|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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***

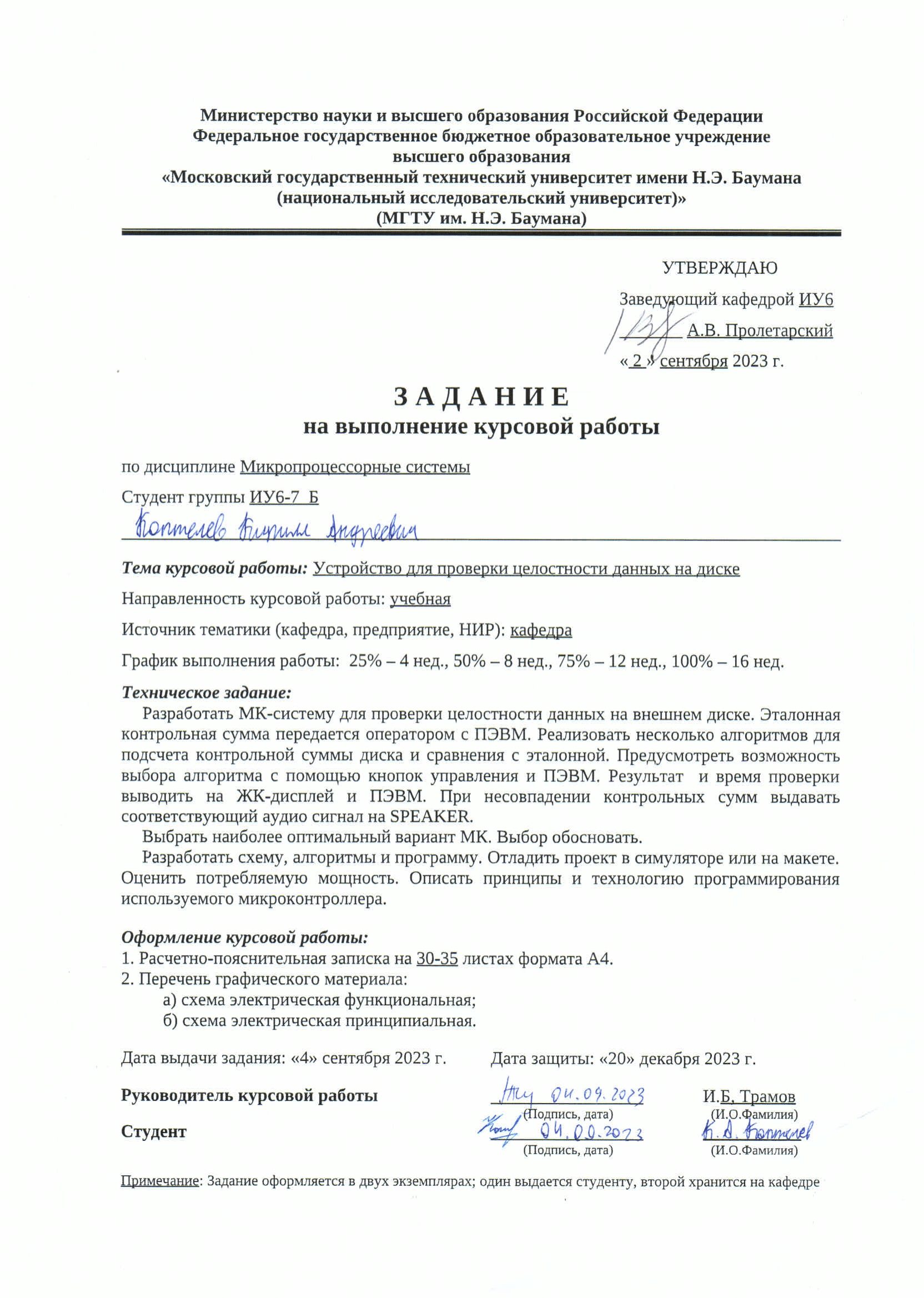
***по дисциплине «Микропроцессорные системы»***

***на тему:***

**Устройство для проверки целостности данных на диске**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент |  |  |  | К.А. Коптелев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Руководитель |  |  |  | И.Б. Трамов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |

*2023 г.*



**РЕФЕРАТ**

РПЗ 48 страниц, 21 рисунок, 8 таблиц, 13 источников, 3 приложения.

МИКРОКОНТРОЛЛЕР, СИСТЕМА, КОНТРОЛЬНАЯ СУММА, ВНЕШНИЙ ДИСК

Объектом разработки является устройство для проверки целостности данных на внешнем диске

Цель работы – создание функционального устройства ограниченной сложности, модель устройства и разработка необходимой документации на объект разработки.

Поставленная цель достигается посредством использования Proteus 8.

В процессе работы над курсовым проектом решаются следующие задачи: выбор МК и драйвера обмена данных, создание функциональной и принципиальной схем системы, расчет потребляемой мощности устройства, разработка алгоритма управления и соответствующей программы МК, а также написание сопутствующей документации.

Оглавление

[ВВЕДЕНИЕ 7](#_Toc153059746)

[1 Конструкторская часть 8](#_Toc153059747)

[1.1 Анализ требований и принцип работы системы 8](#_Toc153059748)

[1.2 Проектирование функциональной схемы 10](#_Toc153059749)

[1.2.1 Микроконтроллер STM32F103C8T6 10](#_Toc153059750)

[1.2.1.1 Используемые элементы 17](#_Toc153059751)

[1.2.1.2 Распределение портов 18](#_Toc153059752)

[1.2.1.3 Организация памяти 19](#_Toc153059753)

[1.2.2 Прием данных от ПЭВМ 20](#_Toc153059754)

[1.2.3 Настройка USART для взаимодействия с ПЭВМ 22](#_Toc153059755)

[1.2.4 LCD-дисплей ST7735 30](#_Toc153059756)

[1.2.5 Настройка SPI для взаимодействия с LCD-дисплеем 30](#_Toc153059757)

[1.2.6 Настройка SPI для взаимодействия с сокетом SD-карты. 31](#_Toc153059758)

[1.2.7 Генератор тактовых импульсов и сброс 31](#_Toc153059759)

[1.2.8 Использование таймера для генерации звукового сигнала 31](#_Toc153059760)

[1.2.9 Построение функциональной схемы 36](#_Toc153059761)

[1.3 Проектирование принципиальной схемы 36](#_Toc153059762)

[1.3.1 Разъем программатора 36](#_Toc153059763)

[1.3.2 Подключение цепи питания 37](#_Toc153059764)

[1.3.3 Расчет сопротивления резисторов 37](#_Toc153059765)

[1.4 Алгоритмы работы системы 41](#_Toc153059766)

[1.4.1 Функция Main и подпрограмма переключение светофоров 41](#_Toc153059767)

[2 Технологическая часть 48](#_Toc153059768)

[2.1 Отладка и тестирование программы 48](#_Toc153059769)

[2.2 Настройка таймера и работа фаз светофора 49](#_Toc153059770)

[2.3 Симуляция работы системы 50](#_Toc153059771)

[2.4 Способы программирования МК 53](#_Toc153059772)

[ЗАКЛЮЧЕНИЕ 56](#_Toc153059773)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 57](#_Toc153059774)

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

МК – микроконтроллер.

ТЗ – техническое задание.

Proteus 8 — пакет программ для автоматизированного проектирования (САПР) электронных схем.

MD5, CRC – алгоритмы для вычисления контрольной суммы.

UART – Universal asynchronous receiver/transmitter – последовательный универсальный синхронный/асинхронный приемопередатчик.

SPI – Serial Peripheral Interface – интерфейс для связи МК с другими внешними устройствами.

LED-дисплей – устройство отображения и передачи визуальной информации, в котором каждой точкой — пикселем — является один или несколько полупроводниковых светодиодов

# ВВЕДЕНИЕ

В данной работе производится разработка устройства для проверки целостности данных на диске.

В процессе выполнения работы проведён анализ технического задания,

создана концепция устройства, разработаны электрические схемы, построен алгоритм и управляющая программа для МК, выполнено интерактивное моделирование устройства.

Система состоит из МК, четырех кнопок для управления процессом проверки целостности, кнопки RESET, LED-дисплея для отображения информации о проверке целостности и подсказок пользователю, разъема для SD-карты, которая выполняет роль внешнего диска, динамика для вывода сигнала о несовпадении контрольных сумм, виртуального терминала для симуляции ввода/вывода с ПЭВМ. Также имеется возможность загружать специально созданный образ SD-карты в SD-карту.

Актуальность разрабатываемого устройства для проверки целостности данных на диске заключается в том, что в современном информационном мире, где цифровые данные играют ключевую роль, обеспечение их целостности является критическим аспектом. Эта система предоставляет пользователям возможность быстрой и надежной проверки целостности данных несколькими алгоритмами, что является важным шагом в обеспечении безопасности и надежности хранимой информации.

# 1 Конструкторская часть

## 1.1 Анализ требований и принцип работы системы

Исходя из требований, изложенных в техническом задании, можно сделать вывод, что задачей работы устройства является чтение внешнего диска, в данном случае SD-карты, чтение эталонной контрольной суммы с ПЭВМ, вычисление контрольной суммы внешнего диска выбранным алгоритмом и сравнение с эталонной, а также вывод отчета об операции.

Система проверки целостности данных на диске имеет три фазы – выбор алгоритма вычисления контрольной суммы, ввод эталонной контрольной суммы с ПЭВМ и вычисление контрольной суммы с выводом отчета, в котором содержится результат сравнения и время вычисления контрольной суммы.

Таблица 1 – Работа устройства для проверки целостности данных на диске

|  |  |
| --- | --- |
| Этап работы | Описание |
| Переключение на 1 фазу | Вывод текущего алгоритма на дисплее, предложение ввести одну из фраз с ПЭВМ |
| Постоянный эффект фазы 1 | Вывод текущего алгоритма на дисплее, вывод на ПЭВМ предложения ввести управляющую команду |
| Переключение на 2 фазу | Вывод на ПЭВМ подсказки для ввода контрольной суммы определенного размера |
| Постоянный эффект фазы 2 | – |
| Переключение на фазу 3 | Вычисление контрольной суммы диска, вывод отчета на LED-дисплей, вывод звука на динамик при несовпадении контрольных сумм |
| Постоянный эффект фазы 3 | – |

После выполнения последнего действия у пользователя есть возможность либо нажать на кнопку «Restart», что переключит программу на начальную фазу, при этом SD-карта не будет смонтирована снова, или же нажать кнопку «Reset», тогда выполнение программы будет перезапущено на уровне микроконтроллера.

Также пользователь может нажать любую из этих кнопок в любую из фаз работы программы.

Кроме того, в программе, как и говорилось выше, есть несколько способов выбрать алгоритм – с помощью ПЭВМ или с помощью кнопок. Существует два сценария:

* Пользователь нажимает кнопку NEXT или PREV, тогда текущий алгоритм переключается на следующий или предыдущий соответственно.
* Пользователь вводит в терминал слово «NEXT» или «PREV», тогда текущий алгоритм также переключается на следующий или предыдущий соответственно. При этом ввод любой другой фразы не вызывает никаких действий.

Всего программа поддерживает три алгоритма вычисления контрольной суммы – один поддерживаемый аппаратно и два реализованных вручную. Аппаратно поддерживаемый алгоритм – CRC. Написанные вручную – CRC8 по алгоритму Купмена и MD5. Стоит заметить, что в зависимости от алгоритма контрольная сумма имеет разный размер. При этом контрольная сумма всегда выводится в шестнадцатеричном формате.

Таблица 2 – Доступные в программе алгоритмы

|  |  |
| --- | --- |
| Название алгоритма | Размер контрольной суммы в символах |
| Аппаратная реализация CRC | 8 |
| Реализация CRC8 по алгоритму Купмена | 8 |
| MD5 | 32 |

После вычисление контрольной суммы она выводится на LED-дисплей, при этом также выводится, совпадает она с эталонной или нет. Если не совпадает, то на динамик в течение секунды выводится звуковой сигнал, обозначающий несовпадение.

Разработанная структурная схема устройства для проверки целостности данных на диске представлена на рисунке 1.

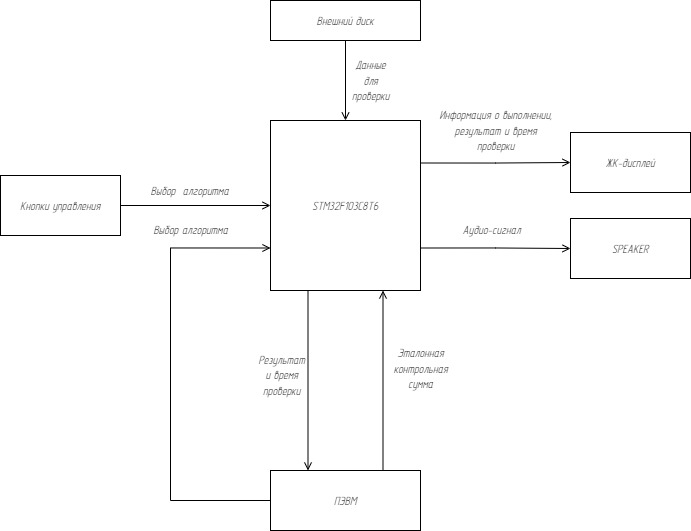


Рисунок 1 – Структурная схема устройства

## 1.2 Проектирование функциональной схемы

В этом разделе приведено функциональное описание работы системы и проектирование функциональной схемы.

### 1.2.1 Микроконтроллер STM32F103C8T6

Основным элементом разрабатываемого устройства является микроконтроллер (МК). Существует множество семейств МК, для разработки выберем из тех, что являются основными [1]:

* 8051 – это 8-битное семейство МК от компании Intel.
* PIC – это серия МК, разработанная компанией Microchip;
* AVR – это серия МК разработанная компанией Atmel;
* ARM – одним из семейств процессоров на базе архитектуры RISC, разработанным компанией Advanced RISC Machines.

Сравнение семейств показано в таблице 3.

Таблица 3 – Сравнение семейств МК

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | 8051 | PIC | AVR | ARM |
| Разрядность | 8 бит | 8/16/32 бит | 8/32 бит | 32 бит, иногда 64 бит |
| Интерфейсы | UART, USART,  SPI, I2C | PIC, UART, USART, LIN, CAN, Ethernet, SPI, I2S | UART, USART, SPI, I2C, иногда CAN, USB, Ethernet | UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI, IrDA, FATFS |
| Скорость | 12 тактов на инстру-кцию | 4 такта на инструкцию | 1 такт на инструкцию | 1 такт на инструкцию |
| Память | ROM, SRAM, FLASH | SRAM, FLASH | Flash, SRAM, EEPROM | Flash, SDRAM, EEPROM |
| Энергопо-требление | Среднее | Низкое | Низкое | Низкое |
| Объем FLASH памяти | До 128 Кб | До 512 Кб | До 256 Кб | До 2056 Кб |

Было выбрано семейство ARM, так как для разрабатываемой системы нужна высокая скорость работы интерфейсов для отрисовки текста на жк-дисплее и работы с SD-картой. Кроме того, для программы потенциально понадобится большой объем FLASH-памяти, чтобы вместить работу конечного автомата, работу с SD-картой и алгоритмы.

ARM включает в себя немалое количество семейств, поэтому рассмотрим только основные

1. STM32, имеющие следующие характеристики:

* Flash-память до 2056 Кбайт;
* RAM до 1,4 Мбайт;
* Максимальная частота ядра до 480 МГц;
* число пинов (ножек) ввода-вывода 16–64;
* самый разнообразный набор периферии

1. NXP, имеющие следующие характеристики:

* FLASH до 2048 Кбайт;
* RAM до 8096 Кбайт;
* Максимальная частота ядра до 360 МГц;
* число пинов ввода-вывода 16-64;
* самый разнообразный набор периферии

1. Toshiba, имеющие следующие характеристики:

* FLASH до 1,5Mбайт;
* RAM до 514 Кбайт;
* Максимальная частота ядра до 120 МГц;
* самый разнообразный набор периферии

Выберем подсемейство STM32 от ST Microelectronics, так как у них самая активная поддержка сообщества, что поможет использовать некоторые готовые решения, например, для взаимодействия с файловой системой FAT на SD-карте. Кроме того, мы имели дело с представителем этого подсемейства в рамках лабораторных работ курса «Микропроцессорные системы», что также является плюсом при выборе.

В подсемействе STM32 семейства ARM был выбран МК STM32F103C8T6, обладающий всем необходимым функционалом для реализации проекта:

* 2 интерфейса SPI для программирования SD-карты и для ЖК-дисплея;
* интерфейс UART для ПЭВМ(виртуального терминала);
* 20 Кбайт RAM;
* 4 таймера, которые могут быть использованы в режиме ШИМ для генерации звукового сигнала;
* 64 Кбайта FLASH-памяти;
* Возможность назначить внешнее прерывание практически на любой PIN;
* Поддержка CRC для вычисления контрольной суммы;
* Поддержка FATFS для файловой системы FAT на SD-карте;
* частота работы до 72 Мгц.

А также с данным МК уже есть опыт работы, что упростит разработку, и не потребует траты времени на изучение функционала МК.

Это экономичный 32-разрядный микроконтроллер, основанный на RISC архитектуре. STM32F103C8T6 обеспечивает производительность 1 миллион операций в секунду на 1 МГц синхронизации за счет выполнения большинства инструкций за один машинный цикл и позволяет оптимизировать потребление энергии за счет изменения частоты синхронизации. Структурная схема МК показана на рисунке 2 и УГО на рисунке 3 [2].

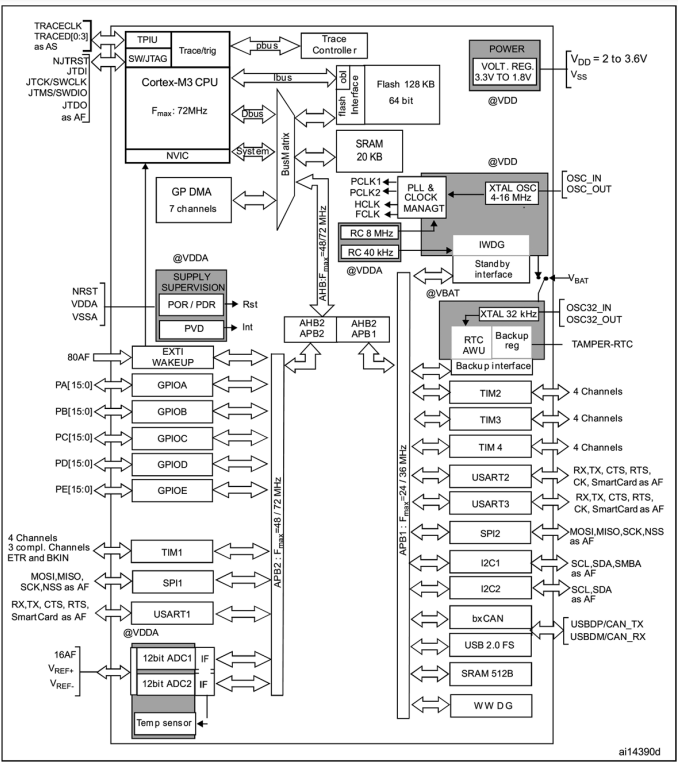


Рисунок 2 – Структурная схема МК STM32F103C8T6

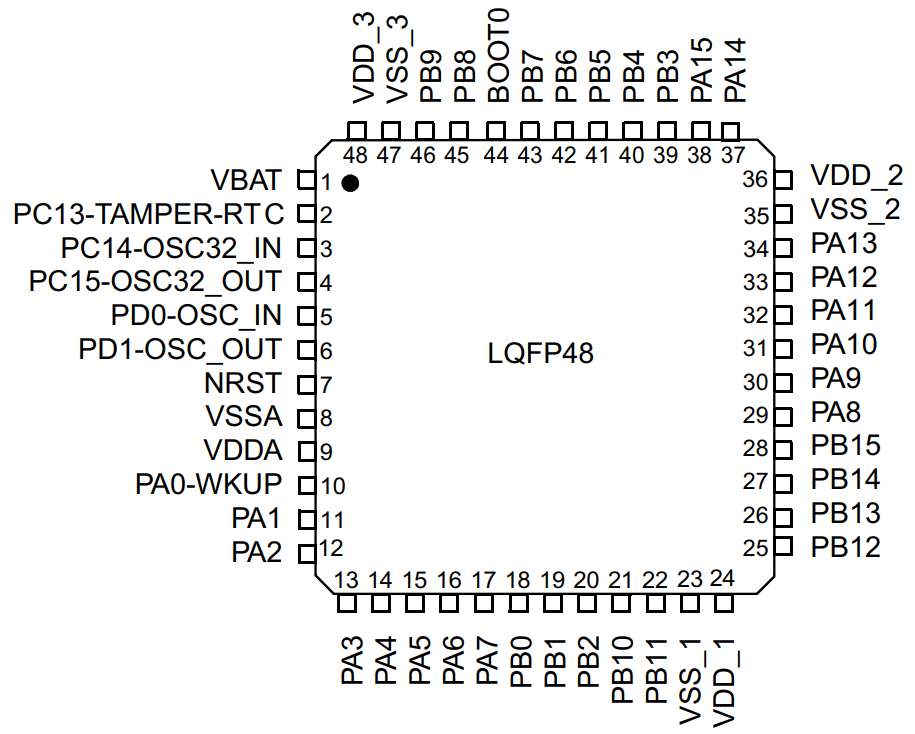
****

Рисунок 3 – УГО МК STM32F103C8T6

Он обладает следующими характеристиками [3]:

1. Архитектура и производительность:

* Процессор Cortex-M3 от ARM с частотой до 72 МГц.
* 32-битная архитектура с набором команд Thumb-2 для эффективной работы.

1. Память:

* 64 КБ флеш-памяти для программного кода.
* 20 КБ ОЗУ (SRAM) для хранения данных.
* Возможность расширения памяти с использованием внешних устройств.

1. Периферийные устройства:

* Несколько портов GPIO (General-Purpose Input/Output) для подключения и управления внешними устройствами.
* USART, SPI, I2C и другие интерфейсы для обмена данными с внешними устройствами.
* АЦП (аналогово-цифровой преобразователь) для измерения аналоговых сигналов.

1. Интерфейсы и коммуникации:

* USB-интерфейс для обмена данными с компьютером или другими устройствами.
* Возможность работы с различными протоколами связи, такими как CAN (Controller Area Network), Ethernet и другими.

1. Прочие особенности:

* Встроенные таймеры и счетчики для управления временем и частотой.
* Низкое энергопотребление в режиме ожидания.
* Защита от переполнения стека и ошибок программирования.

1. Программирование и разработка:

* Поддержка различных интегрированных сред разработки (IDE), таких как Keil, STM32CubeIDE, и других.
* Обширная документация, примеры кода и библиотеки для упрощения разработки.

1. Применение:

* Широко используется в различных приложениях, включая промышленные системы управления, автоматизацию, умные устройства, медицинское оборудование, робототехнику и многое другое.

1. Напряжение питания: 2– 3.6В.

### 1.2.1.1 Используемые элементы

Для функционирования устройства проверки целостности данных на диске в МК STM32F103C8T6 задействованы не все элементы его архитектуры. Выделим и опишем те, что используются во время функционирования схемы [4].

Порты А, В – назначения каждого из них описано в разделе *1.2.1.2*.

Указатель стека – используется для работы со стеком, при вызове подпрограмм. В коде они присутствуют.

SRAM – статическая память МК, где хранятся объявленные переменные.

Регистры общего назначения – предназначены для хранения операндов арифметико-логических операций, а также адресов или отдельных компонентов адресов ячеек памяти.

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

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

SPI – интерфейс для связи МК с другими внешними устройствами. В проекте используется для прошивки МК, для связи с внешним диском, а также для связи с LCD-дисплеем.

Программный счетчик – используется для указания следующей команды выполняемой программы.

Память Flash – память МК, в котором хранится загруженная в него программа.

Регистры команды – 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.

Сигналы управления – синхронизируют обработку данных.

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

Таймеры/счетчики – включает в себя 4 16-ти разрядных таймера TIM1, TIM2, TIM3, TIM4. В проекте используется только один канал TIM1 для генерации ШИМ для динамика.

Управление синхронизацией и сбросом (MCU CTRL. & Timing) – в этом блоке обрабатываются тактовые сигналы и принимается сигнал сброса.

Прерывания – контроллер прерываний обрабатывает внешние прерывания и прерывания от периферийных устройств МК (таймеров, портов ввода/вывода). В проекте используются внешние прерывания для обработки нажатия кнопок и прерывания UART для получения ввода с ПЭВМ.

UART – через этот интерфейс в МК передается информация из ПЭВМ.

Генератор – генератор тактовых импульсов. Необходим для синхронизации работы МК.

### 1.2.1.2 Распределение портов

МК STM32F103C8T6 содержит пять портов – А, В, С, D и Е. Опишем назначение тех, что используются в данной системе для её функционирования.

Порт А:

* РА2 – отправка данных по UART на ПЭВМ;
* РА3 – получение данных по UART с ПЭВМ;
* PA5 – тактовый сигнал (SCK) для SPI для LCD-дисплея;
* PA7 – MOSI-пин для SPI для LCD-дисплея;
* PA10 – RES-пин(RESET) для RESET-сигнала для LCD-дисплея;
* PA11 – DC-пин(Data or Command) для передачи данных или команд на LCD-дислей;
* PA12 – CS-пин(Chip Select) для определения активного устройства, взаимодействующего по SPI(в нашем случае это всегда микроконтроллер) с LCD-дисплеем;
* PA15 – выход первого канала таймера TIM2, на котором генерируется ШИМ для динамика.

Порт В:

* РВ5 – пин с внешним прерыванием по нажатии кнопки переключения на следующий алгоритм;
* РВ6 – пин с внешним прерыванием по нажатии кнопки переключения на предыдущий алгоритм;
* РВ7 – пин с внешним прерыванием по нажатии кнопки подтверждения выбора алгоритма;
* РВ8 – пин с внешним прерыванием по нажатии кнопки перезапуска вычислений;
* РВ10 – CS-пин(Chip Select) для определения активного устройства, взаимодействующего по SPI(в нашем случае это всегда микроконтроллер) с сокетом SD-карты;
* PB13 - тактовый сигнал (SCK) для SPI для сокета SD-карты;
* РВ14 – MISO-пин для SPI для сокета SD-карты;

### 1.2.1.3 Организация памяти

Схема организации памяти МК STM32F103C8T6 показана на рисунке 4.

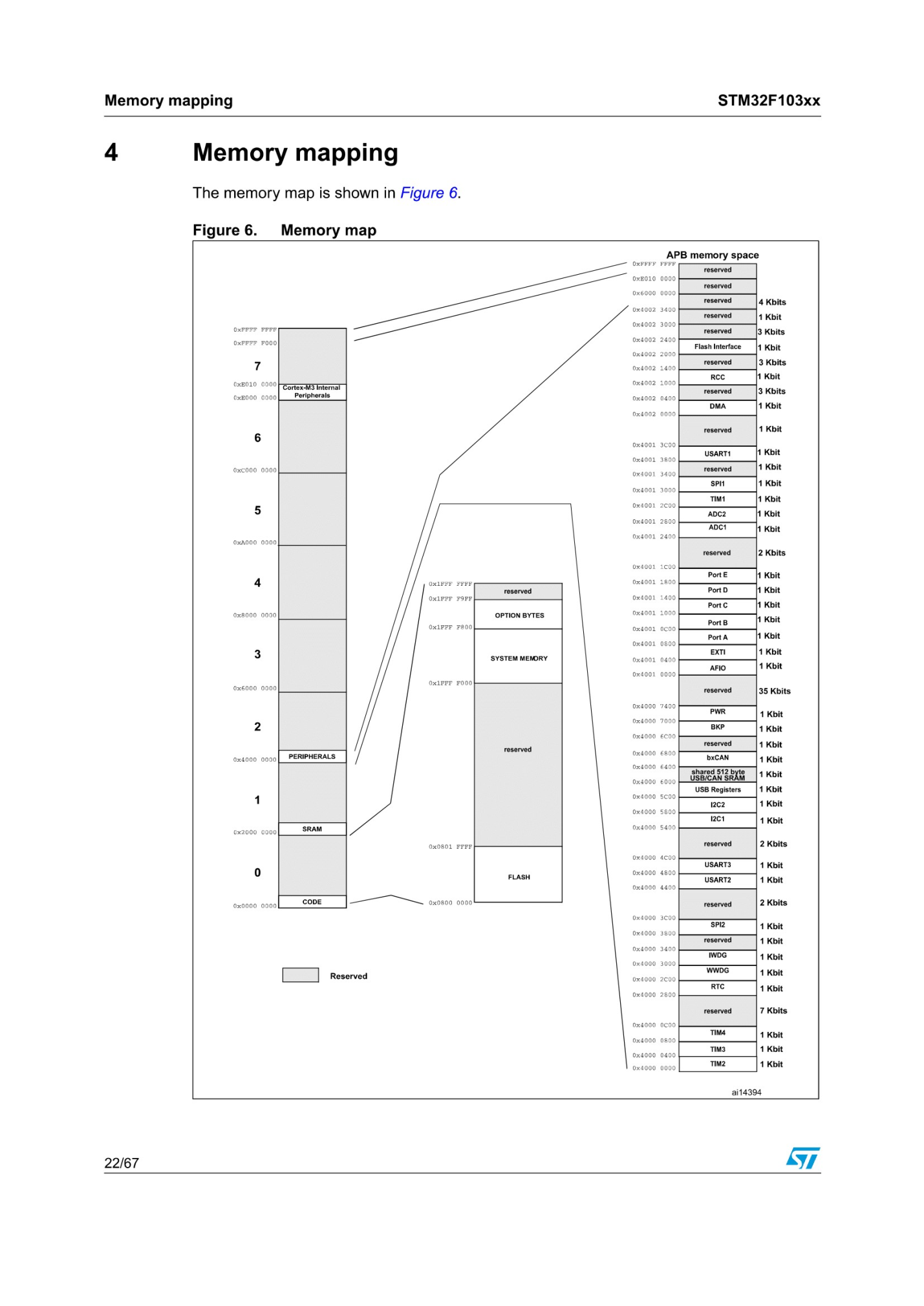


Рисунок 4 – Организация памяти МК STM32F103C8T6

Что-то написать

### 1.2.2 Прием данных от ПЭВМ

Приём данных от ПЭВМ происходит через драйвер MAX232. MAX232 – интегральная схема, преобразующая сигналы последовательного порта RS-232 в цифровые сигналы.

RS-232 – стандарт физического уровня для синхронного и асинхронного интерфейса (USART и UART). Обеспечивает передачу данных и некоторых специальных сигналов между терминалом и устройством приема. Сигнал, поступающий от интерфейса RS-232, через преобразователь передается в микроконтроллер на вход RxD.

К внешнему устройству MAX232 подключен через разъем DB-9. На схеме условное обозначение – ХР2.

Внутреннее изображение MAX232 показано на рисунке 5. Назначение пинов описано в таблице 4 [5].

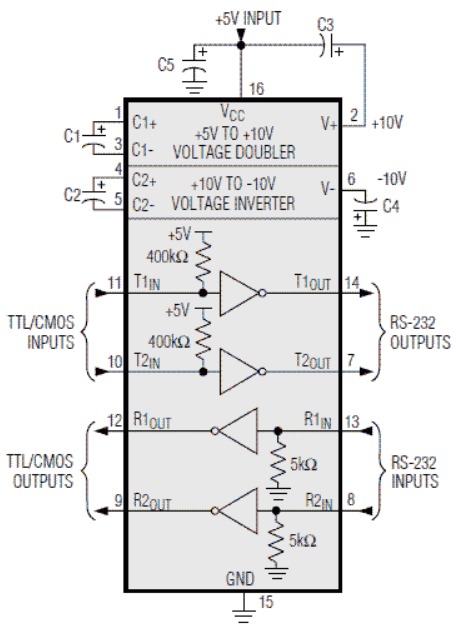


Рисунок 5 – Преобразователь MAX232

Таблица 4 - Назначение пинов MAX232

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Имя | Тип | Описание |
| 1 | C1+ | – | Положительный вывод C1 для подключения конденсатора |
| 2 | VS+ | O | Выход положительного заряда для накопительного конденсатора |
| 3 | C1- | – | Отрицательный вывод C1 для подключения конденсатора |
| 4 | C2+ | – | Положительный вывод C2 для подключения конденсатора |
| 5 | C2- | – | Отрицательный вывод C2 для подключения конденсатора |
| 6 | VS- | O | Выход отрицательного заряда для накопительного конденсатора |
| 7, 14 | T2OUT, T1OUT | O | Вывод данных по линии RS232 |
| 8, 13 | R2IN, R1IN | I | Ввод данных по линии RS232 |
| 9, 12 | R2OUT, R1OUT | O | Вывод логических данных |
| 10, 11 | T2IN, T1IN | I | Ввод логических данных |
| 15 | GND | – | Земля |
| 16 | Vcc | – | Напряжение питания, подключение к внешнему источнику питания 5 В |

Когда микросхема MAX232 получает на вход логический "0" от внешнего устройства, она преобразует его в напряжение от +5 до +15В, а когда получает логическую "1" - преобразует её в напряжение от -5 до -15В, и по тому же принципу выполняет обратные преобразования от RS-232 к внешнему устройству.

### 1.2.3 Настройка USART для взаимодействия с ПЭВМ

Интерфейс USART (Universal Synchronous Asynchronous Receiver Transmitter) в микроконтроллерах STM32 представляет собой универсальный последовательный интерфейс, который может работать в режиме синхронной или асинхронной передачи данных. Он обеспечивает возможность обмена данными между микроконтроллером и другими устройствами, такими как датчики, модули связи и периферийные устройства.

USART в STM32 поддерживает передачу данных через одну линию для приема (RX) и одну для передачи (TX). Он также может работать в полудуплексном режиме, когда одна линия используется для передачи и приема данных.

USART может настраиваться на разные скорости передачи данных (бодрейты), количество бит данных, контроль четности, стоповые биты и другие параметры через специальные регистры микроконтроллера. Это обеспечивает гибкость в настройке передачи данных в соответствии с требованиями конкретного приложения.

USART был выбран для использования, так как выбор стоял между ним и I2C – оба интерфейса SPI уже были заняты под более критичные по скорости задачи. USART был выбран как более быстрый интерфейс; кроме того, его использование это классический ход при работе с терминалом.

В разрабатываемой системе USART используется в асинхронном режиме для вывода текста на виртуальный терминал и для чтения эталонной контрольной суммы с виртуального терминала, который выступает в роли ПЭВМ. Рассмотрим настройку USART для этого конкретного приложения и регистры, с помощью которых это делается.

Настройка USART в разрабатываемой системе показана на рисунках 5 и 6.

|  |  |
| --- | --- |
|  |  |

Рисунки 5,6 – Настройка USART

Таким образом, USART используется в асинхронном режиме, контроль сигнала CTS/RTS отключен, baud rate – 115200 бит/с, длина каждой посылки – 8 бит, включая бит четности, контроль четности отключен, используется один стоп-бит, оверсемплинг в режиме 16-семплирования. Кроме того, включены прерывания для USART.

Оверсемплинг в USART относится к технике, используемой для приема данных в асинхронном режиме. Эта техника помогает улучшить точность синхронизации битов данных, особенно при работе с высокими скоростями передачи данных.

Оверсемплинг подразумевает выбор частоты сэмплирования (число раз, которое система измеряет состояние входного сигнала за определенный промежуток времени) значительно выше, чем минимально необходимая частота для корректного считывания данных.

В USART для асинхронной передачи, оверсемплинг обычно используется для более точного определения момента прихода каждого бита данных. К примеру, в режиме 16-семплирования (16x oversampling), каждый бит данных будет сэмплироваться 16 раз за период передачи, что улучшает точность считывания данных и помогает бороться с потерей или искажением сигнала в условиях шумов или неполадок в канале связи.

Эта техника позволяет повысить устойчивость и надежность приема данных по USART, особенно при работе на высоких скоростях передачи данных или в условиях, где возможны помехи или искажения сигнала.

Всего существует 7 регистров, связанных с настройкой и работой USART: USART\_SR (Status register), USART\_DR (Data register), USART\_BRR (Baud rate register), USART\_CR1 (Control register 1), USART\_CR2 (Control register 2), USART\_CR3 (Control register 3), USART\_GTPR (Guard time and prescaler register). Ниже будут описаны все регистры кроме неиспользованных регистров для настройки.

Начнем с настройки USART. Для этого используются control-регистры и регистр управления скоростью передачи. Начнем с USART\_CR1. Его изображение представлено на рисунке 7.

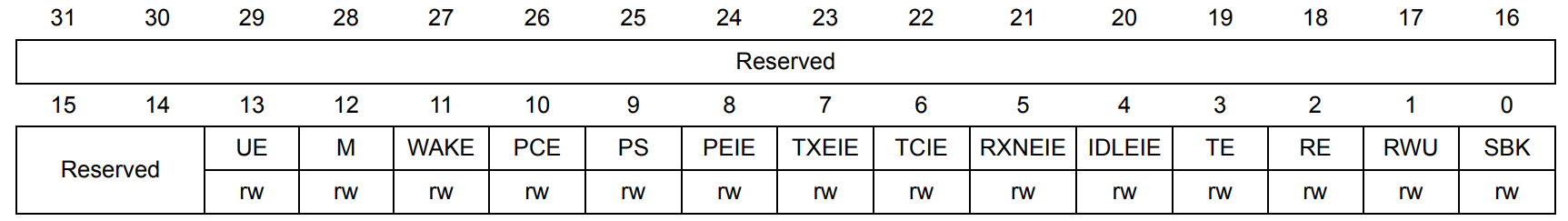


Рисунок 7 – Регистр USART\_CR1

Описание регистра:

**UE:**USART enable - включить USART (включается установкой бита в 1).

**M:**Word length - длина слова, задаёт количество бит данных в одном фрейме. Бит не должен модифицироваться в процессе обмена данными (это касается как передачи, так и приёма). 0 - 1 старт-бит, 8 бит данных, n стоп-битов; 1 - 1 старт-бит, 9 бит данных, n стоп-битов. Примечание. Бит чётности считается битом данных.

**WAKE:**Wakeup method - метод пробуждения USART. 0 - "линия свободна" (Idle line); 1- адресная метка.

**PCE:**Parity control enable - включить аппаратный контроль чётности (генерация бита чётности при передаче данных и проверка в принимаемых данных).

**PS:**Parity selection - выбор метода контроля чётности. Выбор происходит после завершения передачи/приёма текущего байта.  
0 - контроль на чётность; 1 - контроль на нечётность.

**PEIE:**PE interrupt enable - разрешение прерывания от PE. 0 – прерывание запрещено; 1 – генерируется прерывание от USART, когда USART\_SR.PE==1.

**TXEIE:**TXE interrupt enable - разрешение прерывания от TXE. 0 – прерывание запрещено; 1 – генерируется прерывание от USART, когда USART\_SR.TXE==1.

**TCIE:**Transmission complete interrupt enable - разрешение прерывания после завершения передачи. 0 – прерывание запрещено; 1 – генерируется прерывание от USART, когда USART\_SR.TC==1.

**RXNEIE:**RXNE interrupt enable - разрешение прерывания от RXNE. 0: прерывание запрещено; 1: генерируется прерывание от USART, когда USART\_SR.ORE==1 или USART\_SR.RXNE==1.

**IDLEIE:**IDLE interrupt enable - разрешение прерывания при обнаружении, что "линия свободна" (Idle line). 0: прерывание запрещено; 1: генерируется прерывание от USART, когда USART\_SR.IDLE==1.

**TE:**Transmitter enable - включить передатчик USART (включается установкой бита в 1).

**RE:**Receiver enable - включить приёмник USART (включается установкой бита в 1). После установки бита, приёмник начинает поиск старт-бита во входном сигнале.

**RWU:**Receiver wakeup - переводит USART в тихий режим. Этот бит устанавливается и сбрасывается программно, а также может сбрасываться аппаратно при обнаружении пробуждающей последовательности.

**SBK:**Send break - отправить Break посылку. Бит может быть установлен и сброшен программно. Его необходимо программно установить в 1 для формирования Break посылки, он будет сброшен аппаратно во время формирования stop-бита в Break фрейме. 0: Break-символ не передаётся; 1: Break-символ будет передан.

Теперь опишем регистр BRR, с помощью которого контролируется скорость передачи данных через USART. Регистр представлен на рисунке 10.

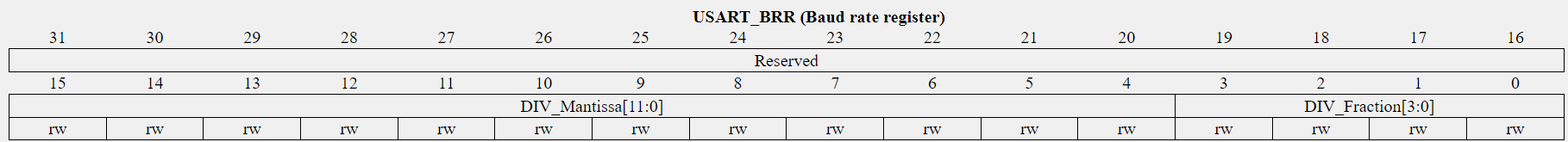


Рисунок 10 – Регистр BRR

**DIV\_Mantissa[11:0]:**mantissa of USARTDIV - целая часть коэффициента деления делителя частоты.

**DIV\_Fraction[3:0]:**fraction of USARTDIV - дробная часть коэффициента деления. В режиме с OVER8==1 в битовом поле DIV\_Fraction[3:0] старший бит [3] не используется и должен быть сброшен.

С помощью регистра USART\_BRR задаётся скорость передачи - одновременно как для приёмника USART, так и для передатчика. На рисунке 11 представлена схема, показывающая, как именно высчитывается скорость передачи.

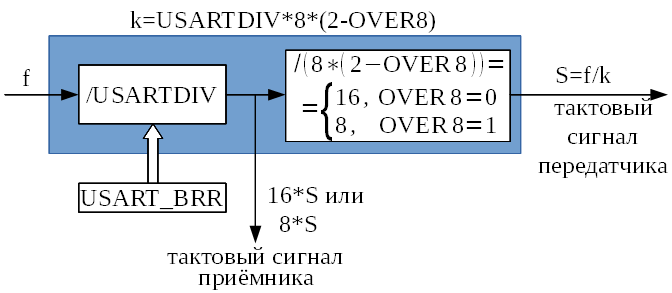


Рисунок 11 – Вычисление скорости приема и передачи

В данной системе было принято решение использовать baud rate = 115200, поэтому был выставлен USART\_BRR = 69. 8000000 / 69 = 115942 ~ 115200.

Далее рассмотрим USART\_DR – регистр, через который передаются непосредственно данные. Он представлен на рисунке 12.

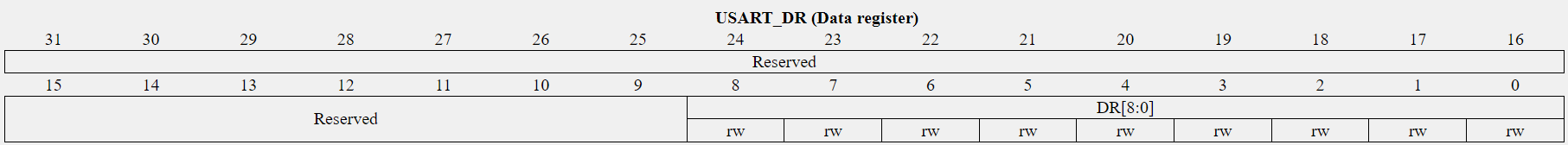


Рисунок 12 – Регистр данных

**DR[8:0]:**Data value - регистр данных. Содержит полученный или передаваемый символ, в зависимости от того, производится чтение из него или запись в регистр. Регистр выполняет двойную функцию за счёт того, что он является составным, он объединяет в себе два регистра: один для передачи (TDR) и один для приёма (RDR). TDR обеспечивает загрузку данных в выходной сдвигающий регистр, сдвигающий регистр преобразует загруженное в него слово в последовательную форму. Получаемые в последовательной форме данные накапливаются в приёмном сдвигающем регистре, когда фрейм получен полностью, данные из сдвигающего регистра передаются в регистр RDR, который реализует параллельный интерфейс между внутренней шиной микроконтроллера и входным сдвигающим регистром.

Когда осуществляется передача данных с включённым контролем чётности (USART\_CR1.PCE==1), старший бит, записываемый в регистр USART\_DR (бит [7] или [8], в зависимости от выбранной длины слова, см. USART\_CR1.M), не учитывается. Он замещается вычисленным битом чётности.

При получении данных с включённым контролем чётности, при чтении из USART\_DR будем получать значение, содержащее полученный бит чётности.

Последний рассматриваемый регистр в USART – USART\_SR(status register). Он представлен на рисунке 13.

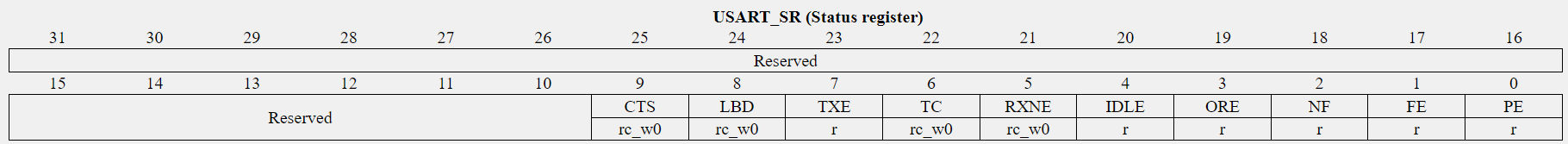


Рисунок 13 – Регистр статуса

**CTS:**CTS flag - флаг изменения состояния nCTS. Устанавливается аппаратно, когда происходит переключение сигнала на входе nCTS. Если установлен бит CTSIE (USART\_CR3.CTSIE==1), то при установке флага генерируется прерывание. Флаг сбрасывается программно записью 0.

**LBD:**LIN break detection flag - флаг приёма посылки Break. Устанавливается аппаратно при обнаружении посылки Break на входе; если установлен бит LBDIE (USART\_CR3.LBDIE==1), то генерируется прерывание. Флаг срсывается программно записью 0.

**TXE:**Transmit data register empty - флаг устанавливается аппаратно, когда содержимое регистра передаваемых данных TDR пересылается в сдвигающий регистр (доступ к TDR осуществляется путём записи в регистр USART\_DR). Если установлен бит TXEIE (USART\_CR1.TXEIE==1), генерируется прерывание. Флаг сбрасывается путём записи в регистр USART\_DR.

**TC:**Transmission complete - флаг завершения передачи, устанавливается аппаратно, если передача фрейма завершена, и флаг TXE установлен (т.е. регистр передаваемых данных пуст, больше нет данных для передачи). Если USART\_CR1.TCIE==1, то при установке флага генерируется прерывание. Флаг сбрасывается программно последовательностью действий: чтение регистра USART\_SR, затем запись в USART\_DR. Также бит может быть сброшен записью в него 0. Примечание. После сброса этот бит установлен.

**RXNE:**Read data register not empty - регистр данных для чтения не пуст. Флаг устанавливается аппаратно, когда содержимое принимающего сдвигающего регистра передаётся в регистр принимаемых данных RDR. Если USART\_CR1.RXNEIE==1, при этом генерируется прерывание. Флаг сбрасывается чтением из регистра USART\_DR. Также бит может быть сброшен записью в него 0.

**IDLE:**IDLE line detected - линия свободна. Флаг устанавливается аппаратно, если обнаружено что линия свободна. Это происходит, если получен целый фрейм единиц. При этом генерируется прерывание, если USART\_CR1.IDLEIE==1. Флаг сбрасывается программно последовательностью действий: чтение регистра USART\_SR с последующим чтением из регистра USART\_DR.

**ORE:**Overrun error - ошибка переполнения. Флаг устанавливается аппаратно, когда слово, полученное в сдвигающей регистр готово к перемещению в регистр принимаемых данных RDR, но RXNE==1 (регистр RDR не пуст, содержит ещё не прочитанные из него принятые USART данные). Если USART\_CR1.RXNEIE==1, то при установке флага генерируется исключение. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART\_SR с последующим чтением из USART\_DR.

**NF:**Noise detected flag - флаг устанавливается аппаратно при обнаружении шума в полученном фрейме. Сбрасывается программно последовательностью действий: чтение из регистра USART\_SR, затем чтение из регистра USART\_DR.

**FE:**Framing error - ошибка фрейма. Флаг устанавливается аппаратно в случае нарушения синхронизации, чрезмерного шума в линии, при обнаружении символа Break. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART\_SR, затем чтение из регистра USART\_DR. Примечание. В отношении генерации прерывания этот флаг полностью аналогичен флагу NF.

**PE:**Parity error - ошибка чётности. Флаг устанавливается аппаратно, когда в принятом фрейме обнаружена ошибка чётности (если контроль чётности включён). Если USART\_CR1.PEIE==1, то генерируется прерывание. Флаг сбрасывается программно последовательностью действий: чтение из регистра USART\_SR, затем чтение либо запись регистра USART\_DR. Перед сбросом флага, программа должна дождаться установки флага RXNE (регистр данных для чтения не пуст).

### 1.2.4 LCD-дисплей ST7735

Почему выбран, полное описание, какие команды принимает, работа в схеме

### 1.2.5 Настройка SPI для взаимодействия с LCD-дисплеем

Описание SPI, регистры, скорость, работа в программе.

### 1.2.6 Настройка SPI для взаимодействия с сокетом SD-карты.

Работа в программе

### 1.2.7 Генератор тактовых импульсов и сброс

Написать про использование внутреннего генератора частоты и про сброс

### 1.2.8 Использование таймера для генерации звукового сигнала

Микроконтроллеры STM32F103xx имеют в своём составе множество таймеров с большим количеством поддерживаемых функций. С помощью любого таймера можно формировать интервалы времени с требуемой длительностью с генерацией прерывания или DMA запроса по окончании интервала. Кроме того, можно формировать одиночные импульсы заданной длительности или периодические импульсы с заданной длительностью и частотой повторения; подсчитывать количество импульсов внешнего сигнала (счётчик может работать в режиме сложения или вычитания); поддерживается режим широтно-импульсной модуляции.

В данной системе используется только один из каналов одного таймера общего назначения. Он используется для генерации ШИМ-сигнала для динамика в случае, если эталонная и вычисленная контрольные суммы не совпали. Используется первый канал второго таймера. Его конфигурация представлена на рисунке 13.

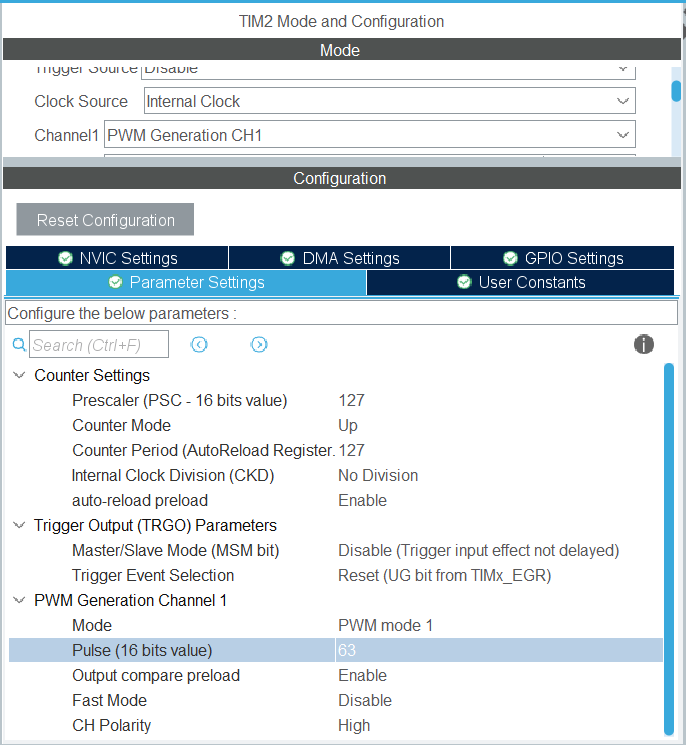


Рисунок 13 – Конфигурация таймера TIM2 для работы в ШИМ-режиме

Далее рассмотрим, как именно данные настройки влияют на реальную работу таймера.

Структурная схема таймера представлена на рисунке 14.

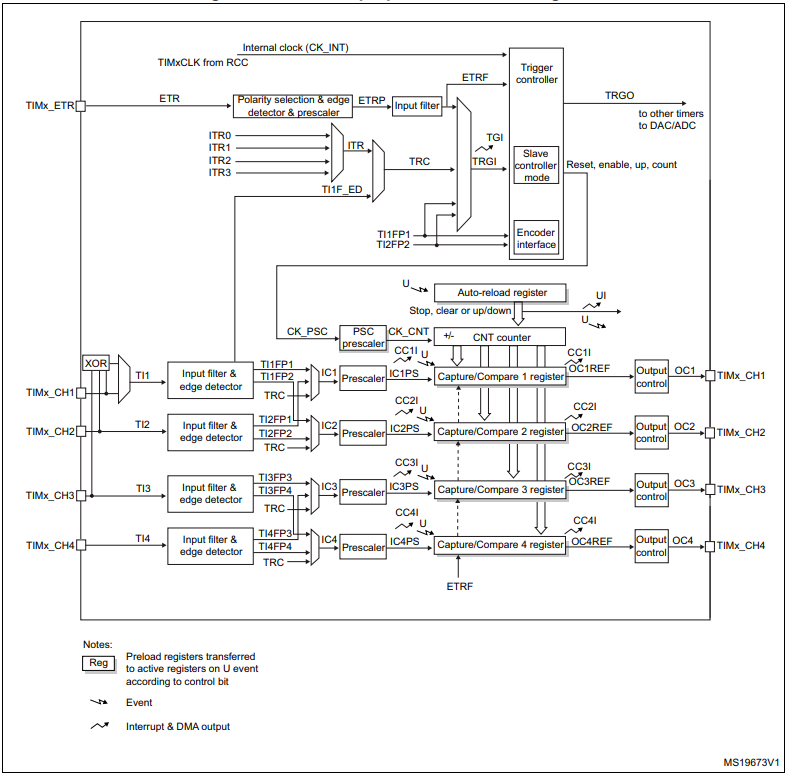


Рисунок 14 – Структурная схема таймера

Таймер непосредственно работает через регистры TIM2\_CNT, TIM2\_PSC и TIM2\_ARR. Они полностью содержат в себе значения текущего счетчика таймера, значение прескейлера и значение автоматической перезагрузки соответственно, и более подробно их рассматривать смысла нет. Прескейлер и автоматическая перезагрузка были выставлены как 127.

Также таймер имеет множество регистров для настройки режима работы. Я рассмотрю только те, которые необходимо было настроить вручную. Начнем с регистра TIM2\_CR1, представленного на рисунке 15.

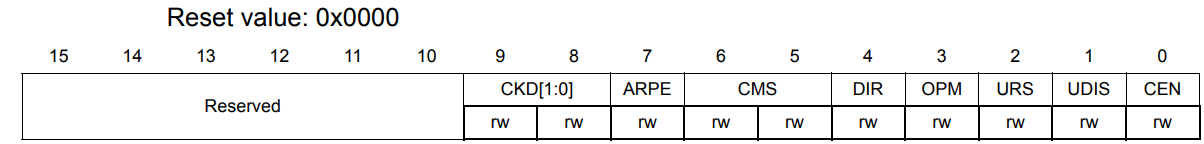


Рисунок 15 – Регистр TIM2\_CR1

**CKD**: Clock division

**ARPE:**Auto-reload preload enable. Бит для включения режима предзагрузки регистра TIMx\_ARR: 0: TIMx\_ARR не буферизируется; 1: используется буферизация регистра TIMx\_ARR. Когда буферизация включена, новое значение, записанное в регистр, начинает использоваться после очередного события обновления.

**CMS**: Center-aligned mode selection

**DIR**: Direction

**OPM:**One-pulse mode.

**URS:**Update request source.

**UDIS:**Update disable.

**CEN:**Counter enable.

Далее рассмотрим TIM2\_CCMR1(Compare and capture mode register), который представлен на рисунке 16. Каналы могут быть использованы в режимы захвата(input) и в режиме сравнения(output). При этом одни биты могут иметь разный смысл в зависимости от режима. Для режима ШИМ необходим режим сравнения, поэтому нужно смотреть на верхний ряд битов.

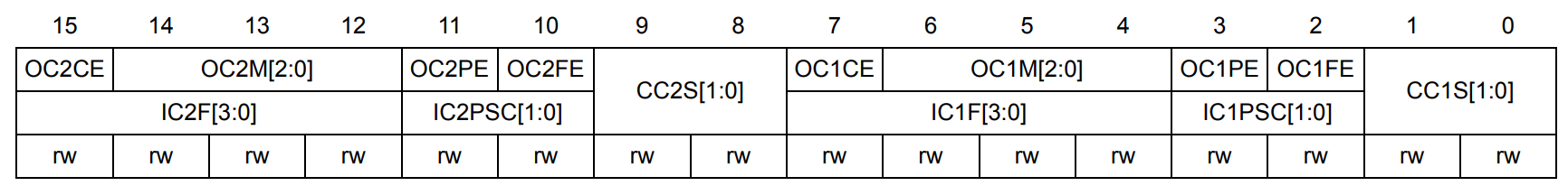


Рисунок 16 – Регистр TIM2\_CR2

**OC2CE**: Output compare 2 clear enable

**OC2M**[2:0]: Output compare 2 mode

**OC2PE**: Output compare 2 preload enable

**OC2FE**: Output compare 2 fast enable

**CC2S**[1:0]: Capture/Compare 2 selection

**OC1CE**: Output compare 1 clear enable

**OC1M**: Output compare 1 mode. Эти биты определяют поведение выходного сигнала. Для режима ШИМ существует два вариант – 110 (PWM Mode 1) и 111 (PWM Mode 2). Был выбран PWM Mode 1, который выдает на пин единицу при возрастающем счетчике и 0 при убывающем(PWM Mode 2 действует наоборот).

**OC1PE**: Output compare 1 preload enable. 0 – предзагрузка отключена; 1 – предзагрузка включена. Для режима ШИМ предзагрузка должна быть включена.

**OC1FE**: Output compare 1 fast enable

**CC1S**: Capture/Compare 1 selection

Последний используемый регистр - TIM2\_CCR1(Compare and Capture register).

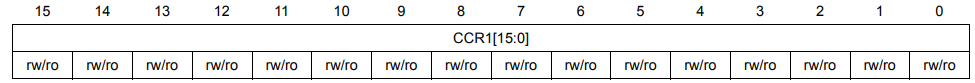


Рисунок 17 – регистр TIM2\_CCR1

Для режима ШИМ была выбрана частота 500Гц, так как это стандартная частота звукового генератора в Proteus 8. Выше перечисленные настройки как раз были сделаны, чтобы ее получить. Частота работы микроконтроллера – 8МГц. f = AutoReload \* Prescaler \* FreqPWM, тогда для того, чтобы получить FreqPWM примерно равную 500Гц, можно выставить AutoReload и Prescaler как 127. 8000000 / 127 / 127 ~= 500Гц. При этом используется коэффициент заполнения 50% - Для этого возьмем для Pulse примерно половину Prescaler – 63. Именно это значение будет храниться в TIM2\_CCR1.

### 1.2.9 Построение функциональной схемы

На основе всех вышеописанных сведений была спроектирована функциональная схема разрабатываемой системы, показанная на рисунке 6 – 7 и в приложении Б [8, 9].

Функциональная схема

Рисунок 7 – Функциональная схема устройства для проверки целостности данных на диске

-----------------------------------------------------------

## 1.3 Проектирование принципиальной схемы

### 1.3.1 Разъем программатора

Для программирования МК используется программатор, для его подключения необходим специальный разъем. Будет использован разъем IDC-06MS. Подключение программатора осуществляется при помощи интерфейса SPI, под что на МК ATmega8515 задействован порт РВ. На принципиальной схеме, которая показана в разделе *1.3.5*, условное обозначение – ХР1.

Он имеет следующие разъемы для подключения к МК:

* MISO – для передачи данных от микроконтроллера в программатор;
* SCK – тактовый сигнал;
* MOSI – для передачи данных от программатора в микроконтроллер;
* Reset – сигналом на RESET программатор вводит контроллер в режим программирования.

### 1.3.2 Подключение цепи питания

Для работы схемы, на неё необходимо подать напряжение, для этого будет использован блок питания ARDV-15-5B, имеющий следующие технические характеристики:

* выходной ток: 3 А;
* выходное напряжение: 5 В;
* входное напряжение: 100 – 240 В;
* пусковой ток: 40 А.

Диаметр центрального проводника – 2 мм и диаметр Jack-а – 2 мм.

Для подключения питания к схеме, будет использовано гнездо питания DS-201, имеющий диаметр центрального проводника – 2 мм и диаметр Jack-а – 2 мм. На принципиальной схеме, которая показана в разделе *1.3.5*, условное обозначение – ХS1.

### 1.3.3 Расчет сопротивления резисторов

Резисторы по своей сути предназначаются, чтобы уменьшить значение тока, подаваемого на устройство. В светофорном блоке резисторы установлены перед светодиодом и со стороны базы транзистора [13].

Сначала определим сопротивление резистора, который предназначен для ограничения величины тока, протекающего через светодиод.

= 105 Ом

где UR – напряжение питания на резисторе,

I – сила протекающего тока.

Ближайшие стандарты – это 100 Ом и 110 Ом. Выбран будет резистор с большим номиналом, то есть 110 Ом – CF-100.

Напряжение на резисторе рассчитывается по формуле:

UR = UK - ΔUVD - ΔUK = 5 - 2,5 - 0,4 = 2,1 В

где UK – напряжение питания;

ΔUVD – напряжение светодиода;

ΔUK – падение напряжение на переходе коллектор-эмиттер.

Ток, протекающий через светодиод, снизился:

= 0,019 А = 19 мА

Для определения резистора со стороны базы, посчитаем ток базы.

= 0,000095 мкА

где – ток коллектора, равный 0,019А;

β – коэффициент усиления биполярного транзистора, для модели 2N2222 равный 200.

= 22105 Ом = 22,1 кОм

Ближайшие стандарты – это 20 кОм и 25 кОм. Выбран будет резистор с большим номиналом, то есть 25 кОм – MO-100.

**1.3.4 Расчет потребляемой мощности**

Потребляемая мощность – это мощность, потребляемая интегральной схемой, которая работает в заданном режиме соответствующего источника питания.

Чтобы рассчитать суммарную мощность, рассчитаем мощность каждого элемента. На все микросхемы подается напряжение +5В. Мощность, потребляемая один устройством, в статическом режиме, рассчитывается формулой:

Р = U \* I

где U – напряжение питания (В);

I– ток потребления микросхемы (мА).

Также в схеме присутствуют резисторы и транзисторы. Мощность для резисторов рассчитывается по формуле:

Р = I2 \* R

где R – сопротивление резистора;

I – ток, проходящий через резистор.

Для транзисторов по формуле:

Р = U \* I

где U – напряжение коллектор-эмиттер;

I – ток коллектора (он был рассчитан в разделе *1.3.3*).

Расчет потребляемого напряжения для каждой микросхемы показан в таблице 8.

Таблица 8 – Потребляемая мощность

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Микросхема | Ток потребления, мА | Потребляемая мощность, мВт | Количе-  ство устройств | Суммарная потребляемая мощность, мВт |
| ATmega8515 | 40 | 200 | 1 | 200 |
| MAX232 | 8 | 40 | 1 | 40 |
| К155ЛА1 | 4 | 20 | 1 | 20 |
| К533ЛН1 | 3 | 15 | 1 | 15 |
| К155ЛА8 | 8 | 40 | 5 | 200 |
| L-793SRD-C | 18 | 47 | 12 | 564 |
| L-813GD | 18 | 47 | 12 | 564 |
| L-793YD | 18 | 47 | 4 | 188 |

Также в схеме используются 28 резисторов CF-100 и МО-100 с номиналом 110 и 25 кОм соответственно, 2 резистора CF-25 с номиналом 1 кОм, и 28 транзисторов 2N2222.

Pсуммарная = РATmega8515 + РMAX232 + РК155ЛА1 + РК533ЛН1 + РК155ЛА8 + РL-793SRD-C + РL-813GD + РL-793YD + Ррезисторов + Ртранзисторов = 200 + 40 + 20 + 15 + 200 + 564 + 564 + 188 + 56,5 + 42 = 2189,5 мВт

Суммарная потребляемая мощность системы равна 2189,5 мВт = 2,2 Вт.

**1.3.5 Построение принципиальной схемы**

На основе всех вышеописанных сведений была спроектирована принципиальная схема разрабатываемой системы, показанная на рисунке 8 – 9 и в приложении Б [8, 9].

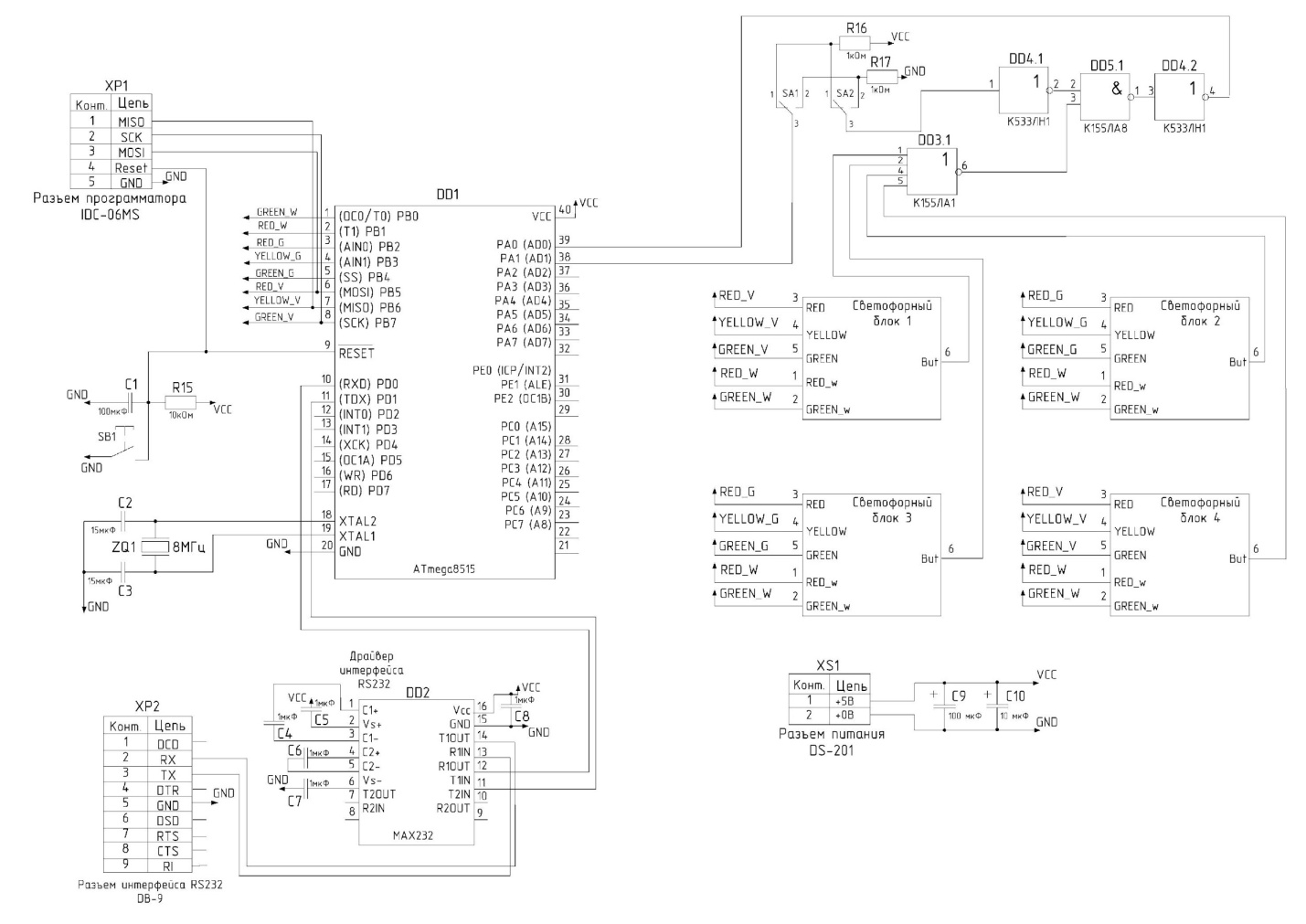


Рисунок 8 – Принципиальная схема

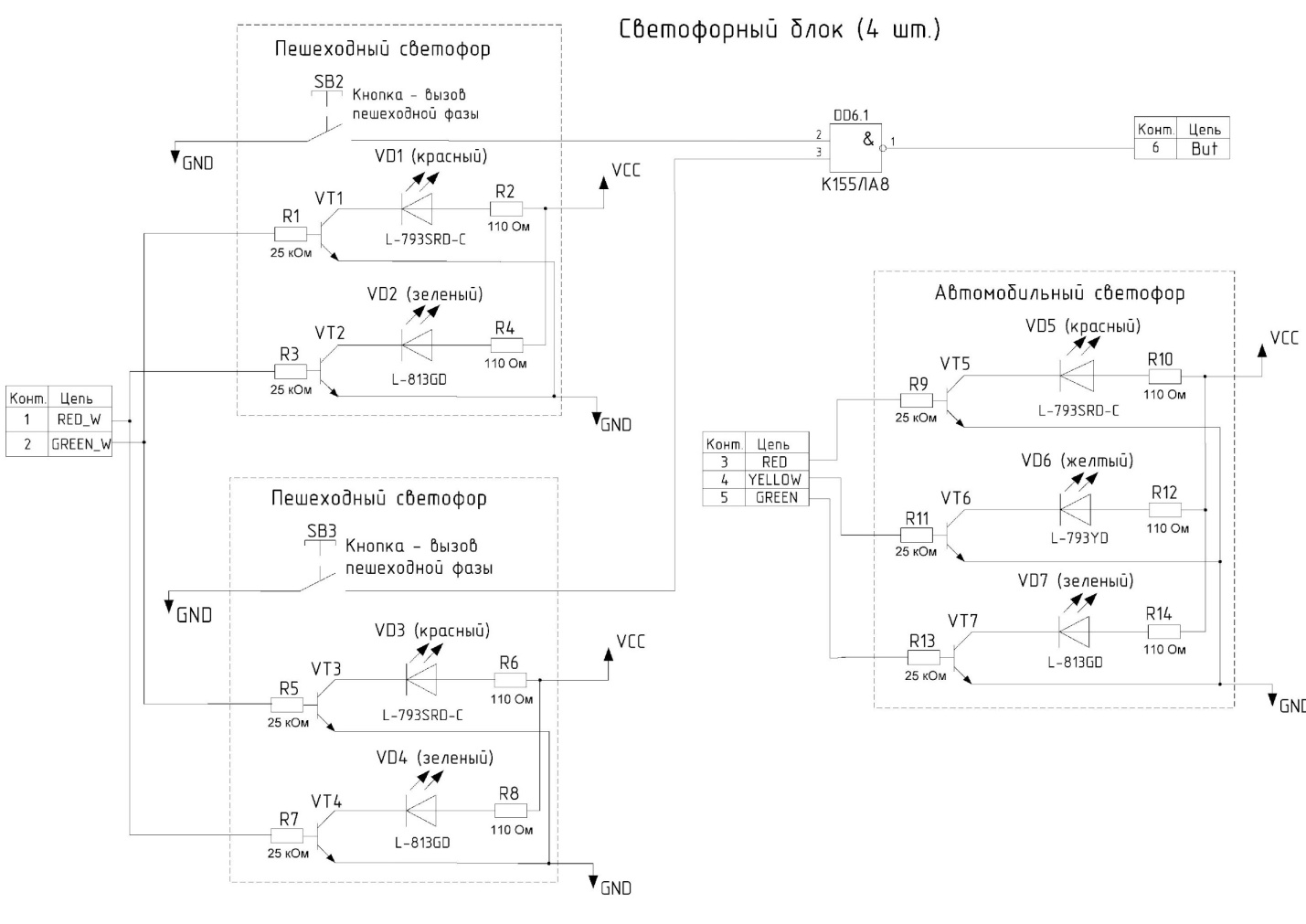


Рисунок 9 – Принципиальная схема. Светофорный блок

## 1.4 Алгоритмы работы системы

### 1.4.1 Функция Main и подпрограмма переключение светофоров

Работа начинается с функции main, из которой вызываются все остальные функции. Сначала идет вызов функции initPorts, которая инициализирует порты. Далее вызовется функция init\_time, которая обрабатывает данные, принятые по uart. После вызовется функция fix\_time, которая конвертирует цифры в числа (например, 2 и 8 в 28). После этого вызовется функция MainTimerInit, которая инициализирует системное время. Далее происходит разрешение прерываний. После вызов функции ButtonAnaliz, которая проверяет нажатие рычагов и кнопок, и вызов включения светодиодов – trafficLight. Схема алгоритма показана на рисунке 10.

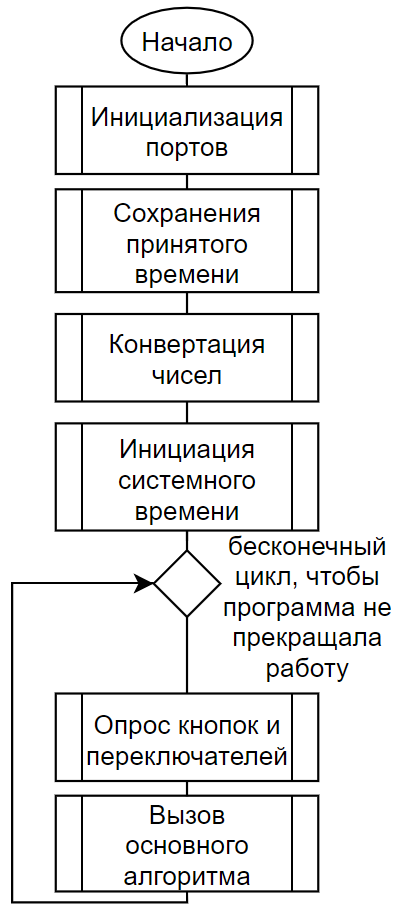


Рисунок 10 – Функция Main

Подпрограмма trafficLight, отвечает за переключение светофоров. Сначала устанавливается, с каким расписанием будет работа. После задается минимальное время для первой и второй фазы. Далее проверка нажатии вызова пешеходной фазы, и если она нажата, то определяется, на каком этапе была вызвана пешеходная фаза. Как алгоритм работает:

1. Зеленый у горизонтальной дороги и красный для остальных
   1. Если нажата кнопка вызова во время первой фазы, перейти к *пункту 8*;
2. Красный для пешеходов и желтый для остальных;
3. Зеленый для вертикальной дороги и красный для всех остальных
   1. Если нажата кнопка вызова во время второй фазы, перейти к *пункту 12*;
4. Желтый для вертикальной и красный для остальных;
5. Зеленый для пешеходов и красный для остальных;
6. Желтый для горизонтальной и красный для остальных;
7. Возвращение к *пункту 1*;
8. Желтый для горизонтальной и красный для остальных;
9. Зеленый для пешеходов и красный остальных.
10. Желтый для вертикальной и красный для остальных;
11. Перейти к *пункту 3*;
12. Желтый для вертикальной и красный для остальных;
13. Зеленый для пешеходов и красный остальных.
14. Перейти *к пункту 6.*

Во время работы каждого этапа происходит установка времени работы светодиодов, выбор портов, на которые будут подаваться данные и вызов самой задержки, которая будет вызываться и проверятся до тех пор, пока время задержки не закончится.

Чтобы параллельно работы был опрос кнопок, в коде везде можно встретить break, то есть после каждой проверки времени работа функции заканчивается и идет вызов проверки кнопок вызовом функции buttonAnaliz.

После чего обратно вызывается trafficLight и возвращается на то место, где был до этого и снова проверяет время работы текущего этапа.

Схема алгоритма показана на рисунке 11.

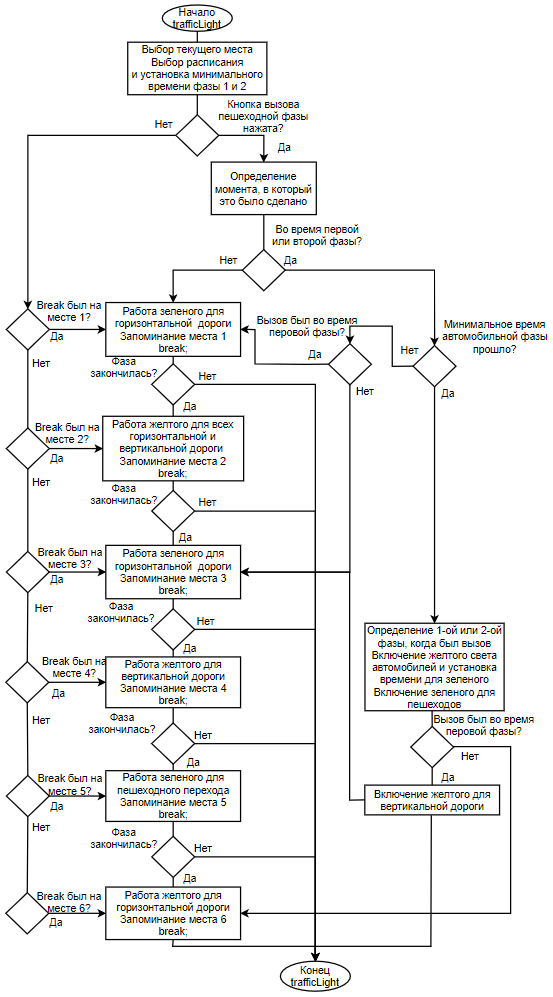


Рисунок 11 – Подпрограмма trafficLight

**1.4.2 Используемые при работе подпрограммы**

В начале работы программы, сначала инициализируются порты МК. Порт В на вывод, порт Д на ввод. Схема алгоритма показана на рисунке 12.

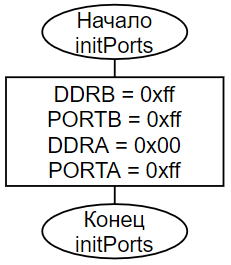


Рисунок 12 – Схема алгоритма функции initPorts

После идет чтение и сохранение передаваемой по интерфейсу UART информации. Сначала устанавливается скорость передачи и разрешения передачи данных, далее посимвольно вызывается подпрограмма чтения входных данных. После данные, принятые в кодировке ascii, переводятся в привычный 2-ый формат. Схема алгоритма показана на рисунке 13.

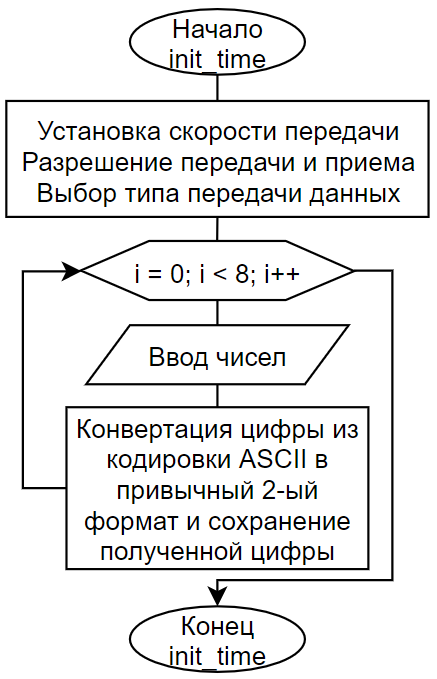


Рисунок 13 – Схема алгоритма подпрограммы init\_time

Далее полученные цифры надо преобразовать в числа, чтобы было удобно с ними работать, за это отвечает подпрограмма fix\_time. Первая цифра числа умножается на 10 и к этой цифре прибавляется вторая цифра числа, например, были введены 2 и 8, которые стали числом 28. Схема алгоритма показана на рисунке 14.

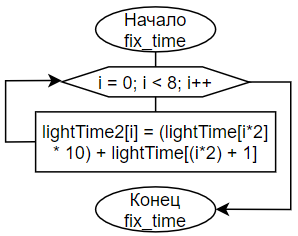


Рисунок 14 – Схема алгоритма функции fix\_time

Также во время работы алгоритма идет опрос кнопок, происходящий в подпрограмме buttonAnaliz. Идет поэтапно опрос одного рычага и одной кнопки. Если на вход PA1 подается 0, то выбирается первое расписание, иначе второе. Если на вход PA0 подаётся 0, то будет вызвана пешеходная фаза. Схема алгоритма показана на рисунке 15.

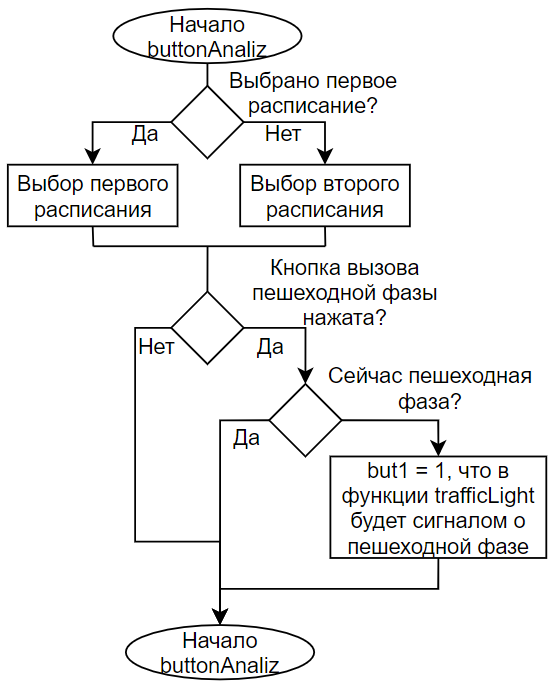


Рисунок 15 – Схема алгоритма функции buttonAnaliz

На основе составленных схем алгоритмов был написан код, который показан в Приложении А (Текст программы).

# 2 Технологическая часть

Для реализации работы модели перекрестка была написана программа на языке Си, после загруженная в МК. Симуляция проводилась в программе Proteus 7.

## 2.1 Отладка и тестирование программы

Программа была отлажена с использованием приложения Proteus 7. Это приложение предназначено для выполнения различных видов моделирования аналоговых и цифровых устройств. В ней наглядно было видно, как ведут себя светодиоды, то есть как МК выполняет заложенный в него заранее написанный алгоритм.

При написании кода были использованы следующие библиотеки:

* avr/io.h – это библиотека ввода/вывода, которая объяснит компилятору какие порты ввода/вывода есть у микроконтроллера, как они обозначены и на что они способны;
* util/delay.h – это библиотека, позволяющая вызывать задержки (delay). При вызове \_delay\_ms в качестве параметра передается время в миллисекундах. В программе используется после чтения данных, перед их выводом, чтобы МК успел обработать полученную информацию;
* stdlib.h – это стандартная библиотека C общего назначения, включающая в себя функции, занимающиеся распределением памяти, управлением процессами, преобразованием и др.;
* util/atomic.h – это библиотека, предоставляющая набор атомарных операций (то есть которые выполнятся либо полностью, либо не выполнятся вообще). Была использована функция ATOMIC\_BLOCK с параметром ATOMIC\_RESTORESTATE (запоминание состояние текущей операции – регистра SREG, и его восстановления после выполнения атомарной операции), чтобы операции выполнялись без возможности их прерывания. Использовалась функция в случаях, когда:
  + увеличивалось системное время;
  + устанавливалось время работы фазы;
  + проверка окончания работы фазы.
* stdbool.h – это библиотека, которая позволяет работать с данными типа bool;

Так же была задействована системная функция ISR – прерывание таймера. Функция вызывается, когда таймер Т0 переполняется. В коде в качестве параметра функции указывается то, с каким прерыванием будет происходить работа – TIMER0\_OVF\_vect, то есть с прерыванием по переполнению таймера T0 [10].

После компиляции создается файл с расширением “.hex”, объем которого равен 5,4 килобайта – столько занимает скомпилированная программа.

По итогу отладки и тестирования, результатом стала функционирующая модель системы перекрестка, работающая в соответствии с ТЗ. Симуляция системы описана в разделе *2.3*.

## 2.2 Настройка таймера и работа фаз светофора

МК работает при частоте 8 МГц, предделитель установлен на 1024, то есть таймер будет переполняться в 1024 раза медленнее, чем работает МК. В написанной программе таймер обновляется при переполнении. Так как максимальное число 255, в таймер Т0 загружается число 248, чтобы до обновления оставалось 8 “тиков”. И с учетом частоты МК и предделителя, таймер будет увеличиваться на ~1 тик в секунду (8 МГц / 1024 = ~8000 и 8000 / 8 = 1000 миллисекунд). Это позволяет удобно отсчитывать время работы фаз системы светофора.

В начале работы системы создаем переменную TimeMS, отвечающую за время внутри системы (в миллисекундах), и обнуляем её. Когда происходит отчет времени какой-либо из фаз, в переменную Timer присваивается текущее время системы и добавляется время работы фазы. И происходит зацикленное сравнение текущего времени системы TimeMS со временем окончания работы фазы Timer до тех пор, пока TimeMS не станет больше Timer.

## 2.3 Симуляция работы системы

Для имитации реальных условий была использована программа Proteus. Схема контроллера модели перекрестка с пешеходной регулировкой показана на рисунке 16. Работа схемы, в данном случае первая фаза – зеленый свет горит для автомобилей на светофорном блоке 2 и 3, а для всех остальных – красный, показана на рисунке 17.

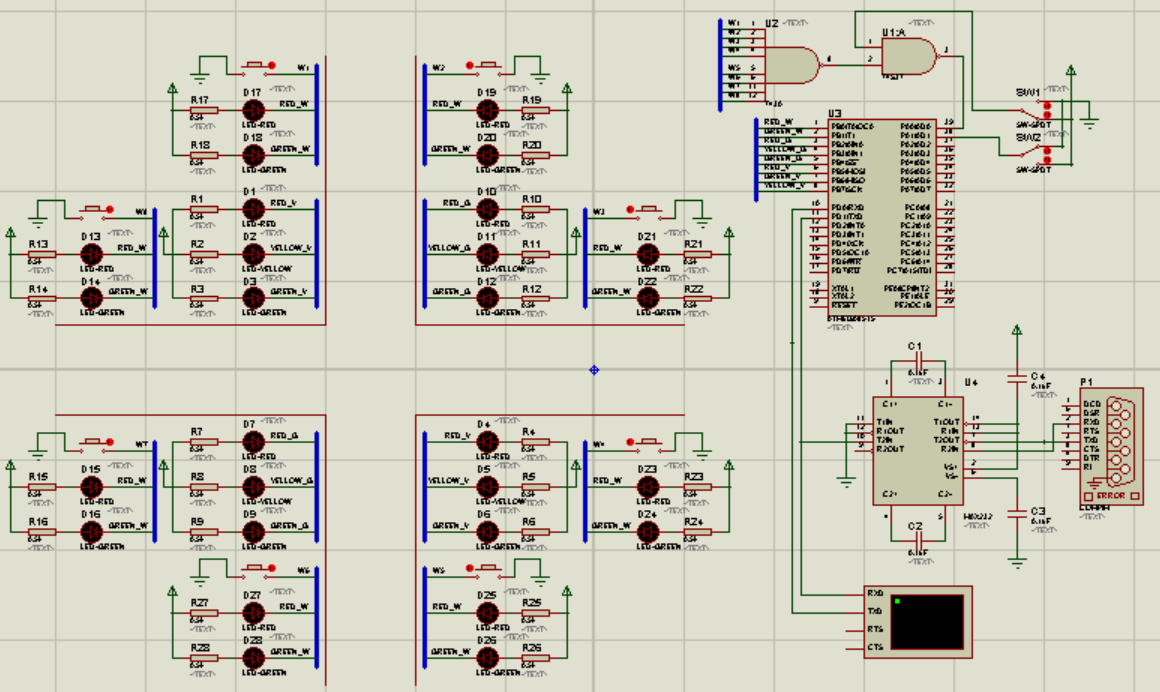


Рисунок 16 – Модель перекрестка

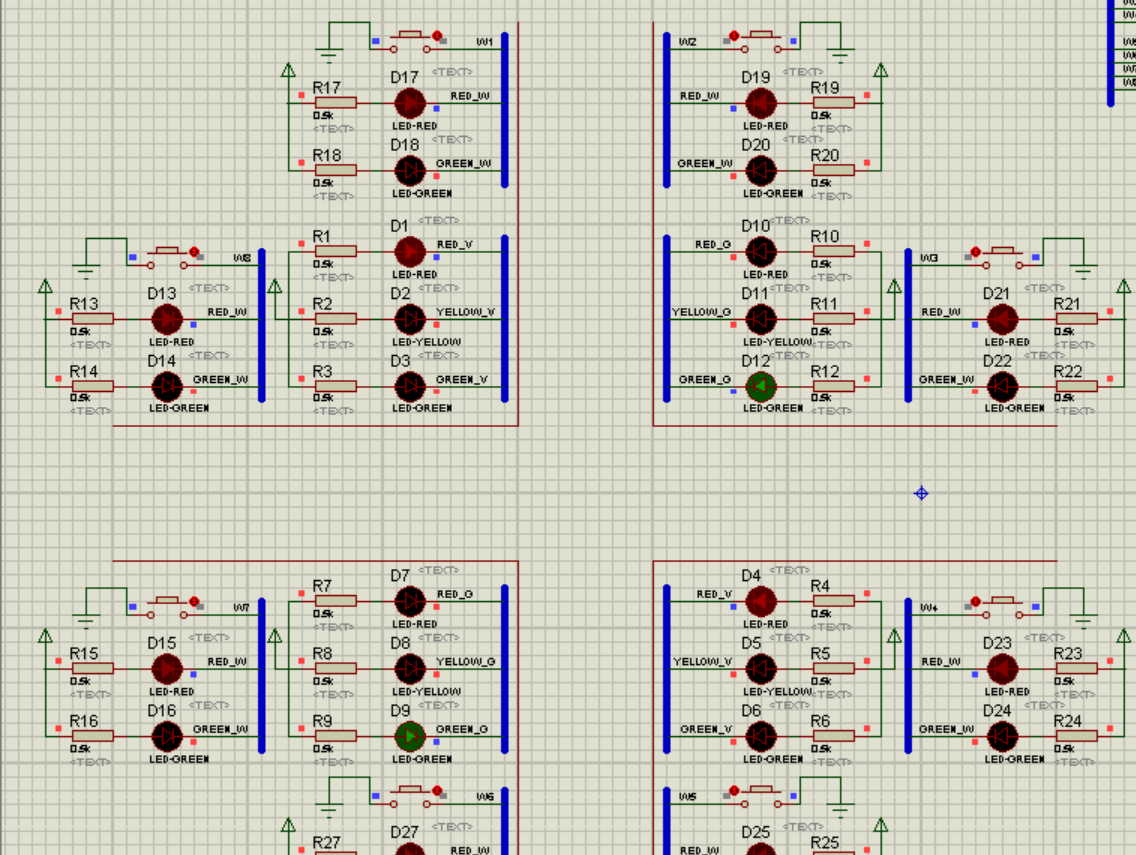


Рисунок 17 – Работа схемы (фаза 1)

Модель в proteus отличается от принципиальной схемы отсутствием транзисторов, так как в симуляции подается стабильные 5 вольт и нет необходимости в распределении поступающего от портов тока.

Для моделирования ввода данных с ПЭВМ используется инструмент системы – Virtual Terminal. Он позволяет эмулировать простейший терминал, который даёт возможность передавать и получать данные по портам RxD и TxD через интерфейс UART.

При запуске системы открывается Virtual Terminal, в который последовательно вводятся 8 чисел, как показано на рисунке 18. После каждого введенного числа, оно передается обратно, для проверки, что передача произошла правильно и выводится сообщение “-ok”, означающее, что передача закончена и можно вводить следующее число.



Рисунок 18 – Ввод чисел

Для проверки передачи снимем показатели с осциллографа, они показаны на рисунке 19 – 20. На рисунке 19 видно передачу цифры 1 в кодировке, соответствующей таблице ASCII – 00110010, которая передается первой при вводе. Ввод начинается с start бита, равного 0 и stop бита, равного 1. Диаграмма, показанная на рисунке 19, которую выдает осциллограф, показывает корректность передаваемых данных.

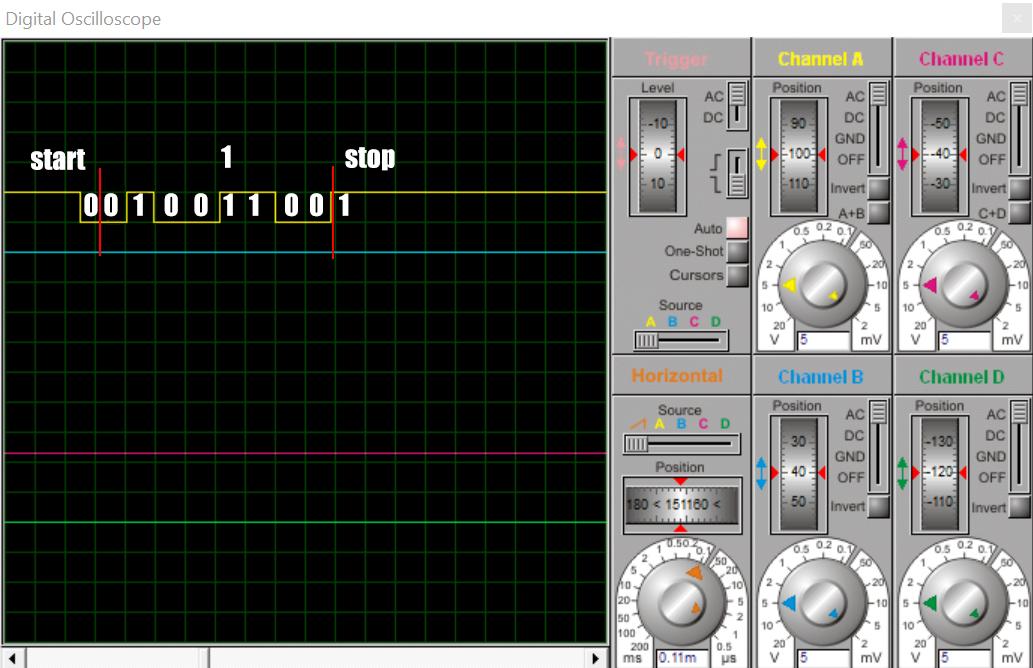


Рисунок 19 – Показатели осциллографа

На рисунке 20 видно передачу данных с МК в MAX232 – синий отрезок и данные, вышедшие из MAX232 – бордовый отрезок. Происходит передача числа 10. Сначала поступает start бит, равный 0, потом 1 – 00110010, а потом 0 – 00110000 по таблице ASCII и заканчивает передачу stop бит, равный 1. Диаграмма, показанная на рисунке 20, которую выдает осциллограф, показывает корректность передаваемых данных.

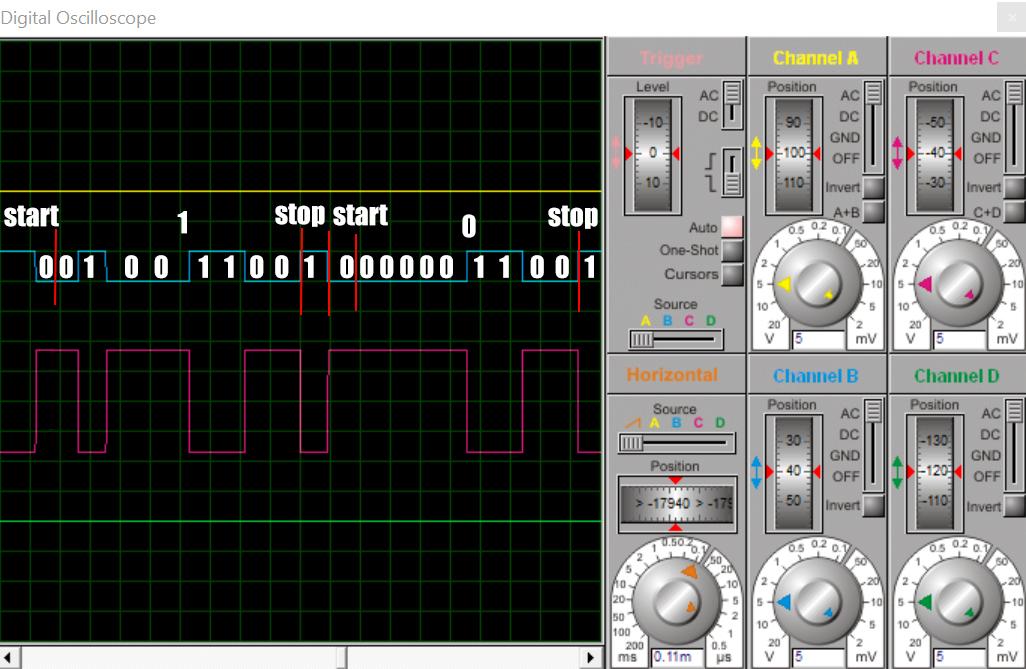


Рисунок 20 – Показатели осциллографа

RS232 основан на TTL-логике, то есть нулевому биту соответствует нулевой уровень напряжения, а единице уровень в +5 В. Стандарт RS232 использует более высокий уровень напряжения, до 15 В, и единице соответствует -15 В, а нулю +15 В. Поэтому выходной – бордовый сигнал выглядит инвертированным относительно входного синего.

## 2.4 Способы программирования МК

После написания и тестирования кода в программе, в который все – это виртуальная модель, идет этап загрузки файла (с расширением hex – бинарный файл) в микроконтроллер. Это может выполнятся следующими способами [12]:

* внутрисхемное программирование (ISP – In-System Programming);
* параллельное высоковольтное программирование;
* через JTAG;
* через Bootloader;
* Pinboard II.

Выбрана прошивка – “внутрисхемное программирование” через канал SPI, так как это простой и популярный метод, с которым уже было знакомство на практике. Программирование МК происходит через программатор и одноименный разъем, о котором было рассказано в разделе *1.3.1*. Визуальное представление показано на рисунке 21.

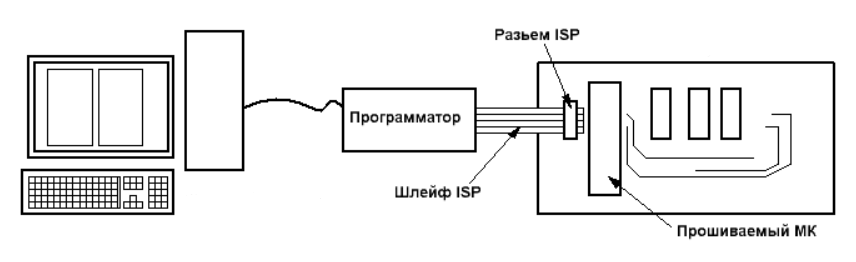


Рисунок 21 – Программирование МК

Прошивка проходит по интерфейсу SPI, для работы программатора нужно 4 контакта и питание (достаточно только земли, чтобы уравнять потенциалы земель программатора и устройства):

* MISO – Master-Input/Slave-Output – данные, идущие от контроллера;
* MOSI – Master-Output/Slave-Input – данные идущие в контроллер;
* SCK – тактовые импульсы интерфейса SPI;
* RESET – сигналом на RESET программатор вводит контроллер в режим программирования;
* GND – земля;
* VCC – питание.

Взаимодействие устройств по интерфейсу SPI требует установки одного из устройств в режим ведущего, а остальных – в режим ведомого. При этом ведущее устройство отвечает за выбор ведомого и инициализацию передачи.

Передача по SPI осуществляется в полнодуплексном режиме, по одному биту за такт в каждую сторону. По возрастающему фронту сигнала SCK ведомое устройство считывает очередной бит с линии MOSI, а по спадающему – выдает следующий бит на линию MISO.

В МК передается бинарный файл с расширением “.hex” с скомпилированной программой.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был создан проект – система светофорного перекрестка с возможностью пешеходной регулировки и двумя расписаниями работы. Система работает на основе МК серии AVR – ATmega8515. Устройство разработано в соответствии с ТЗ.

В процессе работы над курсовой работой была разработана схема электрическая функциональная и принципиальная, спецификация и документация к устройству. Исходный код программы, написанный на языке С, отлажен и протестирован при помощи симулятора Proteus 8.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Микроконтроллеры 8051, PIC, AVR и ARM: отличия и особенности [Электронный ресурс]. – Режим доступа: <http://digitrode.ru/computing-devices/mcu_cpu/1253-mikrokontrollery-8051-pic-avr-i-arm-otlichiya-i-osobennosti.html> (дата обращения: 13.09.2021)

2. ATmega8515 Datasheet [Электронный ресурс]. – Режим доступа: <https://static.chipdip.ru/lib/059/DOC000059786.pdf> (дата обращения: 15.09.2021)

3. ATmega8515, ATmega8515L 8-разрядный микроконтроллер с внутрисхемно программируемой флэш-памятью емкостью 8 кбайт [Электронный ресурс]. – Режим доступа: <http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/avr/atmega8515.htm> (дата обращения: 15.10.2021)

4. Устройство AVR микроконтроллера – Меандр – занимательная электроника [Электронный ресурс]. – Режим доступа: <https://meandr.org/archives/5146> (дата обращения: 09.10.2021)

5. MAX232x Dual EIA-232 Drivers/Receivers datasheet [Электронный ресурс]. – Режим доступа: <https://www.ti.com/lit/ds/symlink/max232.pdf> (дата обращения: 16.10.2021)

6. AVR. Учебный курс. Передача данных через UART [Электронный ресурс]. – Режим доступа: <http://easyelectronics.ru/avr-uchebnyj-kurs-peredacha-dannyx-cherez-uart.html> (дата обращения: 29.10.2021)

7. Подключение микроконтроллера. Ликбез. | Электроника для всех [Электронный ресурс]. – Режим доступа: <http://easyelectronics.ru/podklyuchenie-mikrokontrollera-likbez.html> (дата обращения: 27.09.2021)

8. ГОСТ 2.743-91 ЕСКД ОБОЗНАЧЕНИЯ БУКВЕННО-ЦИФРОВЫЕ В ЭЛЕКТРИЧЕСКИХ СХЕМАХ [Электронный ресурс]. – Режим доступа: <https://docs.cntd.ru/document/1200001985> (дата обращения: 05.10.2021)

9. ГОСТ 2.721-74 ЕСКД ОБОЗНАЧЕНИЯ УСЛОВНЫЕ ГРАФИЧЕСКИЕ В СХЕМАХ [Электронный ресурс]. – Режим доступа: <https://docs.cntd.ru/document/1200007058> (дата обращения: 05.10.2021)

10. Микроконтроллеры AVR: Параметры функции обработки прерываний ISR() в C [Электронный ресурс]. – Режим доступа: <http://avrprog.blogspot.com/2013/03/isrc.html> (дата обращения: 25.09.2021)

11. 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.

12. AVR. Учебный курс. Трактат о программаторах | Электроника для всех [Электронный ресурс]. – Режим доступа: <http://easyelectronics.ru/avr-uchebnyj-kurs-traktat-o-programmatorax.html> (дата обращения: 15.11.2021)

13. Транзисторный ключ \* diodov.net | Электроника для всех [Электронный ресурс]. – Режим доступа: <https://diodov.net/tranzistornyj-klyuch/> (дата обращения: 20.11.2021)

**Приложение А**

Текст программы

Объем исполняемого кода – 388 строк.

#define F\_CPU 8000000UL//частота работы МК

//------------------------------------------------------------

#include <avr/io.h>

#include <util/delay.h>

#include <stdlib.h>

#include <util/atomic.h>

#include <stdbool.h>

#define T\_POLL 248//для таймера, чтобы переполнялся 1000 раз в секунду

static volatile int lightTime[16];//массив, хранящий время работы светофоров расписание1

static volatile uint8\_t lightTime2[8];//массив, хранящий время работы светофоров расписание2

const unsigned char MAX\_STRING=50;//размер сообщения передаваемого по uart

char data[50];//массив для передачи сообщения по уарт

static volatile uint32\_t TimeMs = 0; //переменная для отчета системного времени

static volatile int lightRatio = 1000; //коэф умножения времени работы светофоров

unsigned char but1 = 0; //вызов пешеходного табло

unsigned char but2 = 0; //выбор расписания

ISR(TIMER0\_OVF\_vect)//прерывание при переполнении счетчика Т0

{

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE)//запрет прерываний и сохранение значений регистра SREG (и его восстановления после)

{

TCNT0 = T\_POLL;//чтобы всегда только 8 тиков до переполнения было (всего 255, 255 - 248 = 8)

TimeMs++;//системное время увеличили на 1

}

}

void MainTimerInit(void)//инициализация системного времени

{

TimeMs = 0;// Обнуляем переменную времени системы (в миллисекундах)

//инициализируем таймер, чтобы переполнялся 1000 раз в секунду

TCCR0 = 0;

TCCR0 = (0 << WGM01) | (0 << WGM00); //нормальный режим работы

TCNT0 = T\_POLL; //в счетчик кладем 248, чтобы оставалось 8 сек, так частота работы 8 МГц (типа 8000 / 1024 = 8)

TIFR |= (1 << TOV0); //генерация запроса прерывания

TIMSK |= (1 << TOIE0); //разрешение запроса прерывания

TCCR0 |= (1 << CS02) | (0 << CS01) | (1 << CS00); //предделитель 1024, то есть таймер будет переполняться в 1024 раза медленнее, чем работа МК

}

uint32\_t MainTimerSet(const uint32\_t AddTimeMs)//функция установки таймера на системном времени

{

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE)

{

return TimeMs + AddTimeMs ;//возвращает системное время + время задержки

}

}

bool MainTimerIsExpired(const uint32\_t Timer)//проверка установленного таймера, если время не прошло, то возращаем фолз

{

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE)

{

if (TimeMs >= Timer)

return true;

else

return false;

//возвращаем true если системное время стало >= времени + задержки

}

}

void trafficLight(char button\_state)

{

static uint8\_t state = 0; //для переключения между этапами в основном свиче

static uint8\_t stateBuf = 0; //для внутренней проверки, чтобы не затереть настоящий этап

static uint8\_t state1 = 3; //для переключения между этапами в свиче вызова пешеходного перехода

static uint32\_t Delay; // Переменная программного таймера расписания1

static uint8\_t koef = 1;

static uint32\_t Delay1; // Переменная для установления програамного времени

static uint8\_t status = 1; //закончилась ли работа задержки?

if (button\_state==1) //какое расписание выбрано

koef=0; //первое расписание

else

koef=1; //второе расписание

Delay1 = MainTimerSet(lightTime2[3 + (4 \* koef)] \* lightRatio);

if (but1) //если нажали на кнопку вызова зеленого у пешехода

{

switch (state1)

{

case 0:

{

stateBuf=state;

if (stateBuf==1)

{

//если вызов был во время первого этапа (зеленый для горизонтальной)

PORTB = ~0x29;//желтый для горизонтальной

state=7;

status=1;

}

if(stateBuf==2)

{

//если вызов был во время второго этапа (желтый для всех)

but1=0;

break;

}

if (stateBuf==3)

{

//если вызов был во время третьего этапа (зеленый для вертикальной)

PORTB = ~0x85; //желтый для вертикальной

state=6;

status=1;

}

if (stateBuf==4)

{

//если вызов был во время второго этапа (желтый для вертиклаьной)

but1=0;

break;

}

if(stateBuf==5)

{

//если вызов был во время второго этапа (зеленый для пешеходов)

but1=0;

break;

}

if(stateBuf==6)

{

//если вызов был во время шесторого этапа (желтый для горизонтальной)

but1=0;

break;

}

Delay = MainTimerSet(2000); //задаем время горения желтого

state1 = 1; //переключение на следующий этап светофора

break;

}

//загорание желтого и установка зеленого

case 1:

{

if ( !MainTimerIsExpired(Delay) ) break;

PORTB = ~0x26; //выставление зеленого для пешеходов

Delay = MainTimerSet(lightTime2[2 + (4\*koef)]\*lightRatio); //задаем время работы (задержку)

state1 = 3;

break;

}

case 2:

{

if ( !MainTimerIsExpired(Delay) ) break;

but1=0;

break;

}

//горение зеленого и установка минимального времени

case 3:

{

if ( !MainTimerIsExpired(Delay) ) break;

state1 = 4; //переключение на следующий этап светофора

Delay1 = MainTimerSet(lightTime2[3 + (4\*koef)]\*lightRatio); //установка минимального времени

break;

}

//включение вызова минимальной задержки

case 4:

{

if ( !MainTimerIsExpired(Delay1) ) break;

but1=0; //выключаем работу кнопки вызова пешеходной фазы

state1 = 0; //возвращение и выбор фазы

break;

}

default: break;

}

}

else//во всех случаях, когда пешедох не нажал на кнопку

{

switch (state)

{

case 0:

{

Delay = MainTimerSet(1000); //задаем время в 1 сек.

state = 1; //переключение на следующий этап светофора

break;

}

case 1:

{

//зеленого у горизонтальной

if (status==1)

{

if ( !MainTimerIsExpired(Delay) ) break;

Delay = MainTimerSet(lightTime2[0 + (4\*koef)]\*lightRatio); //задаем время работы (задержку)

PORTB = ~0x31; //выставление зеленого для горизонтальной

status=2;

but1=1; //выключаем работу кнопки вызова пешеходной фазы

}

if ( !MainTimerIsExpired(Delay) ) break; //вызов задержки

state = 2; //переключение на следующий этап светофора

status=1;

break;

}

//желтый для всех

case 2:

{

if (status==1)

{

Delay = MainTimerSet(2000); //задаем задаем время работы желтого в 2 секунды

PORTB = ~0x89; //выставление желтого для всех

status=0;

}

if ( !MainTimerIsExpired(Delay) ) break; //вызов задержки

state = 3; //переключение на следующий этап светофора

status=1;

break;

}

//зеленый для вертикальной

case 3:

{

if (status==1)

{

Delay = MainTimerSet(lightTime2[1+ (4\*koef)]\*lightRatio); //задаем время работы (задержку)

PORTB = ~0x45; //выставление зеленого для вертикальной

status=0;

but1=1; //выключаем работу кнопки вызова пешеходной фазы

}

if ( !MainTimerIsExpired(Delay) ) break; //вызов задержки

state = 4; //переключение на следующий этап светофора

status=1;

break;

}

//желтый для вертикальной

case 4:

{

if (status==1)

{

Delay = MainTimerSet(2000); //задаем задаем время работы желтого в 2 секунды

PORTB = ~0x85; //выставление желтого для вертикальной

status=0;

}

if ( !MainTimerIsExpired(Delay) ) break; //вызов задержки

state = 5; //переключение на следующий этап светофора

status=1;

break;

}

//зеленый для пешеходов

case 5:

{

if (status==1)

{

Delay = MainTimerSet(lightTime2[2 + (4\*koef)]\*lightRatio); //задаем время работы (задержку)

PORTB = ~0x26; //выставление зеленого для пешеходов

status=0;

}

if ( !MainTimerIsExpired(Delay) ) break; //вызов задержки

state = 6; //переключение на следующий этап светофора

status=1;

break;

}

//желтый для горизонтальной

case 6:

{

if (status==1)

{

Delay = MainTimerSet(2000); //задаем задаем время работы желтого в 2 секунды

PORTB = ~0x29; //выставление желтого для горизонтальной

status=0;

}

if ( !MainTimerIsExpired(Delay) ) break; //вызов задержки

state = 1; //переключение на следующий этап светофора

status=1;

break;

}

//желтый для вертикальной при вызое пешеходнйо фазы на первой фазе

case 7:

{

if (status == 1)

{

Delay = MainTimerSet(2000); //задаем задаем время работы желтого в 2 секунды

PORTB = ~0x85; //выставление желтого для вертикальной

status = 0;

}

if (!MainTimerIsExpired(Delay)) break; //вызов задержки

state = 3; //переключение на следующий этап светофора

status = 1;

break;

}

default: break;

}

}

}

int write(char\* str)//передаем по уарт

{

for(int i=0;(i<MAX\_STRING && str[i] != ':'); i++) //приваиваем данные в передадчик, пока они не закончатся, или до символа двоеточия

{

while(!(UCSRA&(1<<UDRE))) {}; //когда UDRE == 1, значит освободился и в него можно записывать новую информацию. Это нужно так

//как мы передаем по одному символу, а не по числу

UDR = str[i]; //присваивание данных в регистр приема, так же являющийс и регистром передачи

}

return 0;

}

int read(char\* str)//читаем по уарт

{

int i=0;

do {

while(!(UCSRA&(1<<RXC))) {}; // когда RXC == 1, пересылка принятого слова из сдвигового регистра приёмника в

//регистр данных UDR завершена и можно считывать данные. Это нужно так как мы передаем по одному символу, а не по числу

str[i]=UDR; //чтение данных из регистра приема

i++;

} while (str[i-1] != ':' && i <MAX\_STRING);//читаем пока не написано двоеточие или пока массив не закончится

return 0;

}

void send\_probel(unsigned char c)//отправка байта (переноса строки) по уарт

{

while(!(UCSRA&(1<<UDRE))) {} //тоже самое, что в writeSerial

UDR = c; //присваивание данных в регистр приема, так же являющийс и регистром передачи

}

int init\_time(char\* str)//функция для чтения передаваемого времени через уарт

{

UBRRL=51;//скорости передачи, расчитанная по формуле: 8000000/(16\*(9600+1))

UCSRB=(1<<TXEN)|(1<<RXEN);//TXEN = 1 - разрешение передачи, RXEN = 1 - разрешение приёма

UCSRC=(1<<URSEL)|(1<<UCSZ0)|( 1<<UCSZ1);//URSEl = 1 - адресация записи данных в регистр UCSRC, UCSZ0 = 1 и UCSZ0 = 1 - для 8 битной послыки данных

for (int i = 0; i < 8; i++)//вводим 8 значений

{

read(str);//читаем строку

\_delay\_ms(100);

write(str);// отправляем ее обратно

write(" - ok");//пишем ОК

for (int b = 0; b < 2; b++)//сохраняем цифры в lightTime

{

lightTime[b + 2\*i] = data[b]&0b00001111; //при передачи с клавы в МК символ передается в кодеровке ascii,

//и имеет первые 4 символа 0011, которые говорят, что эта цифра в кодировке ascii, а нам нужны только вторые

//4 символа, имеющие привычную двоичную форму числа. Для этого умножваем на 00001111, сохраняя нужную форму числа

}

send\_probel(13);//перенос строки, так это 0х0D в таблице ASCII, что в 10-ой будет 13

\_delay\_ms(100);

}

return 0;

}

void fix\_time(void)//конвертация чисел в привычный формат формат (из 2 и 8 получаем 28)

{

for (int i = 0; i < 8; i++)

{

lightTime2[i] = (lightTime[i\*2] \* 10) + lightTime[(i\*2) + 1];

}

}

void buttonAnaliz()//функция опроса кнопок

{

//опрос второго переключателя

if (!(PINA&0x02))//рычаг в

but2 = 1; // состоянии "1" - первое расписание

else

but2 = 0; // состоянии "0" - второе расписание

//это опрос первого переключателя

if (!(PINA&0x01))

if (but1 == 0)

but1 = 1;

}

void initPorts(void)//инициализируем порты

{

DDRB = 0xff; //инициализация на вывод

PORTB = 0xff; //заполняем 1-ами

DDRA = 0x00; //инициализация на ввод

PORTA = 0xff; //заполняем 1-ами

}

int main(void)//главная функция

{

initPorts();//инициализируем порты

init\_time(data);//вводим время из ПЭВМ (терминала)

fix\_time();//конвертим это время

MainTimerInit();//инициализируем системное время

sei();//разрешаем прерывания

while (1) //залетаем в бесконечный цикл, в котором

{

buttonAnaliz();//опрашиваем переключатели

trafficLight(but2);//светофор работает

}

}

**Приложение Б**

Графическая часть

На 2 листах

Электрическая схема функциональная

Электрическая схема принципиальная

**Приложение В**

Перечень элементов

На 3 листах