|  |  |
| --- | --- |
| 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 09.03.01 **Информатика и вычислительная техника**

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***к курсовой работе***

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

***на тему:***

**Индикатор станций маршрута в метрополитене**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-72Б |  |  | С.А. Шишкин |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Руководитель |  |  |  | С.В. Ибрагимов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |

*2022 г.*

**РЕФЕРАТ**

Расчетно-пояснительная записка 37 с., 16 рис., 3 табл., 10 источников, 2 прил.

МИКРОКОНТРОЛЛЕР, ПРОГРАММИРУЕМОЕ УСТРОЙСВО, ТАЙМЕР, ATMEGA16, USART.

Объектом разработки данной курсовой работы является проектирование индикатора станций маршрута в метрополитене.

Цель работы – закрепление знаний, полученных при изучении дисциплины «Микропроцессорные системы», в процессе самостоятельной работы при проектировании индикатора; развитие навыков и умений применять теоретические знания на практике при выполнении учебных проектов, а также по заказам промышленности и в порядке личной инициативы; освоение новых технологий проектирования при выполнении проектных работ.

В процессе выполнения курсовой работы были решены следующие задачи: анализ задания, выбор схемотехнического решения и элементов МК-системы, анализ и выбор радиоэлементов схемы, расчет потребляемой мощности устройства, разработка алгоритмов управления и соответствующей программы микроконтроллера.

В результате был спроектирован требуемый индикатор и получена сопутствующая документация, а именно: функциональная и принципиальная схемы, схемы алгоритмов управления и соответствующая программа микроконтроллера.

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

ТЗ – техническое задание

МК – микроконтроллер

УГО – условное графическое обозначение

Atmega16 – используемый микроконтроллер

USART – (англ. Universal Synchronous receiver/transmitter) Универсальный синхронный приемопередатчик

ПЭВМ – персональная электронно-вычислительная машина

Datasheet – документация на устройство

Proteus ISIS – среда моделирования

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc122304274)

[1 Конструкторская часть 6](#_Toc122304275)

[1.1 Анализ требований 6](#_Toc122304276)

[1.2 Синтез и описание функциональной электрической схемы 7](#_Toc122304277)

[1.2.1 Описание микроконтроллера Atmega16 7](#_Toc122304278)

[1.2.2 Организация памяти микроконтроллера Atmega16 10](#_Toc122304279)

[1.3 Синтез и описание электрической принципиальной схемы 12](#_Toc122304280)

[1.3.1 Подключение программатора и питания 12](#_Toc122304281)

[1.3.2 Электронное табло 12](#_Toc122304282)

[1.3.3 Блок связи с ПЭВМ 13](#_Toc122304283)

[1.3.4 Настройка USART 14](#_Toc122304284)

[1.3.5 Внешняя энергонезависимая память EEPROM 15](#_Toc122304285)

[1.3.6 Расчет потребляемой мощности 16](#_Toc122304286)

[1.4 Описание алгоритмов основных программных модулей 17](#_Toc122304287)

[1.4.1 Главная процедура программы 17](#_Toc122304288)

[1.4.2 Процедура обновления дисплея 18](#_Toc122304289)

[2 Технологическая часть 19](#_Toc122304290)

[2.1 Характеристика использованных систем для разработки и отладки программ 19](#_Toc122304291)

[2.2 Тестирование устройства в симуляторе Proteus 19](#_Toc122304292)

[2.3 Программирование микроконтроллера 22](#_Toc122304293)

[ЗАКЛЮЧЕНИЕ 27](#_Toc122304294)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc122304295)

[ПРИЛОЖЕНИЕ А. Текст исходной программы 29](#_Toc122304296)

[ПРИЛОЖЕНИЕ Б. Спецификация радиоэлементов схемы 36](#_Toc122304297)

**ВВЕДЕНИЕ**

Курсовая работа «Индикатор станций маршрута в метрополитене» выполнялась на основании учебного плана кафедры ИУ6.

Цель данной курсовой работы – проектирование устройства для индикации маршрута в метрополитене, которое включает в себя электронное табло и EEPROM.

Проектирование контроллера с заданными функциями состоит из двух основных частей: конструкторская часть и технологическая часть.

Конструкторская часть включает в себя:

* проектирование структурно-функциональной схемы;
* описание архитектуры, используемого микроконтроллера и описание назначения функциональных элементов схемы;
* описание принципиальной электрической схемы МК-системы с обоснованием выбора используемых радиоэлементов;
* описание алгоритмов функционирования МК-системы;
* расчет потребляемой мощности устройства.

Технологическая часть включает в себя:

* характеристику использованных систем разработки и отладки программ;
* тестирование и отладку программы;
* описание и моделирование работы системы;
* описание способа программирования МК.

По завершении проектирования была выполнена проверка работоспособности схемы и программного обеспечения.

Конструкторская часть

Анализ требований

Согласно техническому заданию, необходимо разработать устройство, которое включает в себя контроллер, электронное табло и EEPROM. Табло отображает следующую станцию. Переключение станций на табло происходит по нажатию кнопки. Устройство должно позволять перезаписывать маршрут любым способом.

Основанием для выполнения данной работы являются:

* учебный план кафедры ИУ6;
* задание на курсовой проект.

Для реализации данного функционала устройство должно содержать следующие структурные блоки:

* микроконтроллер;
* кнопка;
* EEPROM;
* электронное табло;
* блок связи с ПЭВМ.

Структурная схема устройства представлена на рисунке 1.

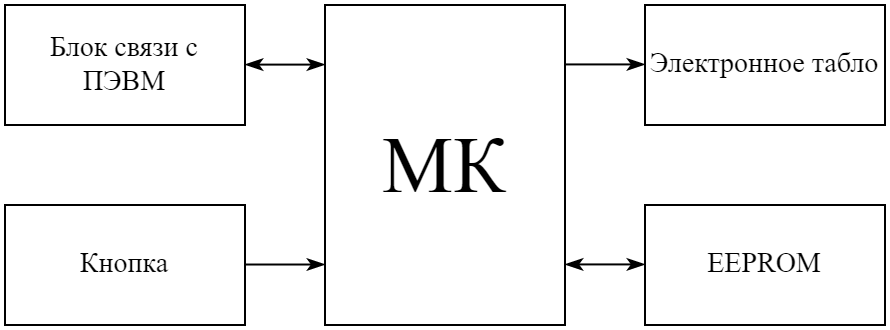


Рисунок 1 – Структурная схема устройства

Синтез и описание функциональной электрической схемы

1.2.1 Описание микроконтроллера Atmega16

В соответствие с техническим заданием для реализации системы необходимо использовать микроконтроллер AVR. При выборе микроконтроллера стоит учитывать то, что в нем необходимо наличие таких интерфейсов, как USART, SPI и TWI для связи с ПЭВМ, прошивки МК и связи с EEPROM соответственно. Также микроконтроллер должен содержать в себе достаточный объем памяти. Учитывая все эти требования, для реализации системы был выбран микроконтроллер Atmega16.

В разрабатываемом устройстве используется микроконтроллер в прямоугольном пластиковом корпусе с 40 выводами (40-pin PDIP). Конфигурация выводов микроконтроллера показана на рисунке 2.

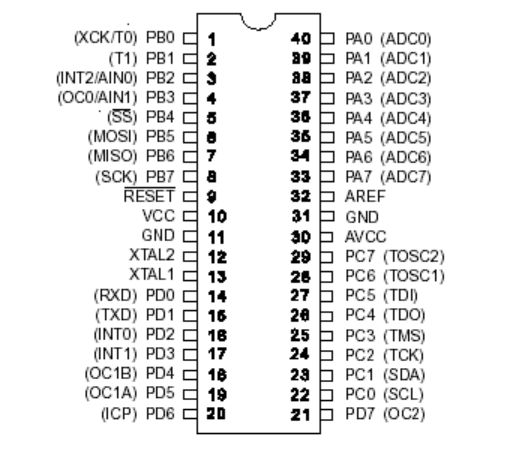


Рисунок 2 – Конфигурация выводов Atmega16

Для подачи питания и связи с внешними цепями микросхема МК имеет 40 контактов, в том числе 32 контакта, которые могут быть сконфигурированы для реализации конкретных задач и требований разработчика.

Atmega16 – экономичный 8-разрядный микроконтроллер, основанный на усиленной AVR RISC архитектуре. Atmega16 обеспечивает производительность 1 миллионов 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.

AVR ядро объединяет богатый набор инструкций с 32 рабочими регистрами общего назначения. Все 32 регистра непосредственно подключены к АЛУ (арифметико-логическое устройство), что позволяет указывать два регистра в одной инструкции и выполнить ее за один цикл. Данная архитектура обладает большей эффективностью кода и в 10 раз большей производительностью по сравнению с CISC микроконтроллерами.

Atmega16 обладает следующими возможностями: 16 кбайт внутрисхемно программируемой флэш-памяти с возможностью чтения во время записи, 512 байт ЭППЗУ, 1 кбайт статического ОЗУ, внешний интерфейс памяти, 32 линий ввода-вывода, 32 рабочих регистров общего назначения, два универсальных таймера-счетчика с режимами компаратора, внутренние и внешние запросы на прерывание, последовательный программируемый УСАПП, программируемый сторожевой таймер с внутренним генератором, последовательный порт SPI и три программно настраиваемых режима управления энергопотреблением. Режим холостого хода останавливает ЦПУ, но оставляет в работе статическое ОЗУ, таймеры-счетчики, порт SPI и систему прерываний. Режим пониженного потребления сохраняет содержимое регистров, но останавливает генератор, выключает все встроенные функции до появления следующего запроса на прерывание или аппаратного сброса. В дежурном режиме генератор на кварцевом резонаторе запущен, а остальная часть отключена. Данный режим позволяет реализовать быстрый запуск в комбинации с малым потреблением.

Atmega16 поддерживается полным набором инструментальных и программных средств для разработки приложений, в т.ч.: Cи-компиляторы, макроассемблеры, программные отладчики/симуляторы, внутрисхемные эмуляторы, оценочные наборы. Структурная схема микроконтроллера представлена на рисунке 3.

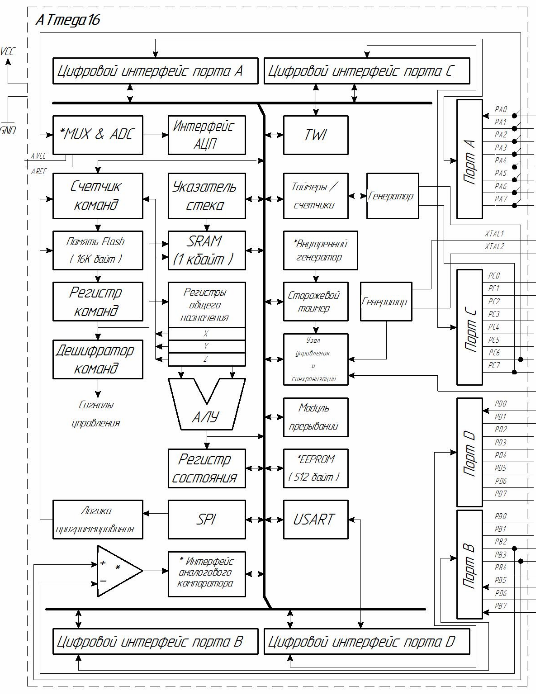
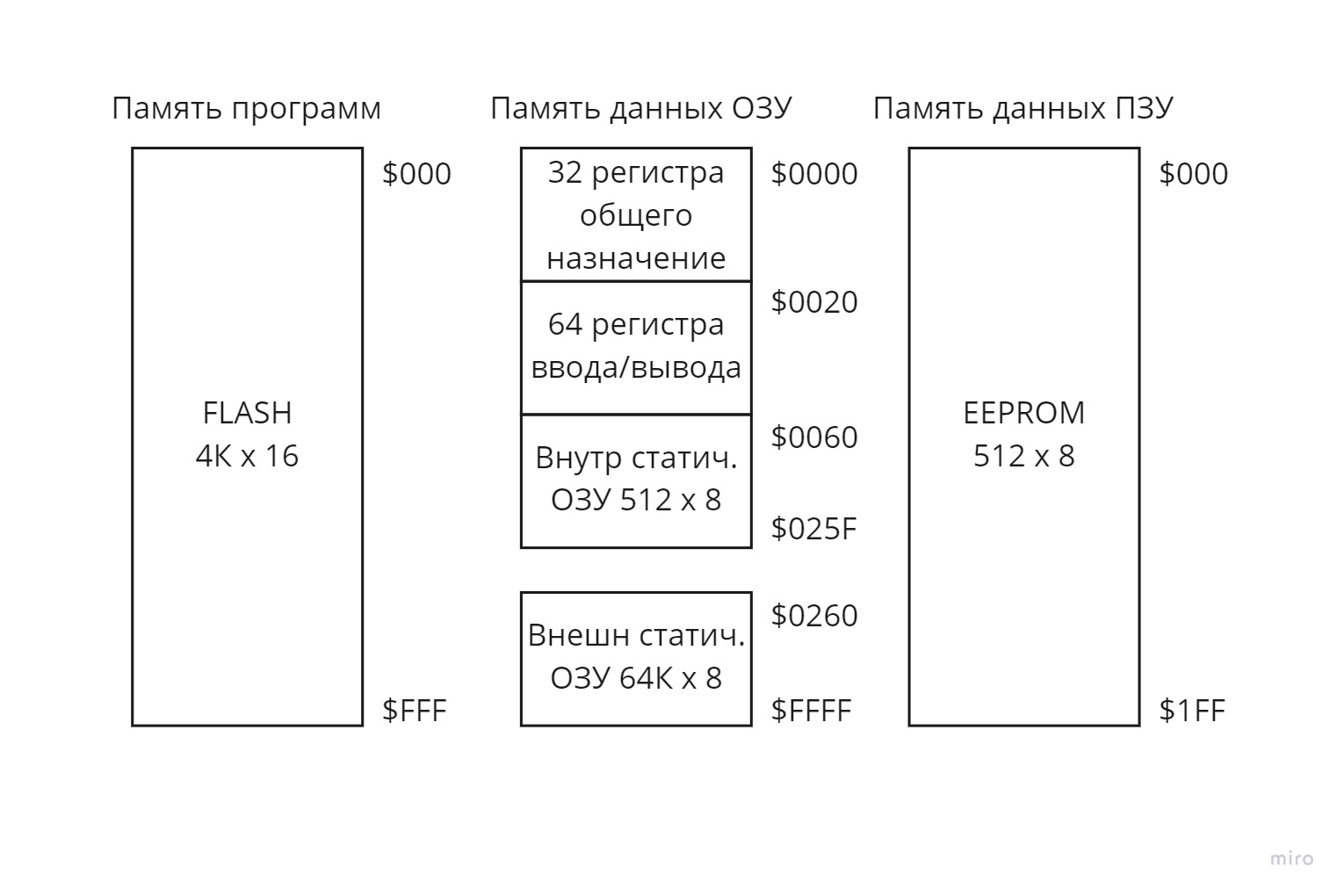


Рисунок 3 – Структурная схема Atmega16

1.2.2 Организация памяти микроконтроллера Atmega16

Реализация Гарвардской архитектуры в Atmega16 разделяет не только память программ и данных, но и каждая из областей памяти данных (ОЗУ и EEPROM) расположены в своем адресном пространстве. Структура памяти микроконтроллера Atmega16 приведена на рисунке 4.

Рисунок 4 – Структура памяти

Память программ предназначена для хранения последовательности команд, управляющих функционированием микроконтроллера, и имеет 16-ти битную организацию. Все AVR имеют Flash-память программ, которая может быть различного размера - от 1 до 256 КБайт. В МК Atmega16 объем Flash-памяти программ 4Кх16 (длина команды 16 разрядов, следовательно каждая команда находится в одной ячейке памяти). Ее главное достоинство в том, что она построена на принципе электрической перепрограммируемости, т. е. допускает многократное стирание и запись информации. Программа заносится во Flash-память AVR как с помощью обычного программатора, так и с помощью SPI-интерфейса, в том числе непосредственно на собранной плате. Возможностью внутрисхемного программирования через коммуникационный интерфейс SPI обладают практически все микроконтроллеры AVR.

Память данных разделена на три части:

* регистровая память;
* оперативная память (ОЗУ или RAM);
* энергозависимая память (ЭСППЗУ или EEPROM).

Регистровая память включает 32 регистра общего назначения (РОН или GPR), объединенных в файл, и служебные регистры ввода/вывода (РВВ). И те и другие расположены в адресном пространстве ОЗУ, но не являются его частью. В области регистров ввода/вывода расположены различные служебные регистры (регистры управления микроконтроллером, регистры состояния и т. п.), а также регистры управления периферийными устройствами, входящими в состав микроконтроллера. Структура 32 регистров общего назначения представлена на рисунке 5.

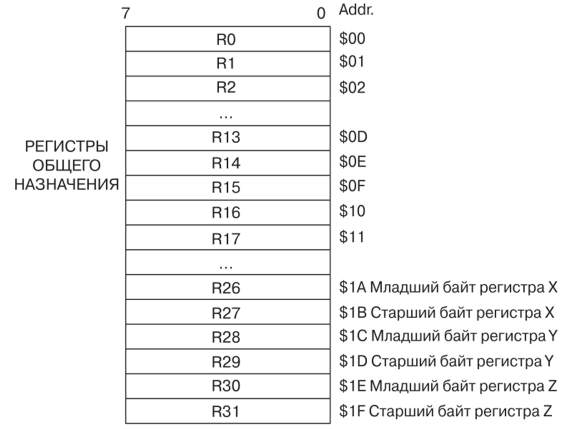


Рисунок 5 – Структура регистров общего назначения

Из рисунка видно, что каждому регистру соответствует адрес памяти данных, отображающий их в первых 32 ячейках пользовательского пространства данных. 16-разрядные регистры (X, Y, Z) получаются путем объединения двух регистров общего назначения.

Синтез и описание электрической принципиальной схемы

Подключение программатора и питания

Для подключения программатора к МК-системе для прошивки МК необходима вилка на плату. Для этого была использована вилка IDC-06MS. Расположение выводов вилки представлено на рисунке 6.

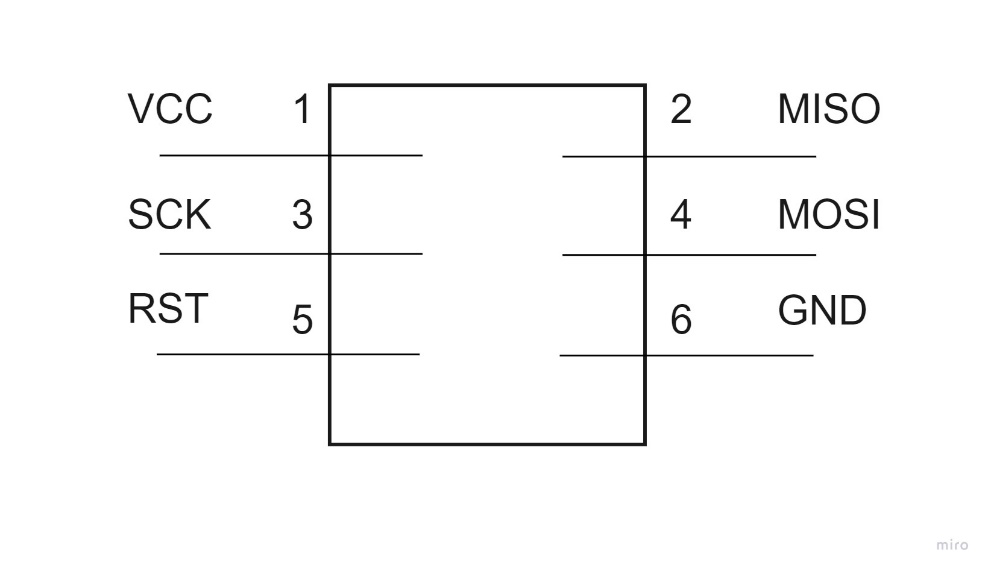


Рисунок 6 – Расположение выводов IDC-06MS

Для питания основных функциональных блоков разрабатываемого устройства необходимо напряжение 5В. Поэтому в разрабатываемой схеме был использован стабилизатор напряжения КР142ЕН5А, который преобразует входные 12В в 5В, а также двухконтактный разъем MPW-2R для подачи 12В.

КР142ЕН5А – стабилизатор напряжения с интегральной схемой с фиксированным напряжением. У стабилизатора 3 вывода, на первый вывод подается входное напряжение (+12В), второй вывод – это земля, а третий вывод – это выходное напряжение (+5В).

Электронное табло

В качестве электронного табло был использован LCD-дисплея LM032L на базе HD44780. Данный LCD-дисплей подключается к AVR микроконтроллеру напрямую к портам. Есть два способа подключения – на 8 бит и на 4 бита. При разработке МК-системы был использован способ подключения на 4 бита.

Выводы DB7…DB0 – это шина данных/адреса.

E – вход синхронизации.

RW – определяет в каком направлении движутся данные. Если 1 – то на чтение из дисплея, если 0, то на запись в дисплей.

RS – определяет, что передается, команда (RS=0) или данные (RS=1). Данные будут записаны в память по текущему адресу, а команда исполнена контроллером. В таблице 1 приведена система команд для управления дисплеем.

Таблица 1 – Система команд для управления LM032L

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | Значение |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Очистка экрана. Счетчик адреса на 0 позицию DDRAM |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | — | Адресация на DDRAM сброс сдвигов, Счетчик адреса на 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | Настройка сдвига экрана и курсора |
| 0 | 0 | 0 | 0 | 1 | D | C | B | Настройка режима отображения |
| 0 | 0 | 0 | 1 | S/C | R/L | — | — | Сдвиг курсора или экрана, в зависимости от битов |
| 0 | 0 | 1 | DL | N | F | — | — | Выбор числа линий, ширины шины и размера символа |
| 0 | 1 | AG | AG | AG | AG | AG | AG | Переключить адресацию на SGRAM и задать адрес в SGRAM |
| 1 | AD | AD | AD | AD | AD | AD | AD | Переключить адресацию на DDRAM и задать адрес в DDRAM |

Блок связи с ПЭВМ

Для организации работы МК-системы в режиме ввода и вывода данных через терминал необходима связь с ПЭВМ. Данная связь была реализована посредством последовательного интерфейса USART через COM-порт компьютера. Для этого был использован драйвер «MAX232».

Микросхема-драйвер фирмы Maxim серии MAX232 используется в качестве преобразователя уровней напряжения при связи устройства с ПЭВМ. Она содержит преобразователь напряжения +5В в напряжение +10В, инвертор, преобразующий напряжение +10В в напряжение -10В, и преобразователь уровней сигналов последовательного интерфейса.

Конфигурация выводов, а также пример подключения драйвера MAX232 представлен на рисунке 7.

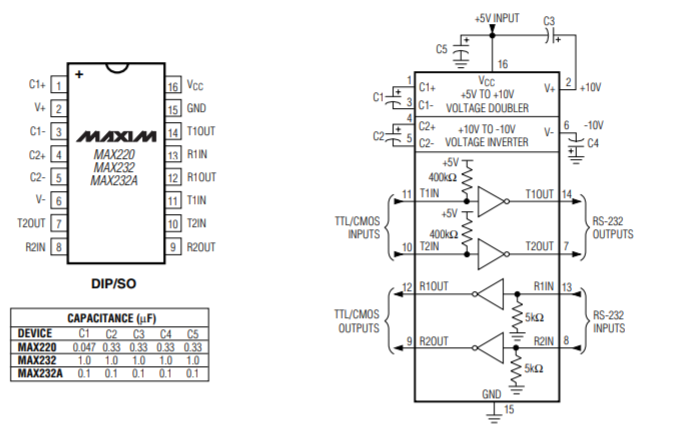


Рисунок 7 – Драйвер MAX232

Настройка USART

UART (универсальный асинхронный приёмопередатчик) – одна из старейших и самых распространенных на сегодняшний день технологий передачи данных. Слово «асинхронный» означает, что интерфейс не использует линию для синхросигнала, приемник и передатчик заранее настраиваются на одну частоту.

В современных микроконтроллерах, часто вместо UART используют полностью с ним совместимый – USART (универсальный асинхронный/синхронный приёмопередатчик). USART это более гибкий в настройки UART с дополнительными возможностями. Например в USART можно регулировать длину слова с более большим диапазоном (от 5 до 9), чем в UART (от 8 до 9). В USART, как видно из названия, возможна как асинхронная так и синхронная передача данных (в UART только асинхронная).

Передача данных в UART осуществляется по одному биту в равные промежутки времени. Этот временной промежуток определяется заданной скоростью UART и для конкретного соединения указывается в бодах, что соответствует количество бит в секунду. Существует общепринятый ряд стандартных скоростей: 4800, 9600, 19200 бод и т.д.

Настройка скорости обмена USART вычисляется по формуле 1:

|  |  |
| --- | --- |
|  | (1) |

где XTAL – частота работы МК (8 МГц), baudrate – требуемая скорость передачи (9600 бод).

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

Таким образом в регистр UBBR записываем значение 51. Разрешаем прием и передачу, записав в биты TXEN и RXEN регистра UCSRB единицы. Настраиваем стандартную 8-битную посылку, записав в бит UCSZ0 регистра UCSRC ноль.

Внешняя энергонезависимая память EEPROM

Последовательная энергонезависимая память 24LC64 обладает объемом памяти 64 КБ и частотой работы с шиной TWI 400 КГц. У микросхемы 8 контактов. WP – Write Protect (защита от записи), A0:A2 используются для выбора переменной части адреса ведомой микросхемы. Назначения выводов микросхемы изображены на рисунке 8.

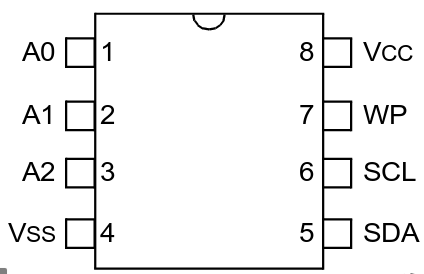


Рисунок 8 – Микросхема 24LC64

К шине TWI может быть подключено множество подчиненных устройств (к примеру, много микросхем внешней памяти EEPROM). Для того, чтобы все микросхемы и контроллер знали, от кого и кому передается информация, в протоколе реализована адресация ведомых устройств. В каждой микросхеме, предназначенной для работы с TWI, на заводе "зашит" определенный адрес. Этот адрес передаем по всей шине, т.е. всем ведомым. Каждый ведомый получает этот адрес и проверяет его ли это адрес.

Так как происходит работа с микросхемой памяти 24LC64, по шине надо передать ее адрес. Так она будет знать, что данные на запись или чтение адресуются именно ей. А остальные микросхемы, если они есть, эти данные будут просто игнорировать.

Постоянная часть адреса 24LC64 – 1010, 3 бита – переменные (для подключения нескольких одинаковых микросхем c одинаковыми заводскими адресами), далее бит 0 – если необходимо записывать в память или 1 – если необходимо читать данные из памяти.

Внешняя память EEPROM используется для хранения маршрутов. Первый байт памяти (адрес 0x0000) используется для хранения количества остановок маршрута. Следующие ячейки, начиная со второй (адрес 0x0001), хранят название остановок маршрута. На каждую остановку отводится 20 байт.

Расчет потребляемой мощности

Для определения потребляемой мощности МК воспользуемся формулой 5:

|  |  |
| --- | --- |
|  | (5) |

Где, – ток потребления МК равный 1.1 мА (взято из даташита), – напряжение питания равное 5В. Рассчитаем потребляемую мощность МК: Вт.

Далее рассчитаем суммарную потребляемую мощность остальных устройств разрабатываемой системы (таблица 3).

Таблица 2 – Потребляемая мощность устройств

|  |  |  |  |
| --- | --- | --- | --- |
|  | Потребляемая мощность , Вт | Количество микросхем | Суммарная потребляемая мощность , Вт |
| Резистор R1-R4 | 0,125 | 4 | 0,5 |
| LM044L | 0,05 | 1 | 0,05 |
| MAX232 | 0,05 | 1 | 0,05 |

Суммарная мощность всей системы равна:

Описание алгоритмов основных программных модулей

Главная процедура программы

При запуске МК-системы производится выполнение главной процедуры программы. Вначале происходит инициализация используемых устройств МК (порты, TWI, USART) и дисплея, после чего программа переходит в бесконечный цикл, где поочередно выполняется опрос кнопок и обновление дисплея. Схема алгоритма главной процедуры программы изображена на рисунке 9.

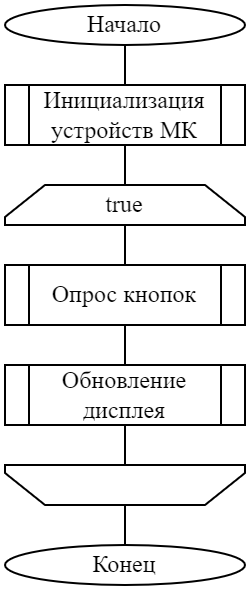


Рисунок 9 – Схема алгоритма главной процедуры программы

Процедура обновления дисплея

В процедуре «lcd\_update» происходит обновление дисплея. Сначала вызвается процедура очистки дислпея, затем происходит перемещение каретки в начало дисплея, а после – вывод на дисплей следующей станции. Схема алгоритма процедуры обновления дисплея изображена на рисунке 10.



Рисунок 10 – Схема алгоритма процедуры обновления дисплея

Процедура прерывания UART по приему

В процедуре «ISR(USART\_RXC\_vect)» происходит обработка прерывания UART по приему. Схема алгоритма процедуры прерывания UART по приему изображена на рисунке 11.

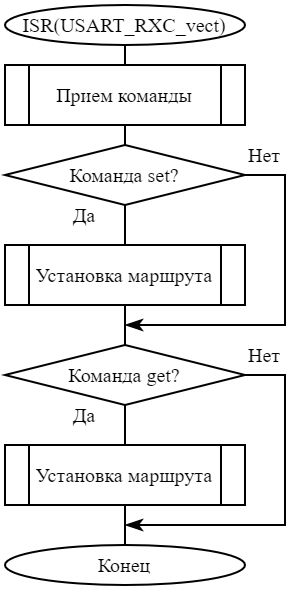


Рисунок 11 – Схема алгоритма процедуры прерывания UART по приему

Для взаимодействия МК и ПЭВМ используется две команды: «set» и «get». Команда «set» предназначена для настройки маршрута с ПЭВМ. Команда «get» предназначена для передачи маршрута с МК на ПЭВМ.

Процедура настройки маршрута

В процедуре «set\_route» происходит настройка маршрута с ПЭВМ. Для этого вначале происходит передача желаемого количества остановок на маршруте. Если количество равно от 0 до 255, то данное количество сохраняется в EEPROM по адрему 0x0000. После этого происходит передача остановки, а затем ее сохранение в EEPROM. Схема алгоритма процедуры настройки маршрута изображена на рисунке 12.

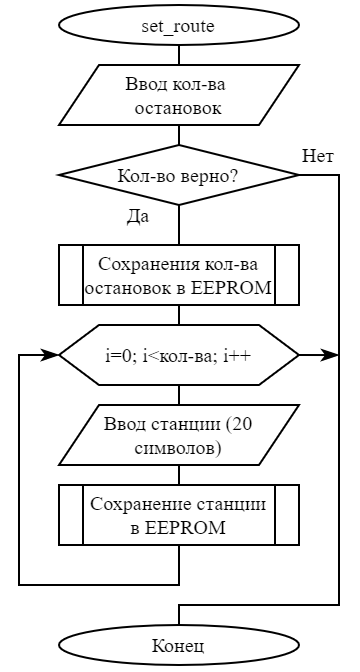


Рисунок 12 – Схема алгоритма процедуры настройки маршрута

Технологическая часть

Характеристика использованных систем для разработки и отладки программ

В ходе проектирования устройства в качестве средств разработки программной части были использованы следующие среды:

* Microchip Studio – для написания программного кода и его отладки, а также прошивки микроконтроллера. Microchip Studio — интегрированная среда (IDE) для разработки 8- и 32-битных AVR-приложений от компании Atmel. Microchip Studio содержит компилятор C/C++ и симулятор, позволяющий отследить выполнение программы. Microchip Studio содержит в себе менеджер проектов, редактор исходного кода, инструменты виртуальной симуляции и внутрисхемной отладки, позволяет писать программы на ассемблере или на C/C++.
* Proteus – для симуляции и тестирования проекта.

При написании программного кода были использованы следующие библиотеки:

* avr/io.h;
* avr/interrupt.h;
* stdio.h;
* stdlib.h;
* stdbool.h.

Тестирование устройства в симуляторе Proteus

Данный проект был промоделирован в Proteus. Были протестированы следующие программные модули:

* работа электронного табло;
* настройка маршрута с ПЭВМ.

На рисунке 13 изображена принципиальная схема устройства в симуляторе Proteus.

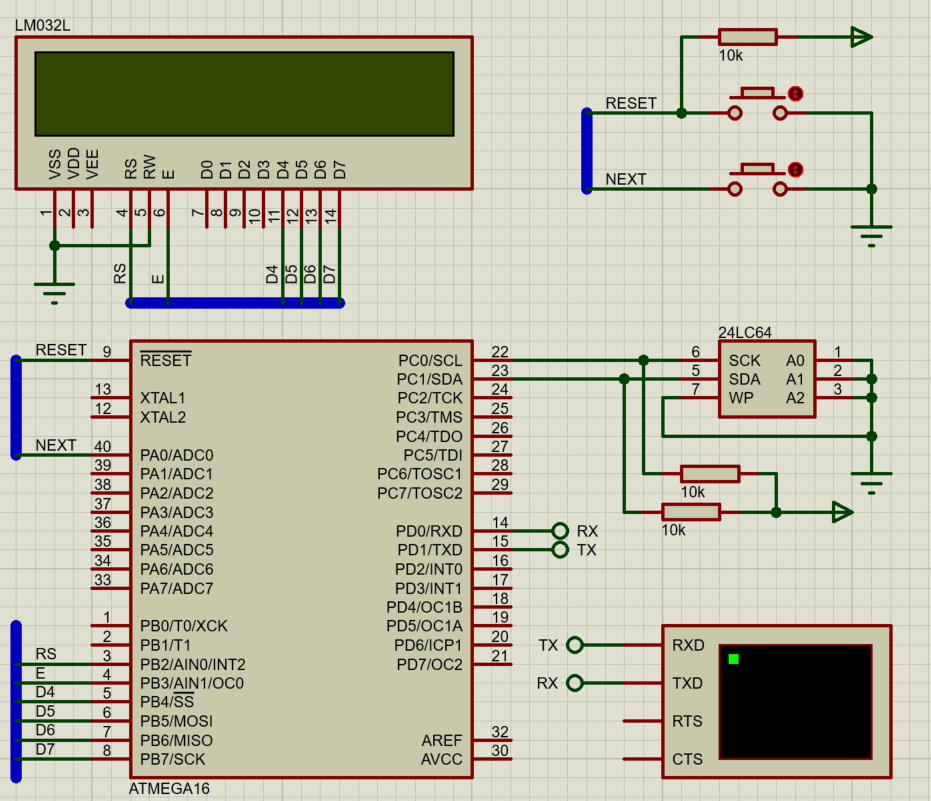


Рисунок 13 – Принципиальная схема устройства в симуляторе Proteus

На рисунке 14 показана работа электронного табло.

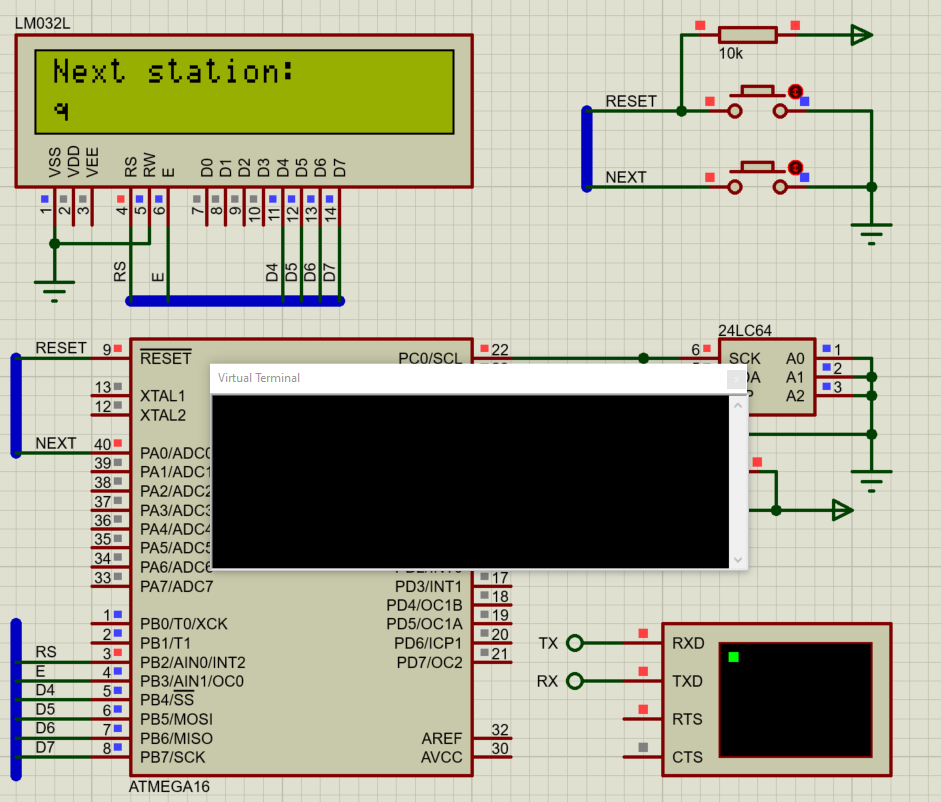


Рисунок 14 – Работа электронного табло

На рисунке 15 продемонстрирована настройка маршрута с ПЭВМ.

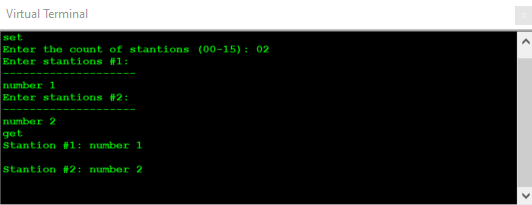


Рисунок 15 – Настройка маршрута с ПЭВМ

Программирование микроконтроллера

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. При программировании AVR программатор всегда функционирует как ведущее устройство, а микроконтроллер как ведомое. SPI является синхронным интерфейсом: все операции синхронизированы фронтами тактового сигнала (SCK), который вырабатывается ведущим устройством. Максимальная скорость передачи ограничена величиной 1/4 тактовой частоты контроллера. На минимальную скорость нет никаких ограничений: без тактового сигнала обмен данными «замораживается», и интерфейс может оставаться в статическом состоянии сколько угодно долго. Для работы с модулем SPI используются три регистра:

* SPDR (SPI Data Register) – регистр данных. В этот регистр заносится байт для последующей его передачи и из него же считывается пришедший байт информации;
* SPSR (SPI Status Register) – статусный регистр. Предназначен для контроля состояния SPI модуля, содержит дополнительный бит управления скоростью обмена;
* SPCR (SPI Control Register) – управляющий регистр. С помощью данного регистра устанавливается конфигурация модуля SPI.

На рисунке 16 представлена структура регистра SPSR.

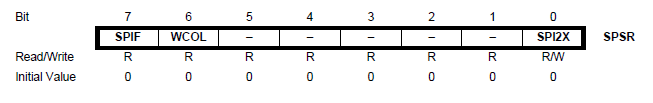


Рисунок 16 – структура регистра SPSR

SPIF – флаг прерывания от SPI. Он устанавливается в 1 по окончании передачи байта данных. Если разрешены прерывания модуля, одновременно с установкой этого флага генерируется прерывание от SPI. Также этот флаг устанавливается в 1 при переводе микроконтроллера из режима master в режим slave с помощью вывода SS. Сброс флага происходит аппаратно, при вызове подпрограммы обработки прерывания или после чтения регистра SPSR с последующим обращением к регистру данных SPDR.

WCOL – флаг конфликта записи. Флаг устанавливается в 1, если во время передачи данных выполняется попытка записи в регистр данных SPDR. Флаг сбрасывается аппаратно после чтения регистра SPSR с последующим обращением к регистру данных SPDR.

SPI2X – бит удвоения скорости обмена. Установка этого разряда в 1 удваивает частоту тактового сигнала SCK. Микроконтроллер при этом должен работать в режиме master.

В таблице 3 представлена зависимость частоты тактового сигнала SCK от битов SPR0, SPR1, SPI2X, где fМК- тактовая частота микроконтроллера.

Таблица 3 – Зависимость частоты тактового сигнала SCK от битов SPR0, SPR1, SPI2X

|  |  |  |  |
| --- | --- | --- | --- |
| SPI2X | SPR1 | SPR0 | SCK Frequency |
| 0 | 0 | 0 | fМК/4 |
| 0 | 0 | 1 | fМК/16 |
| 0 | 1 | 0 | fМК/64 |
| 0 | 1 | 1 | fМК/128 |
| 1 | 0 | 0 | fМК/2 |
| 1 | 0 | 1 | fМК/8 |
| 1 | 1 | 0 | fМК/32 |
| 1 | 1 | 1 | fМК/64 |

На рисунке 17 представлена структура регистра SPСR.

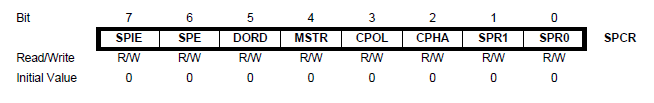


Рисунок 17 – структура регистра SPСR

SPIE – разрешает /запрещает прерывания от модуля SPI. Если бит установлен в 1, прерывания от SPI разрешены.

SPE – включает/выключает модуль SPI. Если бит установлен в 1, модуль SPI включен.

DORD – определяет порядок передачи данных. Когда бит установлен в 1, содержимое регистра данных передается младшим битом вперед. Когда бит сброшен, то старшим битом вперед.

MSTR – определяет режим работы микроконтроллера. Если бит установлен в 1, микроконтроллер работает в режиме Master (ведущий). Если бит сброшен – в режиме Slave (ведомый).

CPOL и CPHA – определяют в каком режиме работает SPI модуль. Требуемый режим работы зависит от используемого периферийного устройства.

SPR1 и SPR0 – определяют частоту тактового сигнала SPI модуля, то есть скорость обмена. Максимально возможная скорость обмена всегда указывается в спецификации периферийного устройства.

Программирование микроконтроллера по SPI осуществляется путем посылки 4-байтовых команд на вывод MOSI МК, в который один или два байта определяют тип операции, остальные – адрес, записываемый байт, установочные биты и биты защиты, пустой байт. При выполнении операции чтения считываемый байт снимается через вывод MISO. Так же можно запрограммировать память данных EEPROM. В каждой команде указывается адрес записываемой ячейки и записываемое значение.

Форматы байтов команд для программирования микроконтроллера представлены на рисунке 18.

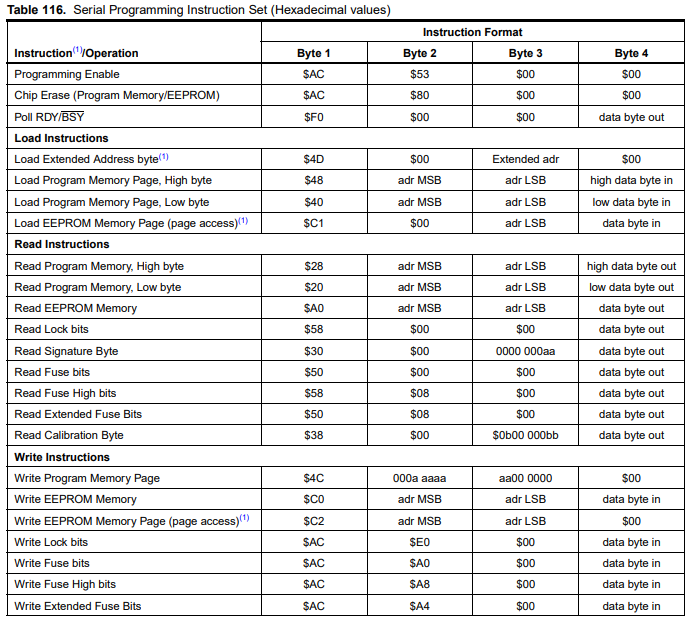


Рисунок 18 – Набор инструкций последовательного программирования через SPI

Для программирования микроконтроллера по последовательному интерфейсу SPI рекомендуется придерживаться следующей последовательности:

Последовательность подачи питания - подать напряжение питания между VCC и GND, когда на входах RESET и SCK установлен "0". В некоторых системах, программатор не может гарантировать, что SCK = 0 при подаче питания. В этом случае необходимо сформировать положительный импульс на RESET длительностью не менее двух тактов ЦП после того, как SCK принял значение "0".

Подождать не менее 20мс и включить последовательное программирование путем записи команды разрешения через вход MOSI.

Инструкции последовательного программирования не выполняются, если связь не синхронизирована. Когда связь синхронизирована, второй байт (0x53) будет возвращаться при выдаче третьего байта команды включения программирования. В зависимости от того корректно или нет принятое значение передаются все четыре байта инструкции. Если 0x53 не был получен, то формируется положительный импульс на входе RESET и вводится новая команда включения программирования.

Флэш-память программируется по одной странице (64 байта) за раз. Страница памяти загружается побайтно, представляя 5 LSB адреса и данных вместе с инструкцией загрузки страницы памяти программы. Чтобы гарантировать корректность загрузки страницы сначала необходимо записать младший байт, а затем старший байт данных по каждому адресу.

Массив памяти EEPROM программируется побайтно, представляя адрес вместе с соответствующей инструкцией записи. Место в памяти EEPROM автоматически стирается перед записью новых данных.

Любую ячейку памяти можно проверить использованием инструкции чтения, которая возвращает содержимое ячейки по указанному адресу путем последовательной передачи на выходе MISO.

По завершении программирования вход RESET необходимо установить на высокое значение, чтобы начать нормальную работу.

Последовательность снятия питания (при необходимости): установка RESET = "1", отключить питание VCC.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было спроектирован и реализован индикатор маршрута в метрополитене на основе микроконтроллера Atmega16. Устройство позволяет отображать маршрут с указанием следующей станции и хранить маршрут в EEPROM.

В результате проектирования были разработаны принципиальная и функциональная электрические схемы для аппаратной части устройства. Также были разработаны коды модулей для программы на языке Си.

Разработанное устройство удовлетворяет требованиям, предъявленным в задании на курсовую работу, и может использоваться в качестве индикатора маршрута в метрополитене.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Хартов, В.Я. Микропроцессорные системы: учеб. пособие для студ. учреждений высш. проф. образования, Академия, М., 2014. – 368с.

2 Хартов, В.Я. Микроконтроллеры AVR. Практикум для начинающих: 2-е издание, Издательство МГТУ им. Н.Э. Баумана, 2012. – 278с.

3 Документация на микроконтроллер Atmega16 [Электронный ресурс]. – URL: http://ww1.microchip.com/downloads/en/devicedoc/doc2512.pdf (дата обращения 25.10.2022).

4 Документация на драйвер MAX232 [Электронный ресурс]. – URL: https://pdf1.alldatasheet.com/datasheet-pdf/view/397176/MAXIM/MAX232.html (дата обращения 25.10.2022).

5 Информация о библиотеке delay.h [Электронный ресурс]. – URL: https://www.nongnu.org/avr-libc/user-manual/group\_\_util\_\_delay.html (дата обращения 25.10.2022).

6 ГОСТ 2.702-2011 Правила выполнения электрических схем

7 ГОСТ 2.710-81 Обозначения буквенно-цифровые в электрических схемах

8 ГОСТ 2.721-74 Обозначения условные графические в схемах. Обозначения общего применения

9 ГОСТ 2.102-68 ЕСКД. Виды и комплектность конструкторских документов

10 ГОСТ 2.105-95 ЕСКД. Текстовые документы

ПРИЛОЖЕНИЕ А. Текст исходной программы

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

#define BAUDRATE 9600L

#include <avr/io.h> //используемые библиотеки

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define false 0

#define true 1

#define LCD\_DDR DDRB //порт дисплея

#define LCD\_PORT PORTB

#define E1 PORTB |= 0b00001000 //установка линии E в 1

#define E0 PORTB &= 0b11110111 //установка линии E в 0

#define RS1 PORTB |= 0b00000100 //установка линии RS в 1 (данные)

#define RS0 PORTB &= 0b11111011 //установка линии RS в 0 (команда)

// Порт кнопок

#define BUT\_DDR DDRA

#define BUT\_PORT PORTA

#define BUT\_PIN PINA

#define BUT\_NEXT PA0

#define slaveF\_SCL 400000 //400 кГц для внешней еепром

#define TW\_START 0x08

#define TW\_REP\_START 0x10

#define slaveAddressConst 0b1010 //постоянная часть адреса ведомого устройства

#define slaveAddressVar 0b000 //переменная часть адреса ведомого устройства

//разряды направления передачи данных

#define READFLAG 1 //чтение

#define WRITEFLAG 0 //запись

//константы для TWI

//master transmitter

#define TW\_MT\_SLA\_ACK 0x18

#define TW\_MT\_DATA\_ACK 0x28

//master receiver

#define TW\_MR\_SLA\_ACK 0x40

#define TW\_MR\_DATA\_NACK 0x58

char rx\_data[32]; //массив символов для приема по usart

char buffer[32]; //массив для вывода символов на дисплей

uint8\_t flag\_next = false;// Флаг переключения станции

const char cmd\_set\_route[] = "set\r"; //команда установки маршрута

const char cmd\_get\_route[] = "get\r"; //команда получения маршрута

static int usart\_put\_char(char c, FILE \*stream);

static int lcd\_put\_char(char c, FILE \*stream);

static FILE usart = FDEV\_SETUP\_STREAM(usart\_put\_char, NULL, \_FDEV\_SETUP\_WRITE);

static FILE lcd = FDEV\_SETUP\_STREAM(lcd\_put\_char, NULL, \_FDEV\_SETUP\_WRITE);

//инициализация TWI

void ee\_init(void)

{

TWBR = (F\_CPU / slaveF\_SCL - 16) / (2 \* /\* TWI\_Prescaler= 4^TWPS \*/1);

if (TWBR < 10)

TWBR = 10;

TWSR &= (~((1<<TWPS1)|(1<<TWPS0)));

}

//запись байта в еепром внешний

uint8\_t ee\_write\_byte(uint16\_t address, uint8\_t data)

{

do

{

TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_START)

{

return false;

}

TWDR = (slaveAddressConst<<4) + (slaveAddressVar<<1) + (WRITEFLAG);

TWCR = (1<<TWINT) | (1<<TWEN);

while(!(TWCR & (1<<TWINT)));

} while((TWSR & 0xF8) != TW\_MT\_SLA\_ACK);

TWDR = (address>>8);

TWCR = (1<<TWINT)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_MT\_DATA\_ACK)

{

return false;

}

TWDR=(address);

TWCR=(1<<TWINT)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_MT\_DATA\_ACK)

return false;

TWDR=(data);

TWCR=(1<<TWINT)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_MT\_DATA\_ACK)

return false;

TWCR=(1<<TWINT)|(1<<TWEN)|(1<<TWSTO);

while(TWCR & (1<<TWSTO));

return true;

}

//чтение байта из внешнего еепром

uint8\_t ee\_read\_byte(uint16\_t address)

{

uint8\_t data;

do

{

TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_START)

return false;

TWDR = (slaveAddressConst<<4) + (slaveAddressVar<<1) + WRITEFLAG;

TWCR = (1<<TWINT)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

}while((TWSR & 0xF8) != TW\_MT\_SLA\_ACK);

/\*\*\*\*\*ПЕРЕДАЕМ АДРЕС ЧТЕНИЯ\*\*\*\*\*\*\*\*/

TWDR = (address>>8);

TWCR = (1<<TWINT)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_MT\_DATA\_ACK)

return false;

TWDR = (address);

TWCR = (1<<TWINT)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_MT\_DATA\_ACK)

return false;

/\*\*\*\*\*ПЕРЕХОД В РЕЖИМ ЧТЕНИЯ\*\*\*\*\*\*\*\*/

TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_REP\_START)

return false;

TWDR = (slaveAddressConst<<4) + (slaveAddressVar<<1) + READFLAG;

TWCR = (1<<TWINT)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_MR\_SLA\_ACK)

return false;

/\*\*\*\*\*СЧИТЫВАЕМ БАЙТ ДАННЫХ\*\*\*\*\*\*\*\*/

TWCR = (1<<TWINT)|(1<<TWEN);

while(!(TWCR & (1<<TWINT)));

if((TWSR & 0xF8) != TW\_MR\_DATA\_NACK)

return false;

data = TWDR;

TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);

while(TWCR & (1<<TWSTO));

return data;

}

//инициализация usart

void usart\_init()

{

stderr = &usart;

UBRRL = F\_CPU / BAUDRATE / 16 - 1; //8 000 000 / 9600 / 16 - 1 = 51

UCSRB = (1<<TXEN)|(1<<RXEN); //разрешение приема и передачи

UCSRC = (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1); //9 бит

UCSRB |= (1<<RXCIE)|(1<<UCSZ2); //разрешение прерывания при передаче

}

//отправка символа по usart

static int usart\_put\_char(char c, FILE \*stream)

{

if (c == '\n')

{

usart\_put\_char('\r', stream);

}

while(!(UCSRA & (1<<UDRE)));

UDR = c;

return 0;

}

//прием данных по usart

void receiving\_usart()

{

memset(rx\_data, 0, sizeof rx\_data);

int i = 0;

do

{

while(!(UCSRA&(1<<RXC)));

rx\_data[i] = UDR;

i++;

} while (rx\_data[i-1] != '\r');

}

uint8\_t check\_number()

{

uint8\_t counter = 0;

uint8\_t counter2 = 0;

for (uint8\_t element = 0; element < 3; element++)

{

for (uint8\_t i = 0; i < 10; i++)

{

if (rx\_data[element] == (i | 0b00110000))

{

counter++;

}

}

if (counter > 0)

{

counter = 0;

}

else

{

counter2++;

}

}

if (counter2 > 0)

{

return 1;

}

else

{

if ( (((rx\_data[0] & 0b00001111) \* 100) + ((rx\_data[1] & 0b00001111) \* 10) + (rx\_data[2] & 0b00001111)) > 255 )

{

return 1;

}

return 0;

}

}

//установка маршрута

void set\_route()

{

fprintf(stderr, "Enter the count of stantions (000-255): ");

receiving\_usart();

//fprintf(stderr, "check\_number: %d\n", check\_number());

if (check\_number())

{

fprintf(stderr, "Wrong count!\n");

}

else

{

ee\_write\_byte(0, ((rx\_data[0] & 0b00001111) \* 100) + ((rx\_data[1] & 0b00001111) \* 10) + (rx\_data[2] & 0b00001111));

\_delay\_ms(5);

for (int stantion = 0; stantion < ee\_read\_byte(0); stantion++)

{

fprintf(stderr, "Enter stantions #%d:\n", stantion + 1);

fprintf(stderr, "--------------------\n");

receiving\_usart();

for (int letter = 0; letter < 20; letter++)

{

ee\_write\_byte(1 + (stantion \* 20 + letter), rx\_data[letter]);

\_delay\_ms(5);

}

}

}

}

//получение маршрута

void get\_route()

{

for (int stantion = 0; stantion < ee\_read\_byte(0); stantion++)

{

fprintf(stderr, "Stantion #%d: ", stantion + 1);

for (int letter = 0; letter < 20; letter++)

{

fprintf(stderr, "%c", ee\_read\_byte(1 + (stantion \* 20 + letter)));

}

fprintf(stderr, " \n");

}

}

//прерывание usart

ISR(USART\_RXC\_vect)

{

receiving\_usart();

if (strcmp(rx\_data, cmd\_set\_route) == 0)

{

set\_route();

}

else if (strcmp(rx\_data, cmd\_get\_route) == 0)

{

get\_route();

}

}

//отправка полбайта в дисплей

void lcd\_send\_halfbyte(unsigned char c)

{

c <<= 4;

E1; //включение линии Е

\_delay\_us(50);

LCD\_PORT &= 0b00001111; //стираем информацию на входах DB4-DB7, остальное не трогаем

LCD\_PORT |= c;

E0; //выключение линии Е

\_delay\_us(50);

}

//отправка байта в дисплей

void lcd\_send\_byte(unsigned char c, unsigned char mode)

{

if (mode == 0)

{

RS0;

}

else

{

RS1;

}

unsigned char hc = 0;

hc = c >> 4;

lcd\_send\_halfbyte(hc);

lcd\_send\_halfbyte(c);

}

//отправка символа в дисплей

void lcd\_send\_char(unsigned char c)

{

lcd\_send\_byte(c, 1);

}

//отправка символа по usart

static int lcd\_put\_char(char c, FILE \*stream)

{

if (c == '\n')

{

lcd\_put\_char('\r', stream);

}

lcd\_send\_char(c);

return 0;

}

//установка координат каретки на дисплее

void lcd\_set\_pos(unsigned char x, unsigned y)

{

switch(y)

{

case 0:

lcd\_send\_byte(x | 0x80, 0);

break;

case 1:

lcd\_send\_byte((0x40 + x) | 0x80, 0);

break;

case 2:

lcd\_send\_byte((0x14 + x) | 0x80, 0);

break;

case 3:

lcd\_send\_byte((0x54 + x) | 0x80, 0);

break;

}

}

//инициализация дисплея

void lcd\_init(void)

{

stdout = &lcd;

\_delay\_ms(15); //Ждем 15 мс (стр 45)

lcd\_send\_halfbyte(0b00000011);

\_delay\_ms(4);

lcd\_send\_halfbyte(0b00000011);

\_delay\_us(100);

lcd\_send\_halfbyte(0b00000011);

\_delay\_ms(1);

lcd\_send\_halfbyte(0b00000010);

\_delay\_ms(1);

lcd\_send\_byte(0b00101000, 0); //4бит-режим (DL=0) и 2 линии (N=1)

\_delay\_ms(1);

lcd\_send\_byte(0b00001100, 0); //включаем изображение на дисплее (D=1), курсоры никакие не включаем (C=0, B=0)

\_delay\_ms(1);

lcd\_send\_byte(0b00000110, 0); //курсор (хоть он у нас и невидимый) будет двигаться влево

\_delay\_ms(1);

}

//отчистка дисплея

void lcd\_clear(void)

{

lcd\_send\_byte(0b00000001, 0);

\_delay\_us(1500);

}

//вывод строки в дисплей

void lcd\_str (char str1[])

{

wchar\_t n;

for (n=0; str1[n] != '\0'; n++)

{

lcd\_send\_char(str1[n]);

}

}

//обновление дисплея

void lcd\_update()

{

static uint8\_t current\_stantion = 0;

if (flag\_next)

{

lcd\_clear();

lcd\_set\_pos(0, 0);

fprintf(stdout, "Next station:");

lcd\_set\_pos(0, 1);

for (int letter = 0; letter < 20; letter++)

{

fprintf(stdout, "%c", ee\_read\_byte(1 + (current\_stantion \* 20 + letter)));

}

current\_stantion++;

if (current\_stantion >= ee\_read\_byte(0))

{

current\_stantion = 0;

}

flag\_next = false;

}

}

//функция опроса кнопки

void ask\_button()

{

if (!(BUT\_PIN & (1<<BUT\_NEXT)))

{

while (!(BUT\_PIN & (1<<BUT\_NEXT)));

flag\_next = true;

}

}

//инициализация всех устройств

void init()

{

LCD\_DDR = 0xff;

LCD\_PORT = 0x00;

BUT\_DDR &= ~(1<<BUT\_NEXT);

BUT\_PORT |= (1<<BUT\_NEXT);

usart\_init();

lcd\_init();

ee\_init();

sei();

}

//точка входа в программу

int main(void)

{

init();

while (1)

{

ask\_button();

lcd\_update();

}

}

ПРИЛОЖЕНИЕ Б. Спецификация радиоэлементов схемы

