подключае
    м файл iarduino_RTC_DS1307.h
38. #include
    "iarduino_RTC_DS3231.h"                                         //  Подключае
    м файл iarduino_RTC_DS3231.h
39. #include
    "iarduino_RTC_RX8025.h"                                         //  Подключае
    м файл iarduino_RTC_RX8025.h
40.                                                                //
41. class iarduino_RTC{                                                 //
42.   public:                                                      //
43.   /** Конструктор класса **/                                          //
44.     iarduino_RTC(uint8_t i, uint8_t j=SS, uint8_t k=SCK, uint8_t
    n=MOSI){               //  Конструктор основного
    класса                    (название [, вывод SS/RST [, вывод SCK/CLK [,
    вывод MOSI/DAT]]])
45.       switch(i){                                               //  Тип
    выбранного модуля
46.         #ifdef RTC_ENABLE_DS1302                                    //
47.         case RTC_DS1302: objClass = new iarduino_RTC_DS1302(j,k,n);
    break;              //  Если используется модуль на базе чипа DS1302, то
```

```
   присваеиваем указателю objClass ссылку на новый объект производного класса
   iarduino_RTC_DS1302 переопределяя на него все виртуальные функции
   полиморфного класса iarduino_RTC_BASE
48.         #endif                                                  //
49.         #ifdef RTC_ENABLE_DS1307                                //
50.         case RTC_DS1307: objClass = new iarduino_RTC_DS1307;
   break;                  //  Если используется модуль на базе чипа DS1307,
   то присваеиваем указателю objClass ссылку на новый объект производного
   класса iarduino_RTC_DS1307 переопределяя на него все виртуальные функции
   полиморфного класса iarduino_RTC_BASE
51.         #endif                                                  //
52.         #ifdef RTC_ENABLE_DS3231                                //
53.         case RTC_DS3231: objClass = new iarduino_RTC_DS3231;
   break;                  //  Если используется модуль на базе чипа DS3231,
   то присваеиваем указателю objClass ссылку на новый объект производного
   класса iarduino_RTC_DS3231 переопределяя на него все виртуальные функции
   полиморфного класса iarduino_RTC_BASE
54.         #endif                                                  //
55.         #ifdef RTC_ENABLE_RX8025                                //
56.         case RTC_RX8025: objClass = new iarduino_RTC_RX8025;
   break;                  //  Если используется модуль на базе чипа RX8025,
   то присваеиваем указателю objClass ссылку на новый объект производного
   класса iarduino_RTC_RX8025 переопределяя на него все виртуальные функции
   полиморфного класса iarduino_RTC_BASE
57.         #endif                                                  //
58.       }                                                         //
59.     }                                                           //
60.   /** Пользовательские функции **/                                    //
61.     void  begin   (void)          {objClass -> begin();
   gettime();}           //  Определяем функцию инициализации
   модуля               (без параметров)
62.     void  period    (uint8_t i)       {valPeriod=i;
   valPeriod*=60000;}            //  Определяем функцию задания минимального
   периода обращения к модулю  (i = период в минутах)
63.     void  blinktime (uint8_t i, float j=1)  {valBlink=i;
   valFrequency=uint32_t(1000/j);}      //  Определяем функцию позволяющую
   мигать одним из параметров времени (i = 0-нет / 1-сек / 2-мин / 3-час / 4-
   день / 5-мес / 6-год / 7-день_недели / 8-полдень , j = частота мигания в
   Гц)
64.     void  gettime   (void)          {gettime("");}                  /
   /  Определяем функцию получения даты и времени из переменных     (без
   параметров)
65.     char*
   gettime   (String);                                  //  Объявляем  функц
   ию «дублёр» получения даты и времени из переменных  (строка с параметрами)
66.     char* gettime   (const
   char*);                                 //  Объявляем  функцию получения
   даты и времени ввиде строки      (строка с параметрами)
67.     void  settime   (int, int=-1, int=-1, int=-1, int=-1, int=-1, int=-
   1);             //  Объявляем  функцию установки даты и
   времени             (сек, мин, час, день, мес, год, день_недели)
```

```
68.      uint32_t gettimeUnix (void)          {gettime(""); return
   Unix;}               //  Определяем функцию получения секунд прошедших с
   начала эпохи Unix (без параметров)
69.      void  settimeUnix
   (uint32_t);                                    //  Объявляем  функцию
   установки секунд прошедших с начала эпохи Unix (сек)
70.                                                                  //
71.    /** Переменные доступные для пользователя
   **/                                 //
72.      uint8_t seconds         =
   0;                                    //  Секунды     0-59
73.      uint8_t minutes         =
   0;                                    //  Минуты      0-59
74.      uint8_t hours           =
   1;                                    //  Часы      1-12
75.      uint8_t Hours           =
   0;                                    //  Часы      0-23
76.      uint8_t midday          =
   0;                                    //  Полдень     0-
   1                           (0-am, 1-pm)
77.      uint8_t day           = 1;                                  //  День
   месяца   1-31
78.      uint8 t weekday         = 0;                                   //  День
   недели   0-6                          (0-воскресенье, 1-понедельник, ... ,
   6-суббота)
79.      uint8_t month           =
   1;                                    //  Месяц     1-12
80.      uint8_t year          =
   0;                                    //  Год       0-
   99                          (без учёта века)
81.       uint32_t Unix          =
   0;                                    //  Секунды     прошедшие с начала эпохи
   Unix           (01.01.1970 00:00:00 GMT)
82.                                                                  //
83.    /** Внутренние переменные **/                                      //
84.      iarduino_RTC_BASE*  objClass;                                    //  О
   бъявляем указатель на объект полиморфного класса         (функции данного
   класса будут переопределены, т.к. указателю будет присвоена ссылка на
   производный класс)
85.      char* charReturn        = (char*)
   malloc(1);                            //  Определяем указатель на символьную
   область памяти в 1 байт      (указатель будет ссылаться на строку вывода
   времени)
86.    const char*   charInput       =
   "waAdhHimsyMDY";                          //  Определяем константу-строку с
   символами требующими замены     (данные символы заменяются функцией
   gettime на значение времени)
87.    const char* charMidday        =
   "ampmAMPM";                               //  Определяем константу-строку для
   вывода полудня            (am / pm / AM / PM)
```

```
88.    const char* charDayMon         =
   "SunMonTueWedThuFriSatJanFebMarAprMayJunJulAugSepOctNovDec";  //  Определя
   ем константу-строку для вывода дня недели или месяца     (Mon ... Sun / Jan
   ... Dec)
89.     uint8_t
   arrCalculationTime[7];                                          //  Объявляем
   массив для рассчёта времени без обращения к модулю     (для хранения
   последних, прочитанных из модуля, значений даты и времени)
90.     uint8_t valBlink        =
   0;                                  //  Определяем параметр времени, который
   должен мигать          (1-сек / 2-мин / 3-час / 4-день / 5-мес / 6-год /
   7-день_недели / 8-полдень)
91.      uint32_t valFrequency      =
   1000;                               //  Определяем частоту мигания параметра
   времени для функции blinktime  (по умолчанию 1 Гц)
92.     uint8_t valCentury        =
   21;                                 //  Определяем переменную для хранения
   текущего века          (по умолчанию 21 век)
93.      uint16_t valPeriod       =
   0;                                  //  Определяем минимальный период опроса
   модуля             (в минутах, от 00 до 255)
94.      uint32_t valRequest        =
   0;                                  //  Определяем время последнего чтения
   регистров времени
95.   private:                                                  //
96.   /** Внутренние функции **/                                          //
97.     void  funcReadTime      (void);                                 //  Объ
   являем функцию чтения даты и времени из регистров модуля     (без
   параметров)
98.     void  funcWriteTime     (int, int, int, int, int, int,
   int);                //  Объявляем функцию записи даты и времени
   в  регистры  модуля     (без параметров)
99.     uint8_t funcConvertCodeToNum  (uint8_t i) {return (i >> 4)*10 + (i &
   0x0F);}            //  Определяем функцию преобразования двоично-
   десятичного кода в число  (код)
100.    uint8_t funcConvertNumToCode  (uint8_t i) {return ((i/10) << 4) +
   (i%10);}            //  Определяем функцию преобразования числа в двоично-
   десятичный код  (число)
101.    void  funcSetMoreTime     (void){hours=(Hours%12)?(Hours%12):12;
   midday=(Hours<12)?0:1;}    //  Корректировка переменных не читаемых из
   модуля            (без параметров)
102.    void  funcCalculationTime   (void);                                //
    Объявляем функцию расчёта времени без обращения к модулю      (без
   параметров)
103.     uint32_t
   funcCalculationUnix   (void);                                  //  Объявляем
   функцию расчёта секунд прошедших с начала эпохи Unix    (без параметров)
104.    void  funcFillChar      (uint8_t, uint8_t, uint8_t,
   uint8_t);              //  Объявляем функцию заполнения строки вывода
   времени          (данные, тип данных, позиция для вставки, мигание)
105.};                                                          //
```

```
106.                                                                    //
107.#endif                                                                  //
108.
109.#endif
```

Листинг файла используемой библиотеки LiquidCrystal_I2C.h

```
1. #ifndef FDB_LIQUID_CRYSTAL_I2C_H
2. #define FDB LIQUID CRYSTAL I2C H
3.
4. #include <inttypes.h>
5. #include <Print.h>
6.
7. // commands
8. #define LCD_CLEARDISPLAY 0x01
9. #define LCD_RETURNHOME 0x02
10. #define LCD_ENTRYMODESET 0x04
11. #define LCD_DISPLAYCONTROL 0x08
12. #define LCD_CURSORSHIFT 0x10
13. #define LCD_FUNCTIONSET 0x20
14. #define LCD_SETCGRAMADDR 0x40
15. #define LCD_SETDDRAMADDR 0x80
16.
17. // flags for display entry mode
18. #define LCD_ENTRYRIGHT 0x00
19. #define LCD_ENTRYLEFT 0x02
20. #define LCD_ENTRYSHIFTINCREMENT 0x01
21. #define LCD_ENTRYSHIFTDECREMENT 0x00
22.
23. // flags for display on/off control
24. #define LCD_DISPLAYON 0x04
25. #define LCD_DISPLAYOFF 0x00
26. #define LCD_CURSORON 0x02
27. #define LCD_CURSOROFF 0x00
28. #define LCD_BLINKON 0x01
29. #define LCD_BLINKOFF 0x00
30.
31. // flags for display/cursor shift
32. #define LCD_DISPLAYMOVE 0x08
33. #define LCD_CURSORMOVE 0x00
34. #define LCD_MOVERIGHT 0x04
35. #define LCD_MOVELEFT 0x00
36.
37. // flags for function set
38. #define LCD_8BITMODE 0x10
39. #define LCD_4BITMODE 0x00
40. #define LCD_2LINE 0x08
41. #define LCD_1LINE 0x00
42. #define LCD_5x10DOTS 0x04
43. #define LCD_5x8DOTS 0x00
44.
```

```
45. // flags for backlight control
46. #define LCD_BACKLIGHT 0x08
47. #define LCD_NOBACKLIGHT 0x00
48.
49. #define En B00000100  // Enable bit
50. #define Rw B00000010  // Read/Write bit
51. #define Rs B00000001  // Register select bit
52.
53. /**
54.  * This is the driver for the Liquid Crystal LCD displays that use the
    I2C bus.
55.  *
56.  * After creating an instance of this class, first call begin() before
    anything else.
57.  * The backlight is on by default, since that is the most likely
    operating mode in
58.  * most cases.
59.  */
60. class LiquidCrystal_I2C : public Print {
61. public:
62.   /**
63.    * Constructor
64.    *
65.    * @param lcd_addr  I2C slave address of the LCD display. Most likely
    printed on the
66.    *          LCD circuit board, or look in the supplied LCD
    documentation.
67.    * @param lcd_cols  Number of columns your LCD display has.
68.    * @param lcd_rows  Number of rows your LCD display has.
69.    * @param charsize  The size in dots that the display has, use
    LCD_5x10DOTS or LCD_5x8DOTS.
70.    */
71.   LiquidCrystal_I2C(uint8_t lcd_addr, uint8_t lcd_cols, uint8_t lcd_rows,
    uint8_t charsize = LCD_5x8DOTS);
72.
73.   /**
74.    * Set the LCD display in the correct begin state, must be called
    before anything else is done.
75.    */
76.   void begin();
77.
78.    /**
79.     * Remove all the characters currently shown. Next print/write
    operation will start
80.     * from the first position on LCD display.
81.     */
82.   void clear();
83.
84.   /**
85.    * Next print/write operation will will start from the first position
    on the LCD display.
```

```
86.     */
87.    void home();
88.
89.     /**
90.      * Do not show any characters on the LCD display. Backlight state will
   remain unchanged.
91.      * Also all characters written on the display will return, when the
   display in enabled again.
92.      */
93.    void noDisplay();
94.
95.    /**
96.     * Show the characters on the LCD display, this is the normal
   behaviour. This method should
97.     * only be used after noDisplay() has been used.
98.     */
99.    void display();
100.
101.   /**
102.    * Do not blink the cursor indicator.
103.    */
104.   void noBlink();
105.
106.   /**
107.    * Start blinking the cursor indicator.
108.    */
109.   void blink();
110.
111.   /**
112.    * Do not show a cursor indicator.
113.    */
114.   void noCursor();
115.
116.   /**
117.    * Show a cursor indicator, cursor can blink on not blink. Use the
118.    * methods blink() and noBlink() for changing cursor blink.
119.    */
120.   void cursor();
121.
122.   void scrollDisplayLeft();
123.   void scrollDisplayRight();
124.   void printLeft();
125.   void printRight();
126.   void leftToRight();
127.   void rightToLeft();
128.   void shiftIncrement();
129.   void shiftDecrement();
130.   void noBacklight();
131.   void backlight();
132.   bool getBacklight();
133.   void autoscroll();
```

```
134.  void noAutoscroll();
135.  void createChar(uint8_t, uint8_t[]);
136.  void setCursor(uint8_t, uint8_t);
137.  virtual size_t write(uint8_t);
138.  void command(uint8_t);
139.
140.  inline void blink_on() { blink(); }
141.  inline void blink_off() { noBlink(); }
142.  inline void cursor_on() { cursor(); }
143.  inline void cursor_off() { noCursor(); }
144.
145.// Compatibility API function aliases
146.  void setBacklight(uint8_t new_val);       // alias for backlight() and
   nobacklight()
147.  void load_custom_character(uint8_t char_num, uint8_t *rows);  // alias
   for createChar()
148.  void printstr(const char[]);
149.
150.private:
151.  void send(uint8_t, uint8_t);
152.  void write4bits(uint8_t);
153.  void expanderWrite(uint8_t);
154.  void pulseEnable(uint8_t);
155.  uint8_t _addr;
156.  uint8_t _displayfunction;
157.  uint8_t _displaycontrol;
158.  uint8_t _displaymode;
159.  uint8_t _cols;
160.  uint8_t _rows;
161.  uint8_t _charsize;
162.  uint8_t _backlightval;
163.};
164.
165.#endif // FDB_LIQUID_CRYSTAL_I2C_H
```

Листинг файла используемой библиотеки Wire.h

```
1. /*
2.   TwoWire.h - TWI/I2C library for Arduino & Wiring
3.   Copyright (c) 2006 Nicholas Zambetti.  All right reserved.
4.
5.   This library is free software; you can redistribute it and/or
6.   modify it under the terms of the GNU Lesser General Public
7.   License as published by the Free Software Foundation; either
8.   version 2.1 of the License, or (at your option) any later version.
9.
10.   This library is distributed in the hope that it will be useful,
11.   but WITHOUT ANY WARRANTY; without even the implied warranty of
12.   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
13.   Lesser General Public License for more details.
14.
```

```
15.   You should have received a copy of the GNU Lesser General Public
16.   License along with this library; if not, write to the Free Software
17.   Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-
   1301  USA
18.
19.   Modified 2012 by Todd Krein (todd@krein.org) to implement repeated
   starts
20.   Modified 2020 by Greyson Christoforo (grey@christoforo.net) to
   implement timeouts
21. */
22.
23. #ifndef TwoWire_h
24. #define TwoWire_h
25.
26. #include <inttypes.h>
27. #include "Stream.h"
28.
29. #define BUFFER_LENGTH 32
30.
31. // WIRE HAS END means Wire has end()
32. #define WIRE_HAS_END 1
33.
34. class TwoWire : public Stream
35. {
36.   private:
37.     static uint8_t rxBuffer[];
38.     static uint8_t rxBufferIndex;
39.     static uint8_t rxBufferLength;
40.
41.     static uint8_t txAddress;
42.     static uint8_t txBuffer[];
43.     static uint8_t txBufferIndex;
44.     static uint8_t txBufferLength;
45.
46.     static uint8_t transmitting;
47.     static void (*user_onRequest)(void);
48.     static void (*user_onReceive)(int);
49.     static void onRequestService(void);
50.     static void onReceiveService(uint8_t*, int);
51.   public:
52.     TwoWire();
53.     void begin();
54.     void begin(uint8_t);
55.     void begin(int);
56.     void end();
57.     void setClock(uint32_t);
58.     void setWireTimeout(uint32_t timeout = 25000, bool reset_with_timeout
   = false);
59.     bool getWireTimeoutFlag(void);
60.     void clearWireTimeoutFlag(void);
61.     void beginTransmission(uint8_t);
```

```
62.     void beginTransmission(int);
63.     uint8_t endTransmission(void);
64.     uint8_t endTransmission(uint8_t);
65.     uint8_t requestFrom(uint8_t, uint8_t);
66.     uint8_t requestFrom(uint8_t, uint8_t, uint8_t);
67.     uint8_t requestFrom(uint8_t, uint8_t, uint32_t, uint8_t, uint8_t);
68.     uint8_t requestFrom(int, int);
69.     uint8_t requestFrom(int, int, int);
70.     virtual size_t write(uint8_t);
71.     virtual size_t write(const uint8_t *, size_t);
72.     virtual int available(void);
73.     virtual int read(void);
74.     virtual int peek(void);
75.     virtual void flush(void);
76.     void onReceive( void (*)(int) );
77.     void onRequest( void (*)(void) );
78.
79.     inline size_t write(unsigned long n) { return write((uint8_t)n); }
80.     inline size_t write(long n) { return write((uint8_t)n); }
81.     inline size t write(unsigned int n) { return write((uint8 t)n); }
82.     inline size_t write(int n) { return write((uint8_t)n); }
83.     using Print::write;
84. };
85.
86. extern TwoWire Wire;
87.
88. #endif
```

ПРИЛОЖЕНИЕ Б

Схема электрическая функциональная

Листов 1

ПРИЛОЖЕНИЕ В

Схема электрическая принципиальная

Листов 1

ПРИЛОЖЕНИЕ Г

Перечень элементов

Листов 2