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:

Контроллер модели перекрестка с пешеходной регулировкой

Студент	ИУ6-74Б		С.П. Пантелеев
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Руководитель			Б.И. Бычков
		(Подпись, дата)	(И.О. Фамилия)

2021 г.

ЛИСТ ЗАДАНИЯ

РЕФЕРАТ

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

МИКРОКОНТРОЛЛЕР, СИСТЕМА, ПЕРЕКРЕСТОК, СВЕТОФОР

Объектом разработки является контроллер модели перекрестка с пешеходной регулировкой.

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

Поставленная цель достигается посредством использования Proteus 7.

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.

Содержание

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК – микроконтроллер.

ТЗ – техническое задание.

Proteus 7 — пакет программ для автоматизированного проектирования (САПР) электронных схем.

ASCII – таблица кодировки символов.

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

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

Введение

В данной работе производится разработка контроллера модели перекрестка с пешеходной регулировкой.

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 и 4 блоков светофоров, каждый из которых включает в себя 1 светофор для автомобилей и 2 для пешеходов. На каждом пешеходном светофоре имеется кнопка для вызова пешеходной фазы вне очереди. Также можно в МК загружать два расписания и имеется возможность переключения между ними.

Актуальность разрабатываемой модели перекрестка, приближенной по функциям к реальности, заключается в том, что светофоры используются повсеместно на дорогах, и без них не было бы регулировки автомобильного трафика. Не все системы светофоров на перекрестках имеют кнопок вызова пешеходных фаз, что учитывается в разработанном устройстве. Также в МК загружается два расписания работы светофоров, что может использоваться как, например, дневное расписание и ночное.

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 регулировка системы светофоров на дорожном перекрестке. Как автомобильных, так и пешеходных.

Система светофоров имеет 3 фазы – автомобильная для горизонтальной дороги, автомобильная для вертикальной дороги и пешеходная. Их работа состоит из 6 действий, которые зацикливаются. Цикл описан в таблице 1.

Таблица 1 – Работа перекрестка

Этап работы	Описание
1-ая фаза	Зеленый для горизонтальной и, соответственно красный для вертикальной дороги и пешеходов
Переключение	Желтый для горизонтальной и вертикальной дороги и красный для пешеходов
2-ая фаза	Зелёный для вертикальной и, соответственно красный для горизонтальной дороги и пешеходов
Переключение	Желтый для вертикальной и красный для горизонтальной дороги и пешеходов
3-ья фаза	Зеленый для пешеходов и красный для горизонтальной и вертикальной дороги
Переключение	Желтый для горизонтальной и, соответственно красный для вертикальной дороги и пешеходов

После выполнения последнего действия цикл возвращается к первому и начинает все сначала.

Также пешеход может вызвать внеочередную пешеходную фазу, нажав на кнопку на пешеходном светофоре, если минимальное время для автомобилей прошло. Возможно два таких случая:

- вызов пешеходной фазы во время первой автомобильной фазы. После минимального времени первой фазы будет внеочередная пешеходная, а после неё вторая автомобильная фаза;
- вызов пешеходной фазы во время второй автомобильной фазы. После минимального времени второй автомобильной фазы будет внеочередная пешеходная, а после неё первая автомобильная фаза.

Система имеет два расписания работы, между которыми можно переключаться. Расписание – это время работы каждой из фаз (2 автомобильных и 1 пешеходная) и минимальное время работы автомобильной фазы.

Время каждой из фаз и минимальное время работы фазы задается вручную, вводом 8 чисел, описание которых показано в таблице 2.

Таблица 2 – Вводимые числа

Порядок ввода	Описание
1-ое	Время (в секундах) работы первой фазы первого расписания
2-ое	Время (в секундах) работы второй фазы первого расписания
3-ее	Время (в секундах) работы третьей фазы первого расписания
4-ое	Минимальное время (в секундах) работы первой и второй фаз первого расписания
5-ое	Время (в секундах) работы первой фазы второго расписания
6-ое	Время (в секундах) работы второй фазы второго расписания

Продолжение таблицы 2

Порядок ввода	Описание
7-ое	Время (в секундах) работы третьей фазы второго расписания
8-ое	Минимальное время (в секундах) работы первой и второй фаз второго расписания

Разработанная структурная схема модели перекрестка представлена на рисунке 1.

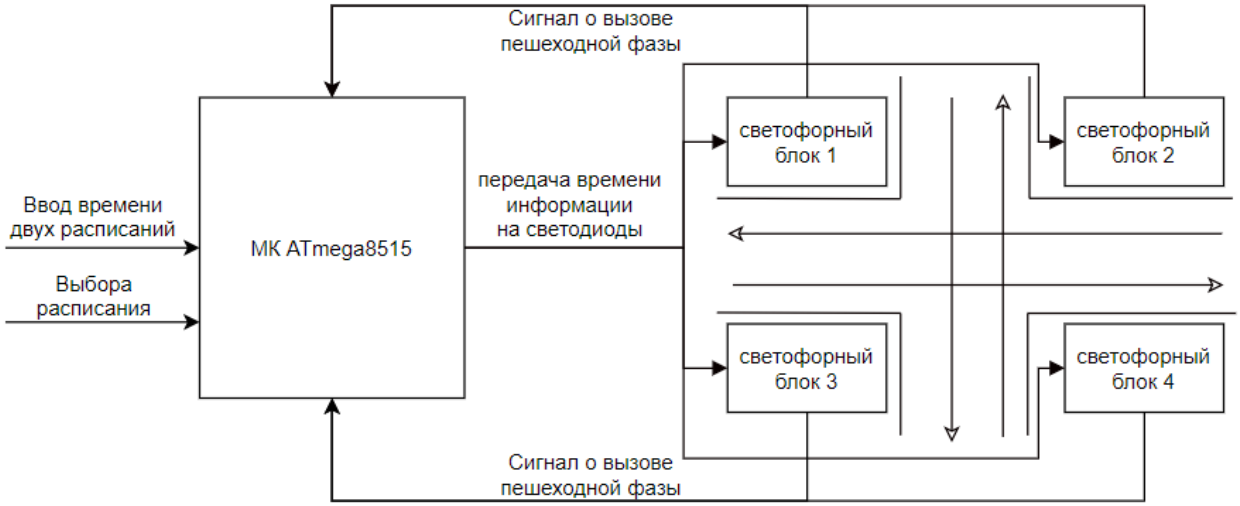


Рисунок 1 – Структурная схема устройства

Первый и четвертый светофорный блок регулируют движение вертикальной дороги, а второй и третий – горизонтальной.

1.2 Проектирование функциональной схемы

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

1.2.1 Микроконтроллер ATmega8515

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, разработанное компанией 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
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 192 Кб

Было выбрано семейство AVR, так как в модели перекрестка важна скорость выполнения операций при переключении света для регулирования

трафика, что могут предоставить МК серии AVR, а также они имеют больший объем памяти по сравнению с семейством ARM. А также с МК серии AVR уже был опыт работы, что упростит разработку системы.

AVR в свою очередь делятся на семейства:

1. TinyAVR, имеющие следующие характеристики:
   - Flash-память до 16 Кбайт;
   - RAM до 512 байт;
   - ROM до 512 байт;
   - число пинов (ножек) ввода-вывода 4–18;
   - небольшой набор периферии.
2. MegaAVR, имеющие следующие характеристики:
   - FLASH до 256 Кбайт;
   - RAM до 16 Кбайт;
   - ROM до 4 Кбайт;
   - число пинов ввода-вывода 23–86;
   - расширенная система команд (ассемблер и C) и периферии.
3. XMEGA AVR, имеющие следующие характеристики:
   - FLASH до 384 Кбайт;
   - RAM до 32 Кбайт;
   - ROM до 4 Кбайт;
   - четырехканальный контроллер DMA (для быстрой работы с памятью и вводом/выводом).

Выберем подсемейство MegaAVR, так как оно имеет больше пинов и объем памяти, чем TinyAVR, а также поддерживает C. XMEGA AVR не был выбран так как они лучше по характеристикам, чем необходимо, то есть весь их потенциал не будет раскрыт.

В подсемействе MegaAVR семейства AVR был выбран МК – ATmega8515, обладающий всем необходимым функционалом для реализации проекта:

- интерфейс SPI для программирования МК;

- интерфейс UART для обмена данными;
- 512 байт ОЗУ;
- 1 8-разрядный счетчик;
- 8 Кбайт FLASH памяти;
- 3 внешних источника прерываний;
- частота работы до 16 Мгц;

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

Это экономичный 8-разрядный микроконтроллер, основанный на AVR RISC архитектуре. ATmega8515 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].

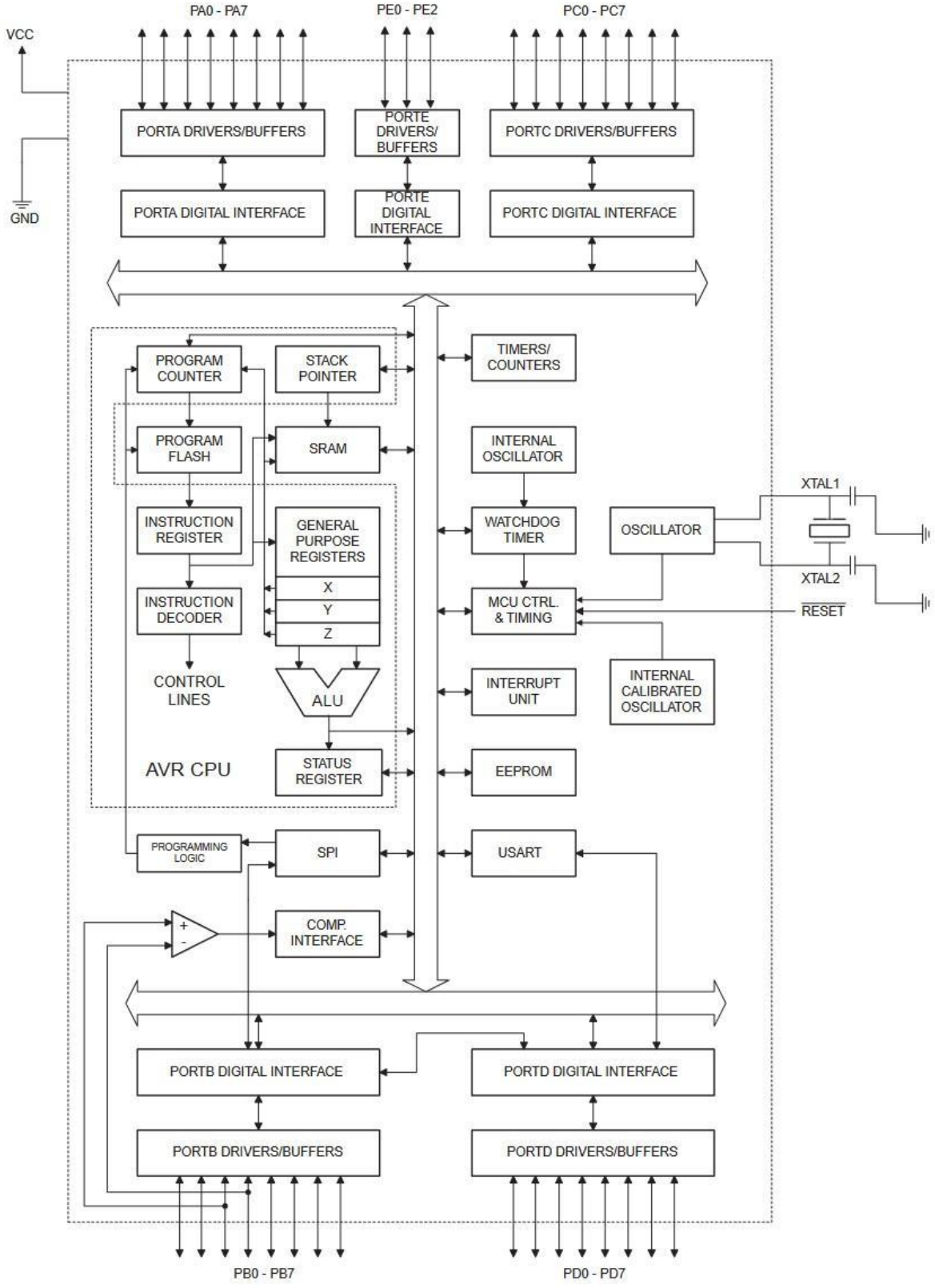


Рисунок 2 – Структурная схема МК ATmega8515

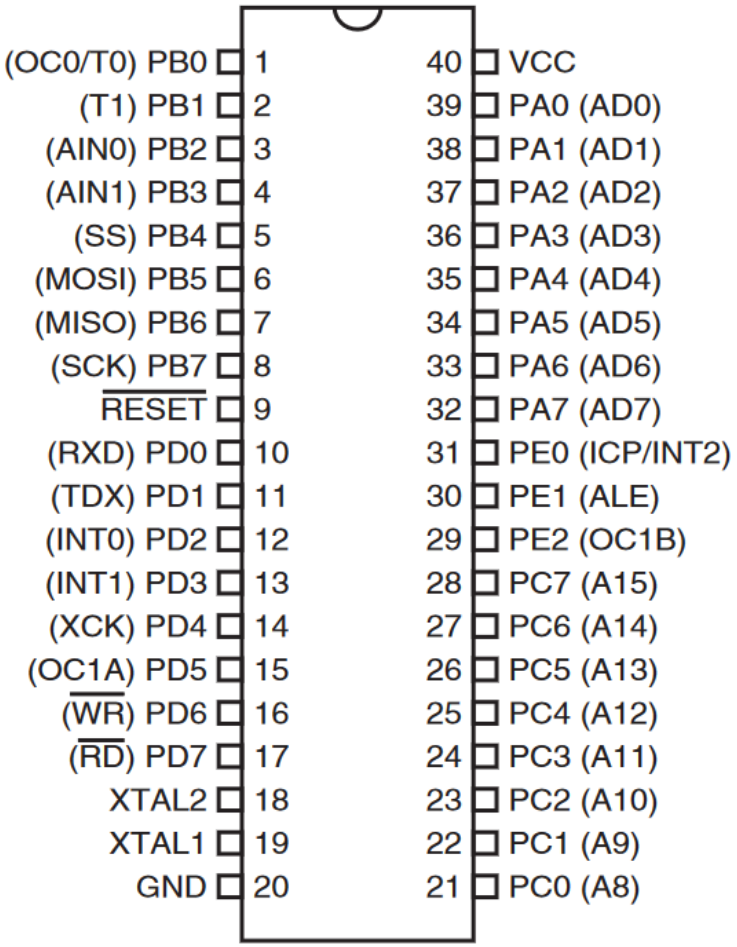


Рисунок 3 – УГО МК ATmega8515

Он обладает следующими характеристиками [3]:

1. RISC архитектура:
   - мощная система команд с 130 инструкциями, большинство из которых выполняются за один машинный цикл;
   - 32 восьмиразрядных рабочих регистров общего назначения;
   - производительность до 16 миллиона операций в секунду при частоте 16 МГц;
   - встроенное умножение за 2 цикла.
2. Энергонезависимые память программ и данных:
   - 8 Кб внутрисхемно-программируемой flash-памяти с возможностью самозаписи;
   - возможность внутрисхемного программирования программой во встроенном секторе начальной загрузки;
   - возможность считывания во время записи;

- 512 байт ЭППЗУ (EEPROM);
- 512 байт внутреннего статического ОЗУ;
- возможность организации внешней области памяти размером до 64 Кб;
- программирование битов защиты программного обеспечения.

3. Периферийные устройства:
   - один 8-разрядный таймер-счетчик с отдельным предделителем и режимом компаратора;
   - один 16-разрядный таймер-счетчик с отдельным предделителем и режимом компаратора;
   - три канала ШИМ (широтно-импульсная модуляция);
   - программируемый последовательный UART (устройство синхронной или асинхронной приемопередачи);
   - последовательный интерфейс SPI с режимами главный и подчиненный;
   - программируемый сторожевой таймер с отдельным встроенным генератором;
   - встроенный аналоговый компаратор.
4. Специальные функции микроконтроллера:
   - сброс при подаче питания и программируемый супервизор питания;
   - встроенный калиброванный RC-генератор;
   - внутренние и внешние источники запросов на прерывание;
   - три режима управления энергопотреблением: холостой ход (Idle), пониженное потребление (Power-down) и дежурный (Standby).
5. Ввод-вывод:
   - 35 программируемых линий ввода-вывода;
   - 47 регистров ввода/вывода.
6. Напряжение питания: 4.5 – 5.5В.
7. Рабочая частота: 1 – 16 МГц.

1.2.1.1 Используемые элементы

Для функционирования системы светофоров в МК ATmega8515 задействованы не все элементы его архитектуры. Выделим и опишем те, что используются во время функционирования схемы [4].

Порты A, B, D – 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 только для прошивки МК.

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 8-разрядные таймеры T0, T2 и 16-разрядный таймер T1. Во время работы данной программы используется только T0.

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 используется прерывание при переполнении счетчика.

UART – через этот интерфейс в МК передается информация из ПЭВМ. В регистр UART информация попадает через порт PD0 (RxD).

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

1.2.1.2 Распределение портов

МК ATmega8515 содержит пять портов – A, B, C, D и E. Опишем назначение тех, что используются в данной системе для её функционирования.

Порт A:

- PA0 – в него поступает сигнал о внеочередном вызове пешеходной фазы. Если поступает логический 0, то происходит вызов пешеходной фазы;
- PA1 – в него поступает сигнал, говорящий о том, какое расписание было выбрано, то есть по каким таймингам будет работать светофор. Если поступает логический 0, то выбирается первое расписание, иначе, то есть, когда логическая 1 – второе.

Порт B:

- PB0 – включение зеленого сигнала пешеходных светофоров (GREEN_W);

- PB1 – включение красного сигнала пешеходных светофоров (RED_W);
- PB2 – вывод информации на автомобильные светодиоды зеленого цвета для горизонтальной дороги (GREEN_G);
- PB3 – вывод информации на автомобильные светодиоды желтого цвета для горизонтальной дороги (YELLOW_G);
- PB4 – вывод информации на автомобильные светодиоды красного цвета для горизонтальной дороги (RED_G);
- PB5 – вывод информации на автомобильные светодиоды зеленого цвета для вертикальной дороги (GREEN_V);
- PB6 – вывод информации на автомобильные светодиоды желтого цвета для вертикальной дороги (YELLOW_V);
- PB7 – вывод информации на автомобильные светодиоды красного цвета для вертикальной дороги. (RED_V).

Порт D:

- PD0 – отвечает за вывод вводимой информации для проверки корректности ввода;
- PD1 – отвечает за принятие вводимой информации (время работы светофоров для автомобилей, пешеходов и минимальное время работы первой и второй фазы).

1.2.1.3 Организация памяти

Схема организации памяти МК ATmega8515 показана на рисунке 4.

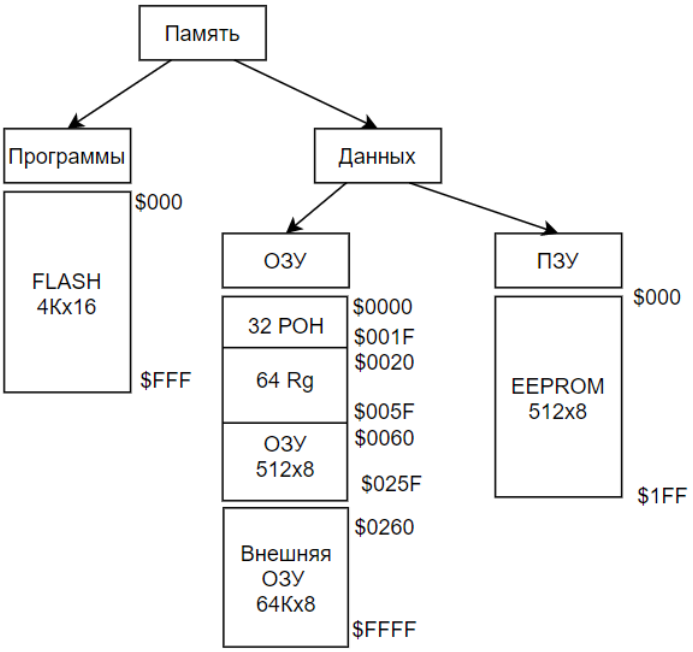


Рисунок 4 – Организация памяти МК ATmega8515

Память программы предназначена для хранения последовательности команд, управляющих функционированием микроконтроллера, и имеет 16-ти битную организацию.

Все AVR имеют Flash-память программ, в выбранном МК её объем 4Кх16, то есть длина команды 16 разрядов. Информацию в flash-память можно мгновенно стереть и записать. Заносится с помощью программатора.

Память данных делиться на три части:

1. Регистровая память – 32 регистра общего назначения и служебные регистры ввода/вывода.
2. Оперативная память (ОЗУ) – МК ATmega8515 имеет объем внутреннего SRAM 512 байт (с адреса $0060 до $025F). Число циклов чтения и записи в RAM не ограничено, но при отключении питания вся информация теряется. Для МК ATmega8515 возможна организация подключения внешнего статического ОЗУ объемом до 64К.
3. Энергонезависимая память (EEPROM) – эта память доступна МК в ходе выполнения, для хранения промежуточных результатов. В

МК ATmega8515 ее объем 512 байт. Также в неё могут быть загружены данные через программатор.

1.2.2 Пульт оператора

Пульт оператора представляет из себя 2 переключателя. Первый (SA1) отвечает за выбор расписания (их вводится два), МК работает по первому расписанию, если переключатель замкнут на логическом 0, и по второму, если на логической 1. Второй (SA2) отвечает за работу кнопок вызова пешеходной фазы на светофорах, если переключатель замкнут на логическом 0, то кнопки не работают, если на логической 1, то работают.

1.2.3 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 – XP2.

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

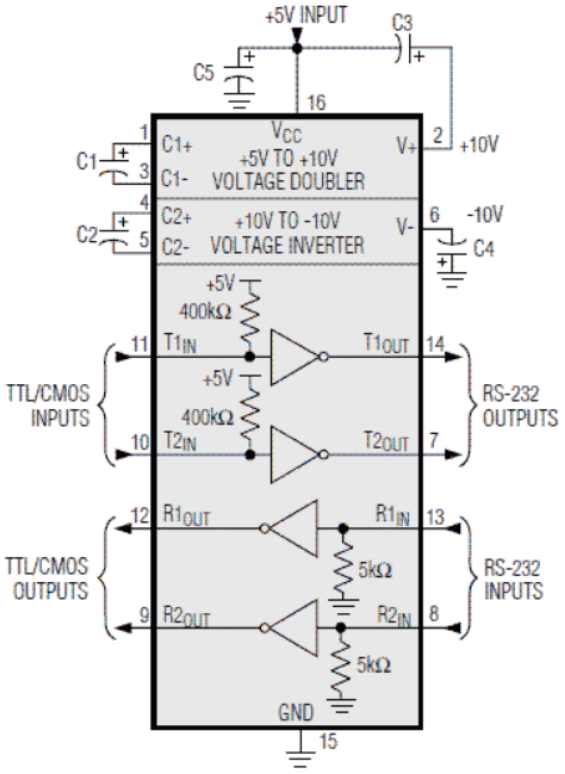


Рисунок 5 – Преобразователь MAX232

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

Продолжение таблицы 4

Номер	Имя	Тип	Описание
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.

1.2.4 Настройка канала передачи

Для передачи информации в МК используется последовательный интерфейс UART, для корректной работы необходимо настроить регистры управления UCSRA, UCSRB и UCSRC [6, 4].

UCSRA. Биты регистра UCSRA показаны в таблице 5.

Таблица 5 – биты регистра UCSRA

Номер	7	6	5	4	3	2	1	0
Название	RXC	TxC	UDRE	FE	DOR	PE	U2X	MPCM
Доступ	R	R/W	R	R	R	R	R	R

Во время работы системы используются только биты RXC и UDRE [6].

RXC – флаг завершения приема. Устанавливается в 1 при наличии не прочитанных данных в буфере приемника. Используется при чтении данных из MAX232. Когда RXC == 1, то в регистре UDR есть данные, их можно из него считать.

UDRE – флаг опустошения регистра данных. Устанавливается в 1 при пустом буфере передатчика. Используется при выводе данных из МК. Когда USRE == 1, то есть регистр UDR пуст, в него можно загружать новые данные для передачи.

UCSRB. Биты регистра UCSRB показаны в таблице 6.

Таблица 6 – биты регистра UCSRB

Номер	7	6	5	4	3	2	1	0
Название	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ	RXB8	TXB8
Доступ	R	R/W	R/W	R/W	R/W	R/W	R	R/W

Во время работы системы, для управления приемом и передачей информации, будут использоваться только биты TXEN и RXEN [6].

TXEN – разрешение передачи. Если разряд сбросится во время передачи, выключение передатчика произойдет только после завершения передачи.

RXEN – при установке в 1 разрешается работа приемника. При сбросе разряда работа приемника запрещается, буфер сбрасывается.

UCSRC. Биты регистра UCSRC показаны в таблице 7.

Таблица 7 – биты регистра UCSRC

Номер	7	6	5	4	3	2	1	0
Название	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Доступ	R	R/W	R/W	R/W	R/W	R/W	R	R/W

Во время работы системы будут использоваться только биты URSEL, UCSZ0 и UCSZ1 [6].

URSEL – регистры UCSRC и UBRRH находятся в одном адресном пространстве, и чтобы понять, куда записывать данные используется бит URSEL. При URSEL равным 1 в UCSRC, при 0 в UBRRH.

UCSZ0 и UCSZ1 – формат посылки. Каждый из них принимает значение 1 – для 8 битной посылки данных.

Скорость передачи определяется выражением [4]:

$$BAUD = \frac{f_{osc}}{16(UBRR+1)}$$

где BAUD — скорость передачи (бод);

f_{osc} — тактовая частота микроконтроллера (Гц);

UBRR — содержимое регистров UBRR0H, UBRR0L контроллера скорости передачи.

Зададим скорость 9600 бод. Получится:

$$UBRR = \frac{f_{osc}}{16*BAUD} - 1 = \frac{8*10^6}{16*9600} - 1 = 51_{10}$$

UBRRL будет принимать значение 51.

1.2.5 Светофорный блок

Светофорный блок представляет из себя один автомобильный и два пешеходных светофора. Автомобильный светофор имеет красный (стоп для машин), желтый (предупреждающий о включении зеленого или красного света) и зеленый (разрешающий движение) цвета светодиодов. Пешеходный светофор имеет красный (стоп для пешеходов) и зеленый (можно идти) цвет светодиодов.

Пешеходные светофоры оснащены кнопками вызова пешеходной фазы вне очереди. Её вызов может произойти после того, как пройдет минимальное время горения зеленого света у автомобильного светофора.

Информация (1 – горит, 0 - выключен) подается с МК в соответствии с разработанным и загруженным алгоритмом.

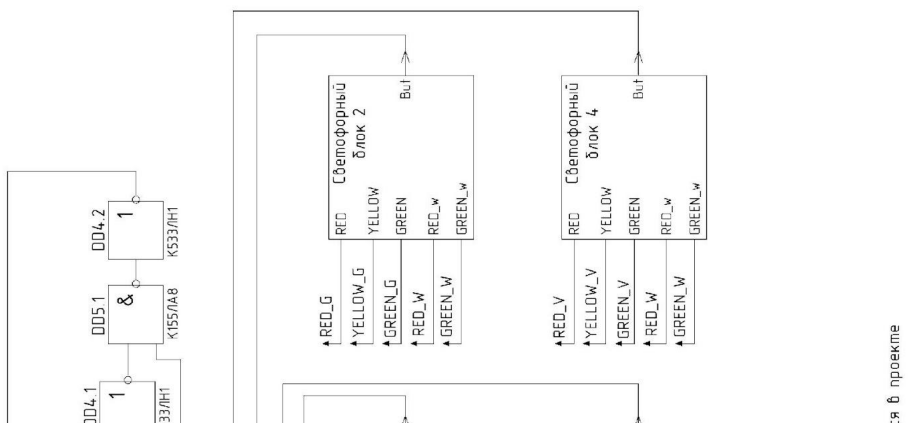
1.2.6 Генератор тактовых импульсов и сброс

Для работы МК необходим тактовый генератор. Внутри МК есть собственный, но для надежности подключим внешний резонатор. Он подключается к внутреннему через входы XTAL1 и XTAL2 [7]. Частота резонатора равна 8 МГц для соответствующей тактовой частоты МК.

В AVR есть внутренняя схема сброса, и сигнал RESET изнутри уже подтянут резистором 100кОм к Vcc. Чтобы сброс мог происходить вручную, подключим выход RESET к питанию, через резистор в 10кОм и конденсатор в 100 мкф [7]. При включении схемы конденсатор разряжен и напряжение на RESET близко к нулю – МК не стартует. Но со временем, через резистор, конденсатор зарядится и напряжение на RESET достигнет логической единицы и МК запустится.

1.2.7 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].



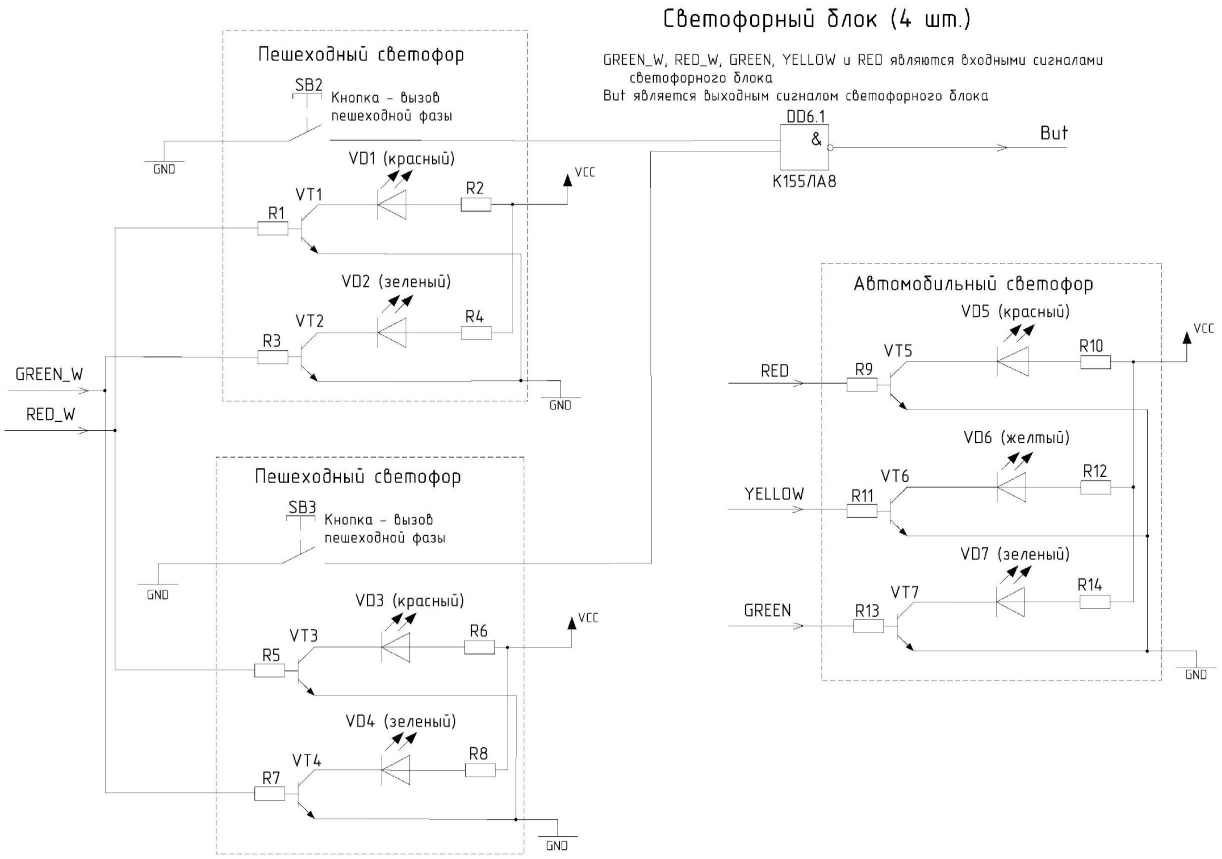


Рисунок 7 – Функциональная схема светофорного блока

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 PB. На принципиальной схеме, которая показана в разделе *1.3.5*, условное обозначение – XP1.

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 – XS1.

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.

$$R_K = \frac{U_R}{I} = \frac{2{,}1}{0{,}02} = 105 \text{ Ом}$$

где U_R – 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:

$$U_R = U_K - \Delta U_{VD} - \Delta U_K = 5 - 2{,}5 - 0{,}4 = 2{,}1 \text{ В}$$

где U_K – напряжение питания;

ΔU_{VD} – напряжение светодиода;

ΔU_K – падение напряжение на переходе коллектор-эмиттер.

Ток, протекающий через светодиод, снизился:

$$I_к = \frac{U}{R} = \frac{2{,}1}{110} = 0{,}019 \text{ А} = 19 \text{ мА}$$

Для определения резистора со стороны базы, посчитаем ток базы.

$$I = \frac{I_к}{\beta} = \frac{0{,}019}{200} = 0{,}000095 \text{ мкА}$$

где $I_к$ – ток коллектора, равный 0,019А;

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

$$R_Б = \frac{U}{I} = \frac{2{,}1}{0{,}000095} = 22105 \text{ Ом} = 22{,}1 \text{ кОм}$$

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

$$P = U * I$$

где U – напряжение питания (В);

I – ток потребления микросхемы (мА).

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

$$P = I^2 * R$$

где R – сопротивление резистора;

I – ток, проходящий через резистор.

Для транзисторов по формуле:

$$P = U * I$$

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	Количество 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 MO-100 с номиналом 110 и 25 кОм соответственно, 2 резистора CF-25 с номиналом 1 кОм, и 28 транзисторов 2N2222.

$P_{суммарная} = P_{ATmega8515} + P_{MAX232} + P_{К155ЛА1} + P_{К533ЛН1} + P_{К155ЛА8} + P_{L\text{-}793SRD\text{-}C} + P_{L\text{-}813GD} + P_{L\text{-}793YD} + P_{резисторов} + P_{транзисторов} = 200 + 40 + 20 + 15 + 200 + 564 + 564 + 188 + 56,5 + 42 = 2189,5$ 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].

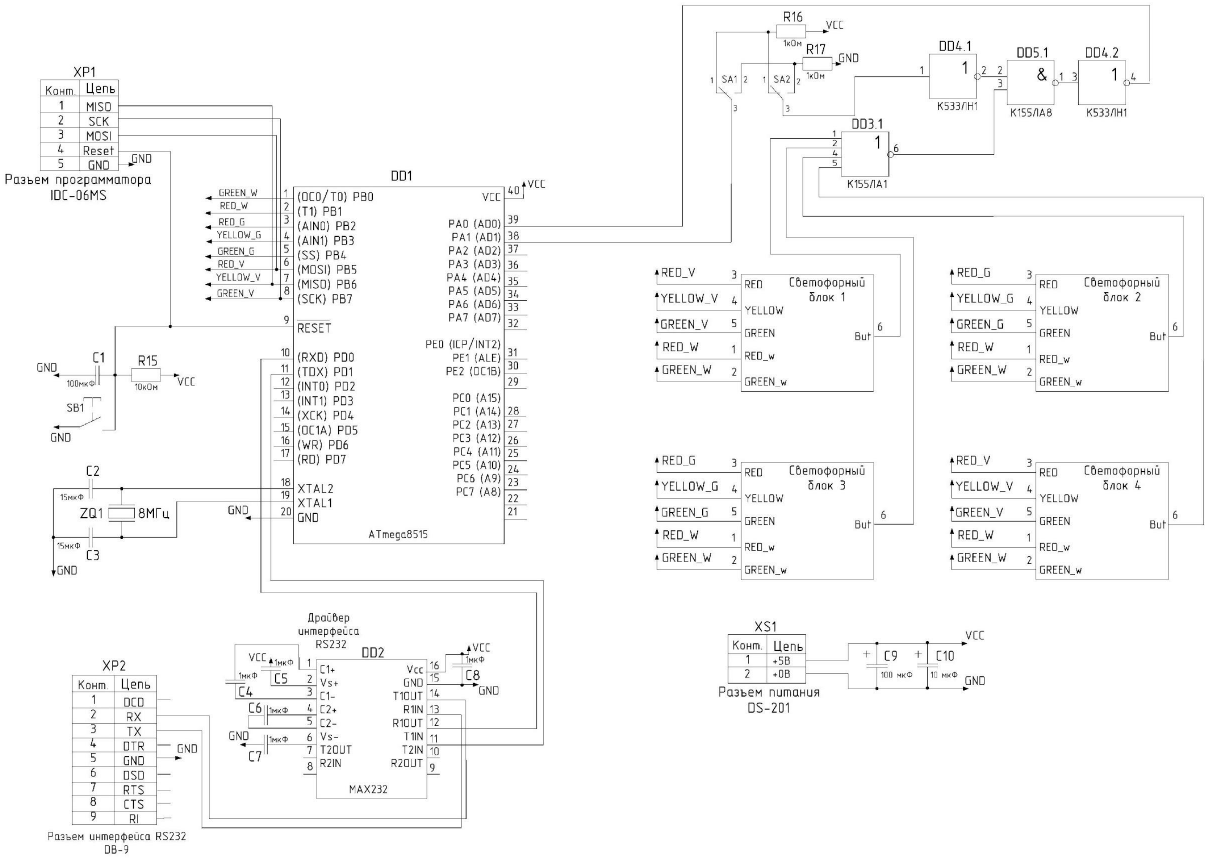


Рисунок 8 – Принципиальная схема

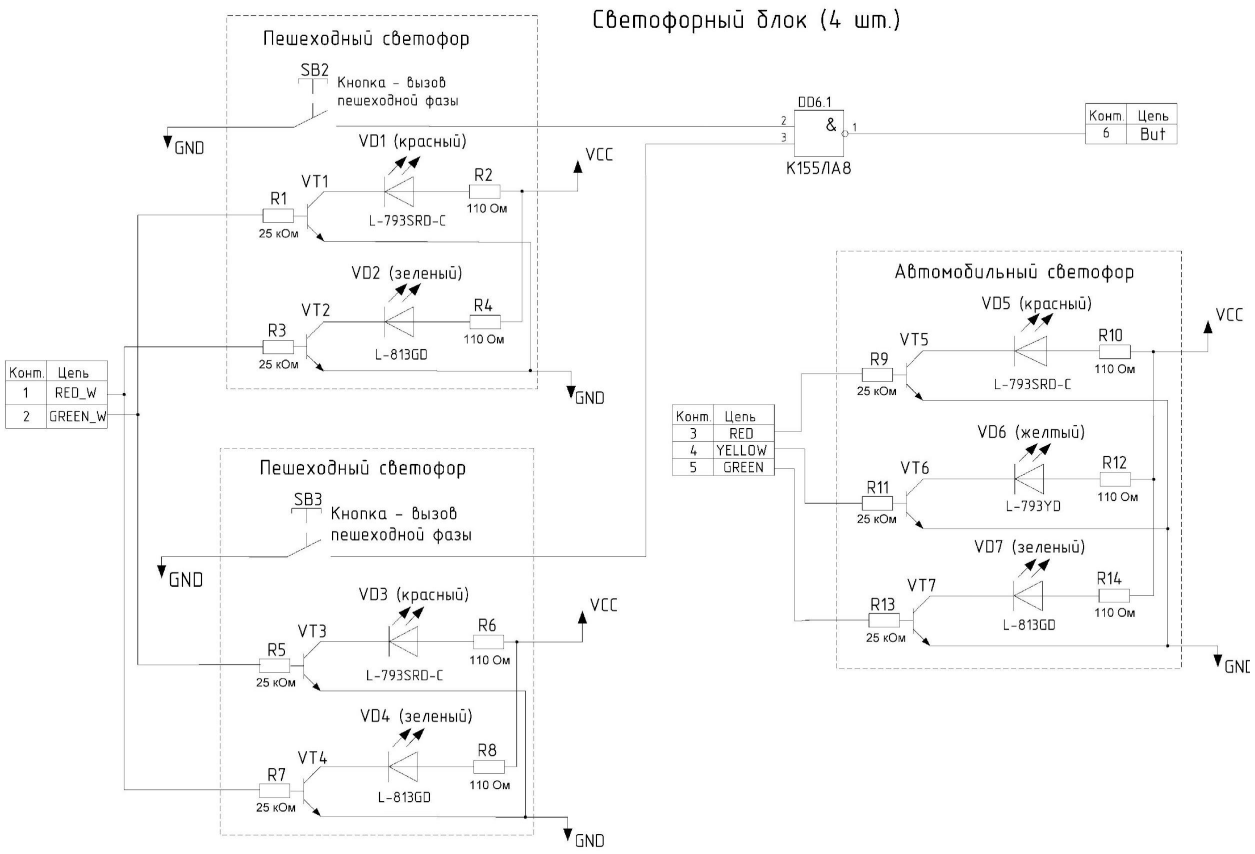


Рисунок 9 – Принципиальная схема. Светофорный блок

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.

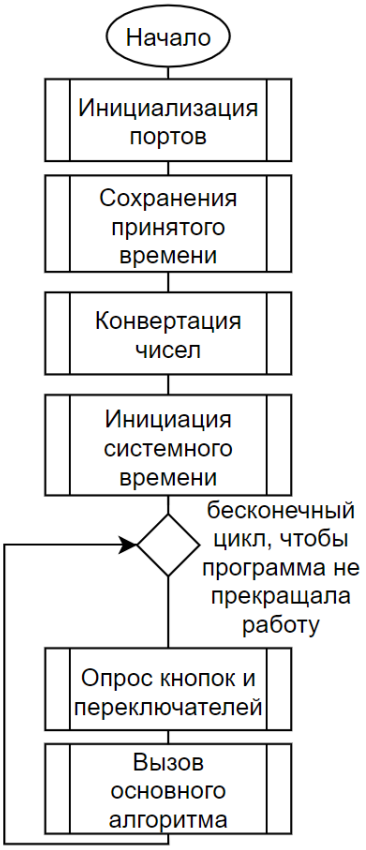


Рисунок 10 – Функция Main

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
    a. 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
    a. 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.

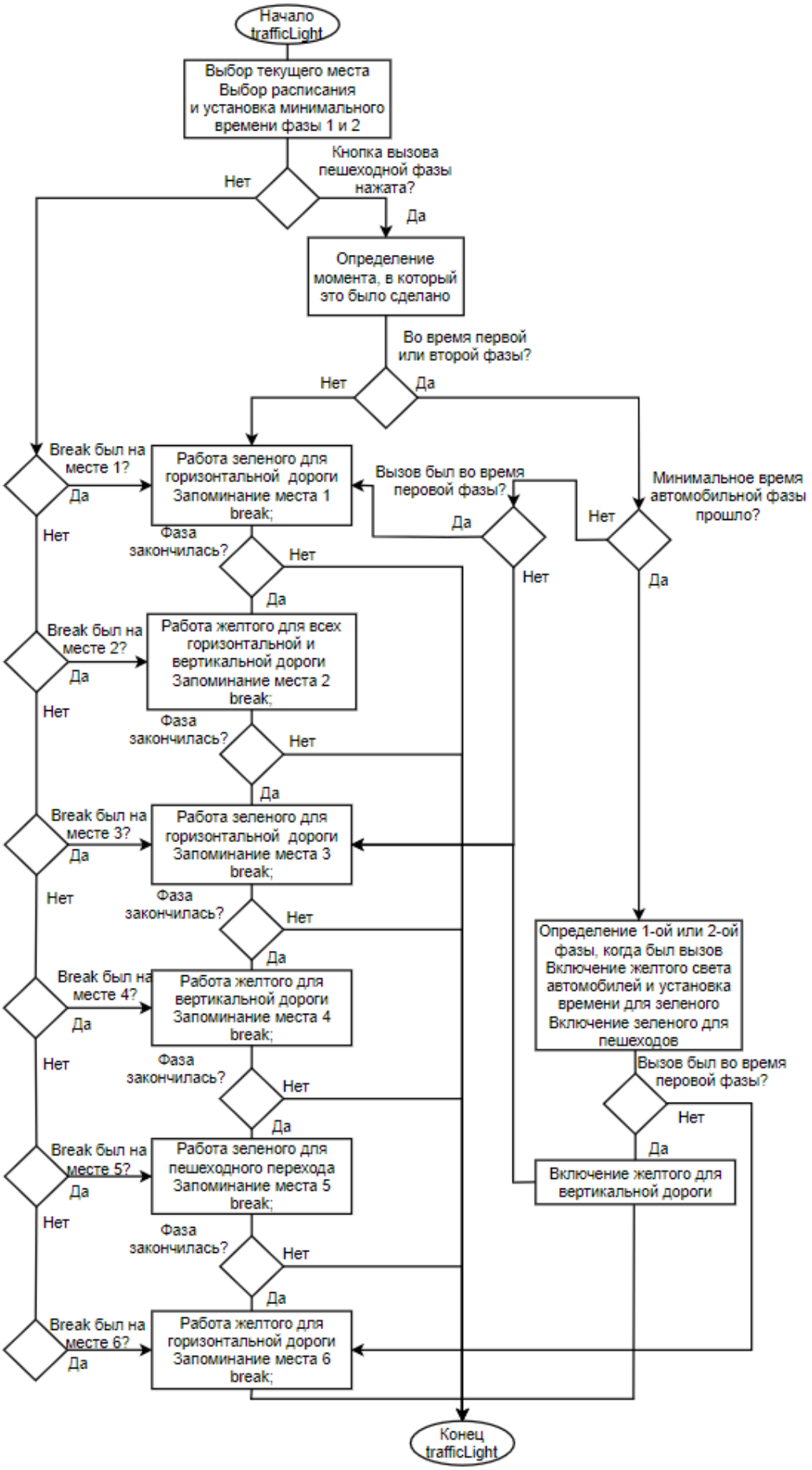

Начало trafficLight
Выбор текущего места
Выбор расписания
и установка минимального
времени фазы 1 и 2
Кнопка вызова
пешеходной фазы
нажата?
Нет
Да
Определение
момента, в который
это было сделано
Во время первой
или второй фазы?
Нет
Да
Break был на
месте 1?
Да
Нет
Работа зеленого для
горизонтальной дороги
Запоминание места 1
break;
Фаза
закончилась?
Нет
Да
Вызов был во время
перовой фазы?
Да
Нет
Минимальное время
автомобильной фазы
прошло?
Нет
Да
Break был на
месте 2?
Да
Нет
Работа желтого для всех
горизонтальной и
вертикальной дороги
Запоминание места 2
break;
Фаза
закончилась?
Нет
Да
Break был на
месте 3?
Да
Нет
Работа зеленого для
горизонтальной дороги
Запоминание места 3
break;
Фаза
закончилась?
Нет
Да
Break был на
месте 4?
Да
Нет
Работа желтого для
вертикальной дороги
Запоминание места 4
break;
Фаза
закончилась?
Нет
Да
Break был на
месте 5?
Да
Нет
Работа зеленого для
пешеходного перехода
Запоминание места 5
break;
Фаза
закончилась?
Нет
Да
Break был на
месте 6?
Да
Работа желтого для
горизонтальной дороги
Запоминание места 6
break;
Определение 1-ой или 2-ой
фазы, когда был вызов
Включение желтого света
автомобилей и установка
времени для зеленого
Включение зеленого для
пешеходов
Вызов был во время
перовой фазы?
Нет
Да
Включение желтого для
вертикальной дороги
Конец trafficLight

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.

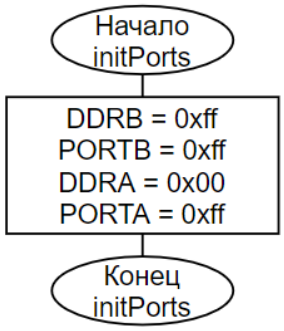


Рисунок 12 – Схема алгоритма функции initPorts

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.

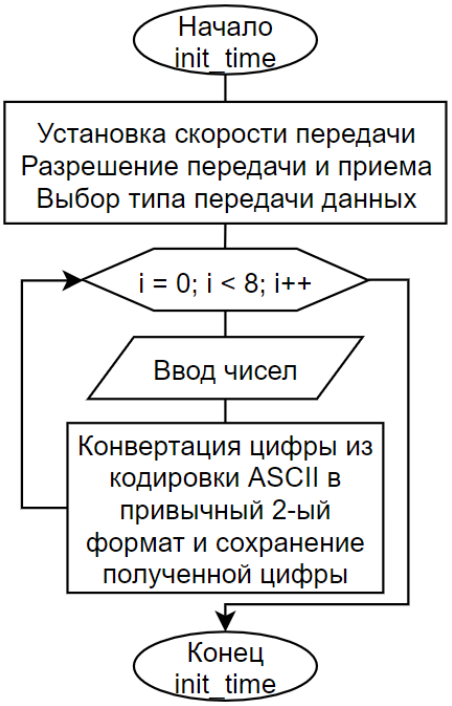


Рисунок 13 – Схема алгоритма подпрограммы init_time

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.

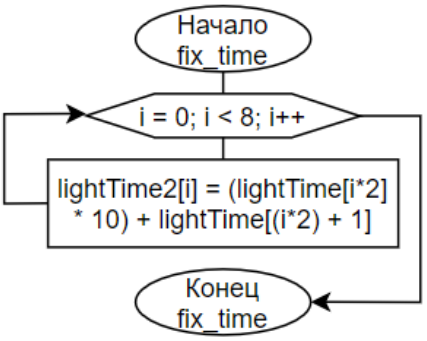


Рисунок 14 – Схема алгоритма функции fix_time

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.

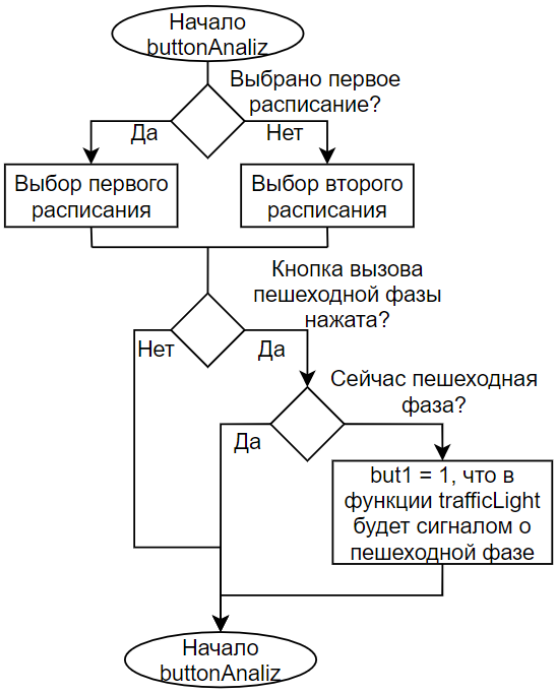


Рисунок 15 – Схема алгоритма функции buttonAnaliz

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 С 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 T0 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 T0 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.

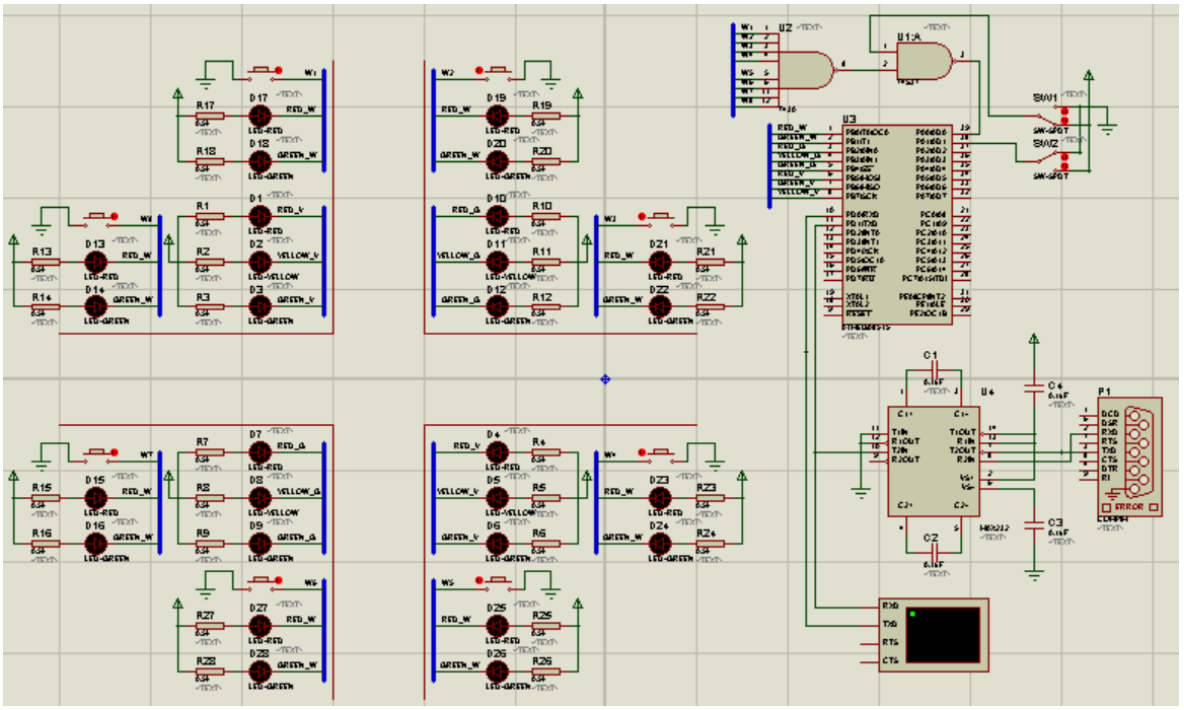


Рисунок 16 – Модель перекрестка

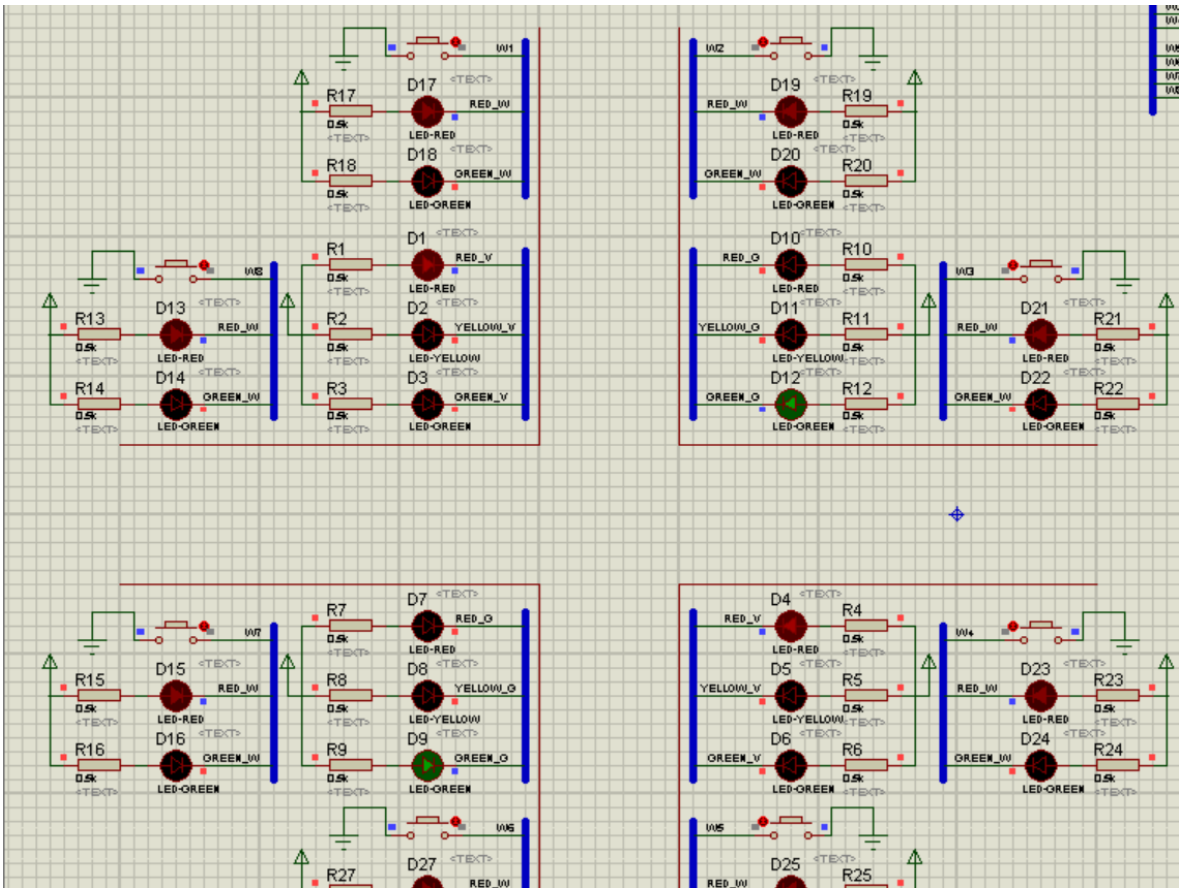


Рисунок 17 – Работа схемы (фаза 1)

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.

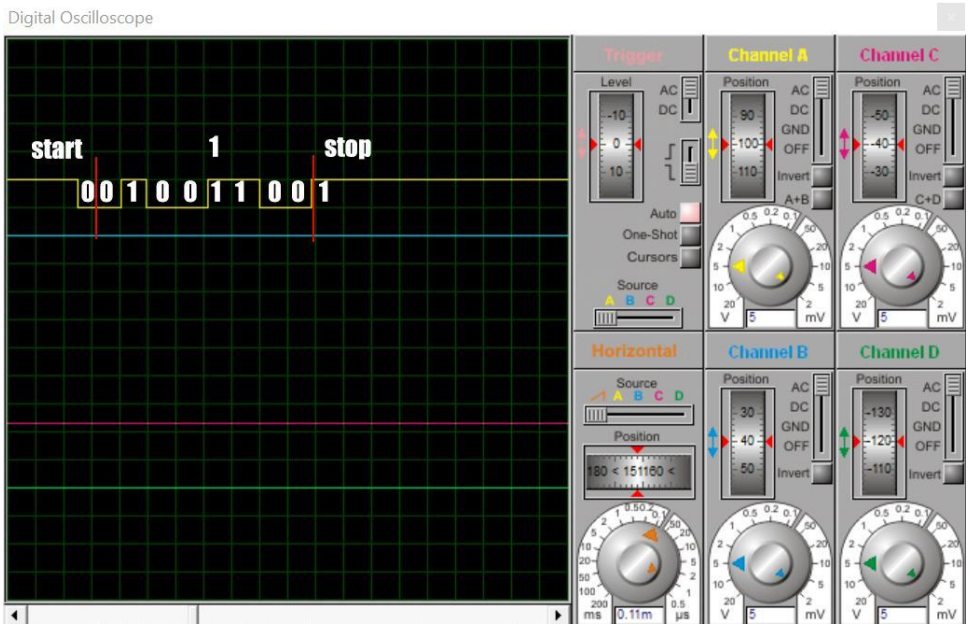


Рисунок 19 – Показатели осциллографа

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.

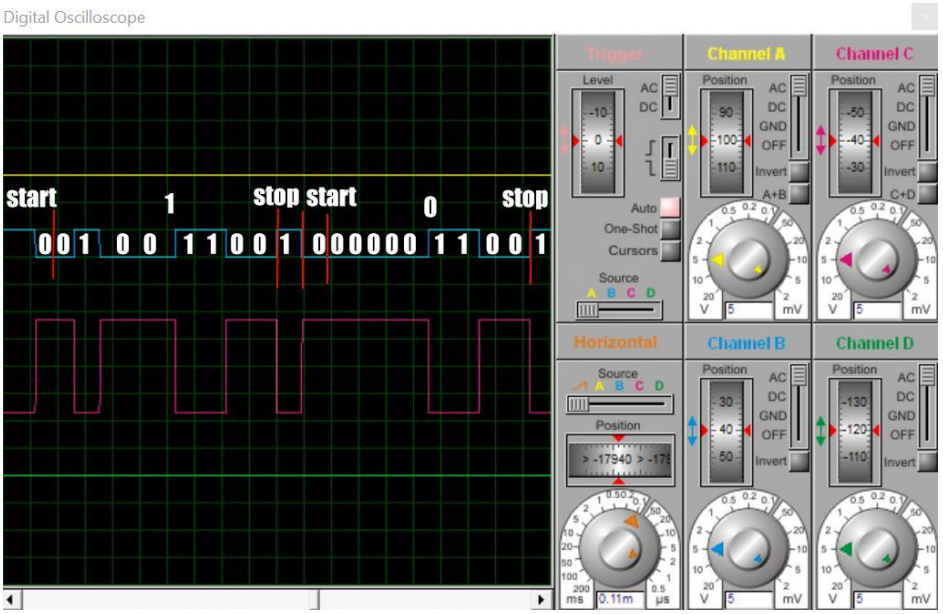


Рисунок 20 – Показатели осциллографа

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.

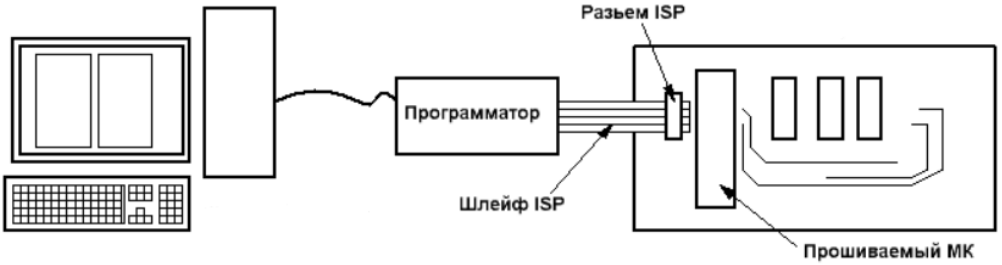


Рисунок 21 – Программирование МК

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 7.

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 T0
{
     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
праграамного 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 OK
          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 0x0D 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