|  |  |
| --- | --- |
| 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:***

**Система управления потоком посетителей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-73Б |  |  | А.Н. Пахомова |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Руководитель |  |  |  | С.В. Ибрагимов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |

*2022 г.*

# РЕФЕРАТ

РПЗ 39 с., 10 рис., 1 табл., 4 ист., 2 прил.

МИКРОКОНТРОЛЛЕР, СИСТЕМА УПРАВЛЕНИЯ, ATMEGA128, ATMEGA16, USART.

Объектом разработки данной курсовой работы является проектирование системы управления потоком посетителей.

Цель работы – закрепление знаний, полученных при изучении дисциплины «Микропроцессорные системы», в процессе самостоятельной работы при проектировании системы управления потоком посетителей; 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 было спроектирована требуемая МК-система и получена сопутствующая документация, а именно: функциональная и принципиальная схемы, схемы алгоритмов управления и соответствующая программа микроконтроллера.

# ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК – микроконтроллер

ATmega128 – используемый микроконтроллер

ATmega16 – используемый микроконтроллер

SPI – (Serial Peripheral Interface) последовательный периферийный

интерфейс

Proteus ISIS – среда моделирования

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

синхронный приемопередатчик

ПЭВМ – персональная электронно-вычислительная машина

# СОДЕРЖАНИЕ

[РЕФЕРАТ 3](#_Toc91628137)

[ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 4](#_Toc91628138)

[СОДЕРЖАНИЕ 5](#_Toc91628139)

[ВВЕДЕНИЕ 7](#_Toc91628140)

[1 Конструкторская часть 8](#_Toc91628141)

[1.1 Анализ требований 8](#_Toc91628142)

[1.2 Синтез и описание функциональной электрической схемы 9](#_Toc91628143)

[1.2.1 Описание архитектуры и технических характеристик микроконтроллера 9](#_Toc91628144)

[1.2.1.1 Описание микроконтроллера ATmega128 9](#_Toc91628145)

[1.2.1.2 Организация памяти микроконтроллера 12](#_Toc91628146)

[1.2.2 Описание интерфейса TWI 14](#_Toc91628147)

[1.3 Синтез и описание электрической принципиальной схемы 17](#_Toc91628148)

[1.3.1 Программатор и схема питания МК-системы 17](#_Toc91628149)

[1.3.2 Пульт оператора и устройство индикации 18](#_Toc91628150)

[1.3.3 Блок связи с ПЭВМ 19](#_Toc91628151)

[1.3.4 Датчик температуры DS1621 21](#_Toc91628152)

[1.3.5 Протокол работы с датчиком температуры DS1621 22](#_Toc91628153)

[1.3.6 Внешняя энергонезависимая EEPROM 24LC64 24](#_Toc91628154)

[1.3.7 Настройка аппаратного таймера 25](#_Toc91628155)

[1.3.8 Расчет потребляемой мощности 25](#_Toc91628156)

[1.4 Описание алгоритмов основных программных модулей 27](#_Toc91628157)

[1.4.1 Главная процедура 27](#_Toc91628158)

[1.4.2 Процедура обновления температуры 29](#_Toc91628159)

[1.4.3 Процедура регистрации данных 29](#_Toc91628160)

[1.4.4 Процедура передачи показаний в ПЭВМ 32](#_Toc91628161)

[2 Технологическая часть 34](#_Toc91628162)

[2.1 Характеристика использованных систем для разработки и отладки программ 34](#_Toc91628163)

[2.2 Тестирование устройства в симуляторе Proteus 34](#_Toc91628164)

[2.3 Программирование микроконтроллера 36](#_Toc91628165)

[ЗАКЛЮЧЕНИЕ 42](#_Toc91628166)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 43](#_Toc91628167)

[ПРИЛОЖЕНИЕ А. Текст исходной программы 44](#_Toc91628168)

[ПРИЛОЖЕНИЕ Б. Спецификация радиоэлементов схемы 54](#_Toc91628169)

# ВВЕДЕНИЕ

Курсовая работа «Система управления потоком посетителей» выполнялась на основании учебного плана кафедры ИУ6.

Цель данной курсовой работы – разработка на основе микроконтроллеров AVR системы управления потоком посетителей.

Разработка системы управления потоком посетителей 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.

## 1 Конструкторская часть

### 1.1 Анализ требований

Согласно техническому заданию, необходимо разработать на основе микроконтроллера семейства AVR систему управления потоком посетителей. Система должна состоять из трех устройств, расположенных в разных местах здания. Два МК предназначены для выдачи номера посетителю, путем изображения его на дисплее по нажатию кнопку, один – для вызова человека в порядке очереди. Взаимодействие между ними осуществляется по интерфейсу USART.

Основанием для выполнения данной работы являются:

* учебный план кафедры ИУ6;
* задание на курсовую работу.

Для реализации данного функционала устройство должно содержать следующие структурные блоки:

* два ведущих микроконтроллера;
* ведомый микроконтроллер;
* пульт управления для каждого МК.

Структурная схема устройства представлена на рисунке 1.

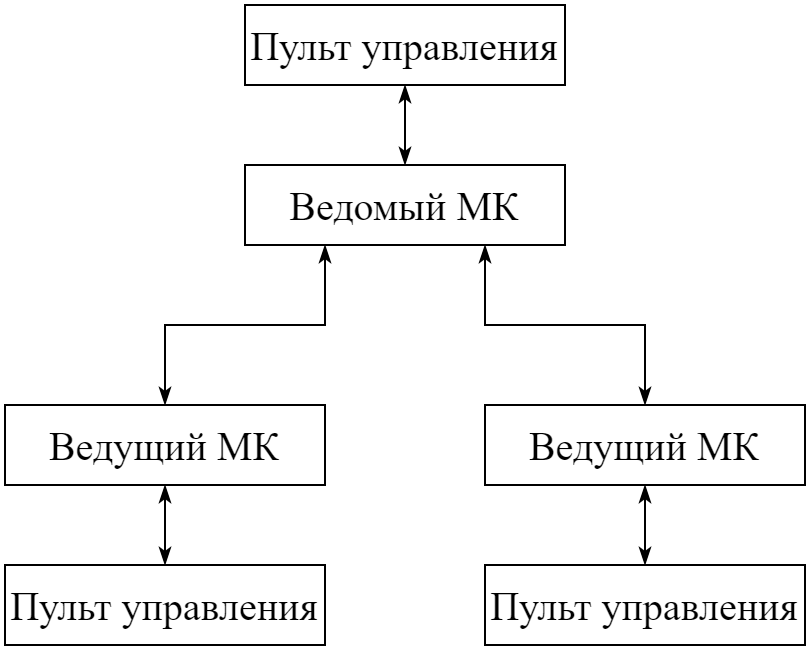


Рисунок 1 – Структурная схема устройства

### 1.2 Описание архитектуры и технических характеристик микроконтроллеров

#### 1.2.1 Микроконтроллеры ATmega128 и ATmega16

В разрабатываемом устройстве использовано два разных микроконтроллера фирмы Atmel – ATmega128 и ATmega16. Конфигурации выводов микроконтроллера ATmega128 и ATmega16 показаны на рисунке 2.

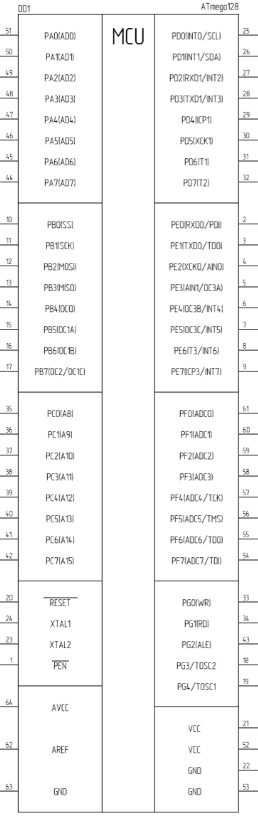
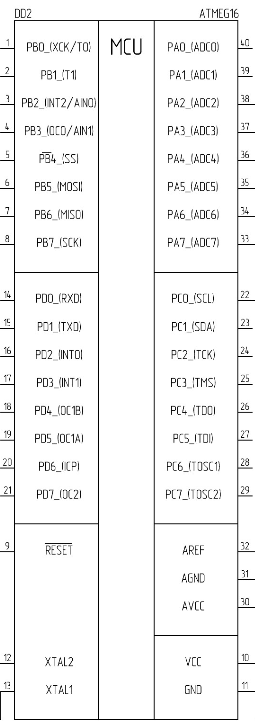
 

Рисунок 2 – Расположение выводов МК

ATmega128 – маломощный 8-разр. КМОП микроконтроллер, основанный на расширенной AVR RISC-архитектуре. За счет выполнения большинства инструкций за один машинный цикл ATmega128 достигает производительности 1 млн. операций в секунду/МГц, что позволяет проектировщикам систем оптимизировать соотношение энергопотребления и быстродействия.

Ядро AVR сочетает богатый набор инструкций с 32 универсальными рабочими регистрами. Все 32 регистра непосредственно подключены к арифметико-логическому устройству (АЛУ), который позволяет указать два различных регистра в одной инструкции и выполнить ее за один цикл. Данная архитектура обладает большей эффективностью кода за счет достижения производительности в 10 раз выше по сравнению с обычными CISC-микроконтроллерами.

ATmega128 содержит следующие элементы: 128 кбайт внутрисистемно программируемой флэш-памяти с поддержкой чтения во время записи, 4 кбайт ЭСППЗУ, 4 кбайт статического ОЗУ, 53 линии универсального ввода-вывода, 32 универсальных рабочих регистра, счетчик реального времени (RTC), четыре гибких таймера-счетчика с режимами сравнения и ШИМ, 2 УСАПП, двухпроводной последовательный интерфейс ориентированный на передачу байт, 8-канальный 10-разр. АЦП с опциональным дифференциальным входом с программируемым коэффициентом усиления, программируемый сторожевой таймер с внутренним генератором, последовательный порт SPI, испытательный интерфейс JTAG совместимый со стандартом IEEE 1149.1, который также используется для доступа к встроенной системе отладке и для программирования, а также шесть программно выбираемых режимов уменьшения мощности. Режим холостого хода (Idle) останавливает ЦПУ, но при этом поддерживая работу статического ОЗУ, таймеров-счетчиков, SPI-порта и системы прерываний. Режим выключения (Powerdown) позволяет сохранить содержимое регистров, при остановленном генераторе и выключении встроенных функций до следующего прерывания или аппаратного сброса. В экономичном режиме (Power-save) асинхронный таймер продолжает работу, позволяя пользователю сохранить функцию счета времени в то время, когда остальная часть контроллера находится в состоянии сна. Режим снижения шумов АЦП (ADC Noise Reduction) останавливает ЦПУ и все модули ввода-вывода, кроме асинхронного таймера и АЦП для минимизации импульсных шумов в процессе преобразования АЦП. В дежурном режиме (Standby) кварцевый/резонаторный генератор продолжают работу, а остальная часть микроконтроллера находится в режиме сна. Данный режим характеризуется малой потребляемой мощностью, но при этом позволяет достичь самого быстрого возврата в рабочий режим. В расширенном дежурном режиме (Extended Standby) основной генератор и асинхронный таймер продолжают работать.

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.

Микроконтроллеры производятся по технологии высокоплотной энергонезависимой памяти компании Atmel. Встроенная внутрисистемно программируемая флэш-память позволяет перепрограммировать память программ непосредственно внутри системы через последовательный интерфейс SPI с помощью простого программатора или с помощью автономной программы в загрузочном секторе. Загрузочная программа может использовать любой интерфейс для загрузки прикладной программы во флэш-память. Программа в загрузочном секторе продолжает работу в процессе обновления прикладной секции флэш-памяти, тем самым поддерживая двухоперационность: чтение во время записи. За счет сочетания 8-разр. RISC ЦПУ с внутрисистемно самопрограммируемой флэш-памятью в одной микросхеме ATmega128 является мощным микроконтроллером, позволяющим достичь высокой степени гибкости и эффективной стоимости при проектировании большинства приложений встроенного управления.

ATmega128 и ATmega16 поддерживаются полным набором программных и аппаратных средств для проектирования, в т.ч.: Си-компиляторы, макроассемблеры, программные отладчики/симуляторы, внутрисистемные эмуляторы и оценочные наборы.

##### 1.2.2 Организация памяти

В соответствии с гарвардской архитектурой память AVR-микроконтроллера разделена на две области: память данных и память программ. Кроме того, МК содержит память на ЭСППЗУ для энергонезависимого хранения данных.

Внутрисистемно программируемая флэш-память программ ATmega128 содержит 128 кбайт (16 кбайт у ATmega16) внутренней внутрисистемно перепрограммируемой флэш-памяти для хранения программы. Поскольку все AVR-инструкции являются 16 или 32-разр., то флэш-память организована как 64 кбит х 16. Для программной защиты флэш-память программ разделена на два сектора: сектор программы начальной загрузки и сектор прикладной программы.

Флэш-память характеризуется износостойкостью не менее 10000 циклов запись/стирание. Таблицы констант могут располагаться в пределах всего пространства памяти программ.

ATmega128 – сложный микроконтроллер с большим числом периферийных устройств, которые управляются через 64 ячейки памяти, зарезервированных в кодах операций инструкций IN и OUT. Для расширенной области ввода-вывода в статическом ОЗУ по адресам $60 - $FF необходимо использовать только инструкции ST/STS/STD и LD/LDS/LDD.

В нормальном режиме первые 4352 ячейки памяти данных относятся к файлу регистров, памяти ввода-вывода, расширенной памяти ввода-вывода и встроенному статическому ОЗУ данных. В первых 32 ячейках расположен файл регистров, следующие 64 ячейки занимает стандартная память ввода-вывода, а за ними следуют 160 ячеек расширенной памяти ввода-вывода. Замыкают внутреннюю память данных 4096 ячеек внутреннего статического ОЗУ данных.

Реализовано пять различных способов адресации для охвата всей памяти: прямая, косвенная со смещением, косвенная, косвенная с предварительным декрементом и косвенная с последующим инкрементом. Регистры R26…R31 из файла регистров используются как регистры-указатели для косвенной адресации. Прямая адресация позволяет адресоваться ко всей памяти данных. Косвенная адресация со смещением позволяет адресовать 63 ячейки, начиная с адреса, указанного в регистрах Y или Z. При использовании инструкции косвенной адресации с предварительным декрементом и последующим инкрементом значения адресных регистров X, Y и Z, соответственно декрементируются до или инкрементируются после выполнения инструкции. 32 рабочих регистров общего назначения, 64 регистра ввода-вывода и 4096 байт внутреннего статического ОЗУ данных в ATmega128 доступны с помощью всех этих режимов адресации.

при установке бита TWINT.

#### 1.3 Программатор и схема питания микроконтроллеров

Для подключения программаторов к МК был использован разъем IDC-06MS. Расположение выводов разъема представлено на рисунке 3.

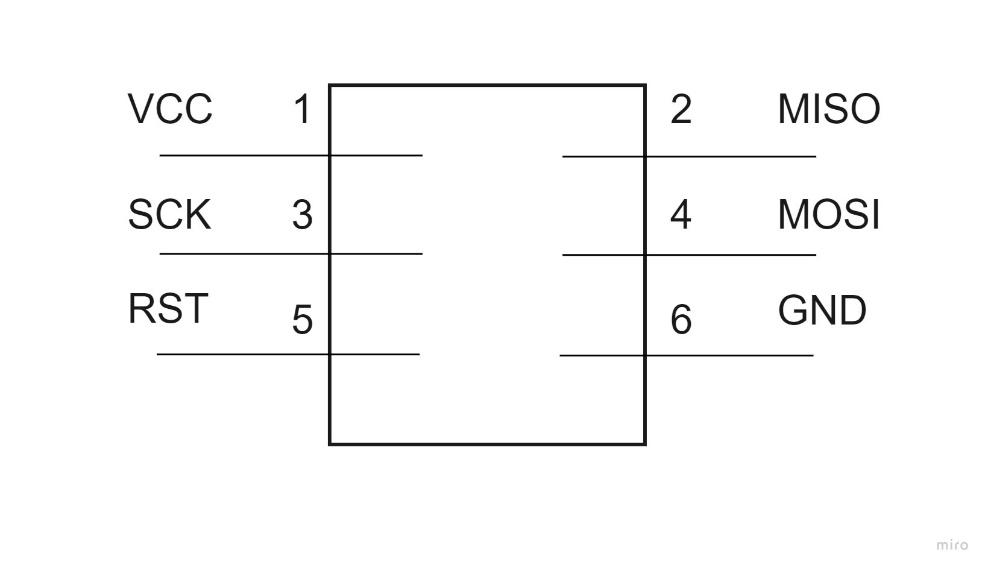


Рисунок 3 – Расположение выводов IDC-06MS

Разъем IDC-06MS используется для программирования микроконтроллера посредством интерфейса SPI. SPI модуль микроконтроллера использует для своей работы 4 вывода: MOSI, MISO, SCK и SS. Когда модуль не задействован, эти выводы являются линиями портов ввода/вывода общего назначения. Назначение данных выводов описано ниже.

SS (chip select) – это ножка выбора устройства. Если на ведомом устройстве на данной ножке установится низкий уровень, то данное устройство будет откликаться и обмениваться информацией по шине SPI, если высокий, то не будет.

MOSI (master output slave input) – это ножка выхода ведущего устройства и входа ведомого устройства.

MISO (master input slave output) – наоборот, выход ведомого, вход ведущего.

SCK – ножка синхронизации. Ко всем устройствам, участвующим в обмене информации по данной шине, подаются синхроимпульсы с определённой частотой.

Микроконтроллеры питаются от напряжения +5 В, а от сети поступает напряжение +12 В, поэтому был использован стабилизатор КР142ЕН5А для преобразования входного напряжения от внешнего блока питания в +5 В для подачи питания на МК.

КР142ЕН5А – стабилизатор напряжения с интегральной схемой с фиксированным напряжением. Помимо использования в качестве стабилизатора постоянного напряжения, это устройство можно использовать с внешними компонентами для получения регулируемых выходных напряжения и тока. У стабилизатора 3 вывода, на первый вывод подается входное напряжение (+12В), второй вывод – это земля, а третий вывод — это выходное напряжение (+5В).

#### 1.4 Пульт управления

Для каждого из трех МК был разработан пульт управления, который состоит из: ЖК-дисплей, кнопка. У ведомого МК кнопка предназначена для переключения очереди, а у ведущих – для получения талона. Для индикации очереди и полученных талонов было использовано три ЖК-дисплея LM044L.

ЖК-дисплей LM044L подключается к AVR микроконтроллеру напрямую к портам. Есть два способа подключения – на 8 бит и на 4 бита. Был использован второй способ подключения.

#### 1.5 Связь между МК

Связь между МК была реализована посредством последовательного интерфейса USART. USART это более гибкий в настройки UART с дополнительными возможностями. В USART можно регулировать длину слова с более большим диапазоном (от 5 до 9) чем в UART (от 8 до 9). В USART 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. Настройка скорости обмена USART вычисляется по формуле 1:

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

где:

XTAL – частота работы МК,

baudrate – требуемая скорость передачи.

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

Таким образом в регистр UBBR записывается значение 51 для установки скорости передачи данных в 9600 бод при частоте работы МК равной 8 МГц.

Для передачи сообщений на дальние расстояния были использованы драйверы MAX485. Микросхема MAX485, производства компании Maxim Integrated, представляет собой маломощный полудуплексный трансивер для обслуживания RS-485 и RS-422 сетей, содержащий раздельные драйверы приемника и передатчика. Особенностью данного приемопередатчика является ограничение скорости нарастания выходного напряжения, что сводит к минимуму электромагнитные помехи и отражения сигнала, вызванные неправильной нагрузкой кабелей связи, что позволяет без ошибок передавать данные со скоростью до 250 кбит/с.

При номинальном напряжении питания 5 В и полной нагрузке, приемник MAX485 потребляет всего 120 мкА, а передатчик – только 250 мкА. Кроме того, в режиме отключения потребление падает до 0,1 мкА. Высокое входное сопротивление приемника позволяет соединять до 128 однотипных трансиверов на общей шине.

Драйвер передатчика MAX485 имеет защиту от короткого замыкания и перегрева, которая переводит его выводы в состояние высокого импеданса. Повышение отказоустойчивости приемника, в случае обрыва во входной цепи, достигается переводом его выхода в состояние логической единицы. Конфигурация выводов драйвера MAX485представлена на рисунке 7.

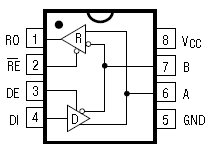


Рисунок 4 – Драйвер MAX485

#### 1.6 Расчет потребляемой мощности

Для определения потребляемой мощности МК была использована формула 2:

|  |  |
| --- | --- |
|  | (2) |

Где, – ток на выводах МК равный 10 мА, – напряжение питания равное 5В, N – число задействованных выводов МК равное 11. Рассчитаем потребляемую мощность ведомого МК: Вт. Рассчитаем потребляемую мощность ведущего МК: Вт. Далее рассчитаем суммарную потребляемую мощность остальных устройств системы (таблица 1).

Таблица 1 – Потребляемая мощность устройств системы

|  |  |  |  |
| --- | --- | --- | --- |
|  | Потребляемая мощность , Вт | Количество микросхем | Суммарная потребляемая мощность , Вт |
| Резистор R1-R6 | 0,125 | 6 | 0,75 |
| LM044L | 0,05 | 3 | 0,15 |

Суммарная мощность всей системы равна:

### 1.7 Описание алгоритмов основных программных модулей

#### 1.7.1 Главные процедуры ведомого и ведущих МК

Перед переходом в основную часть программы необходима инициализация всех используемых компонентов МК: USART, дисплея, портов. После инициализации в ведомом МК происходит опрос кнопки и отображение очереди, а в ведущем – опрос кнопки. Схемы алгоритмов главных процедур изображены на рисунке 4.

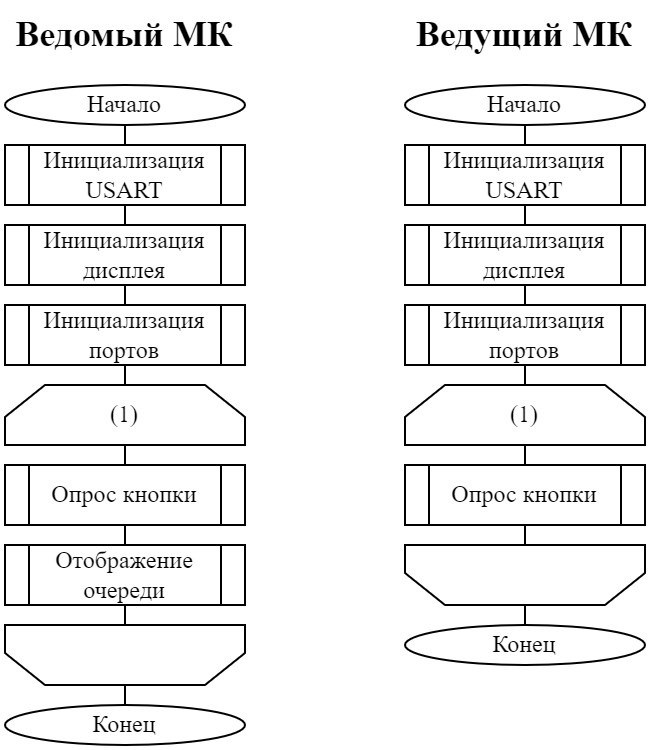


Рисунок 4 – Схемы алгоритмов главных процедур

#### 1.7.2 Процедура отображения очереди на дисплее

Для динамического отображения очереди была использована переменная, которая хранит номер текущего талона. Относительно этой переменной происходит отрисовка очереди на дисплее. Схема алгоритма отображения очереди на дисплее изображена на рисунке 5.

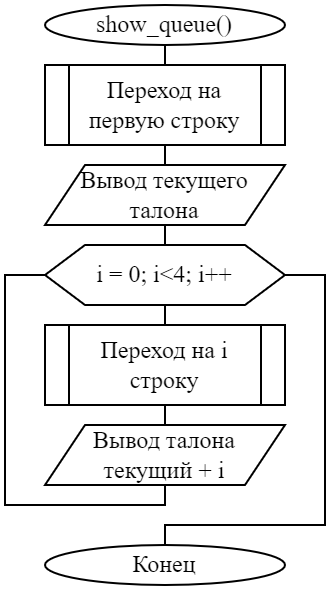


Рисунок 5 – Схема алгоритма отображения очереди на дисплее

#### 1.7.3 Процедура опроса кнопки ведущего МК

Для опроса кнопки на статусный регистр порта кнопки накладывается битовая маска 0b00000001, а после результат инвертируется. Если получилась единица (true), то происходит инкрементации переменной currentCoupon (текущий талон). Схема алгоритма процедуры опроса кнопки изображена на рисунке 6.

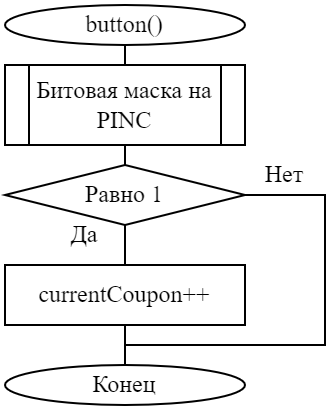


Рисунок 6 – Схема алгоритма отображения очереди на дисплее

## 2 Технологическая часть

### 2.1 Характеристика использованных систем для разработки и отладки программ

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

* Microchip Studio – интегрированная среда разработки программного обеспечения для микроконтроллеров семейства AVR фирмы Atmel.
* Proteus – для симуляции и тестирования проекта.

### 2.2 Тестирование устройства в симуляторе Proteus

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

* выдача талонов на двух ведущих МК;
* отображение очереди на ведомом МК;
* движение очереди на ведомом МК.

На рисунке 7 изображена схема устройства в симуляторе Proteus.

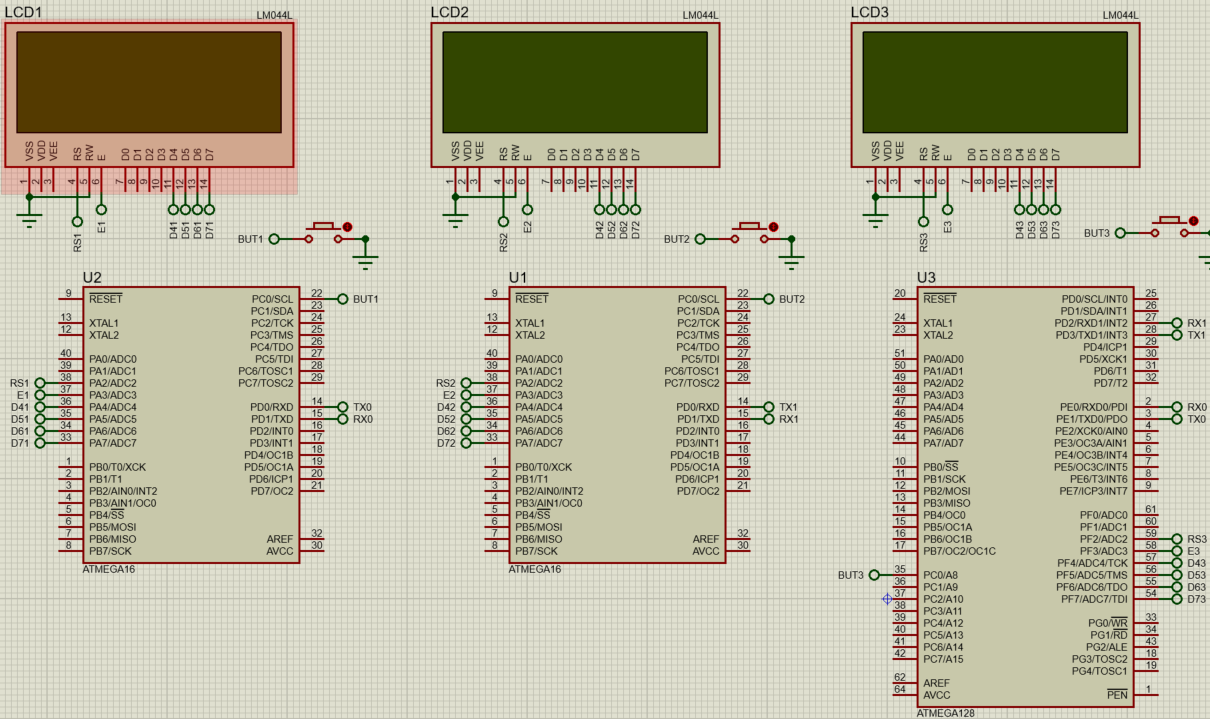


Рисунок 7 – Схема устройства в симуляторе Proteus

На рисунке 8 изображена выдача талонов на ведущих МК.

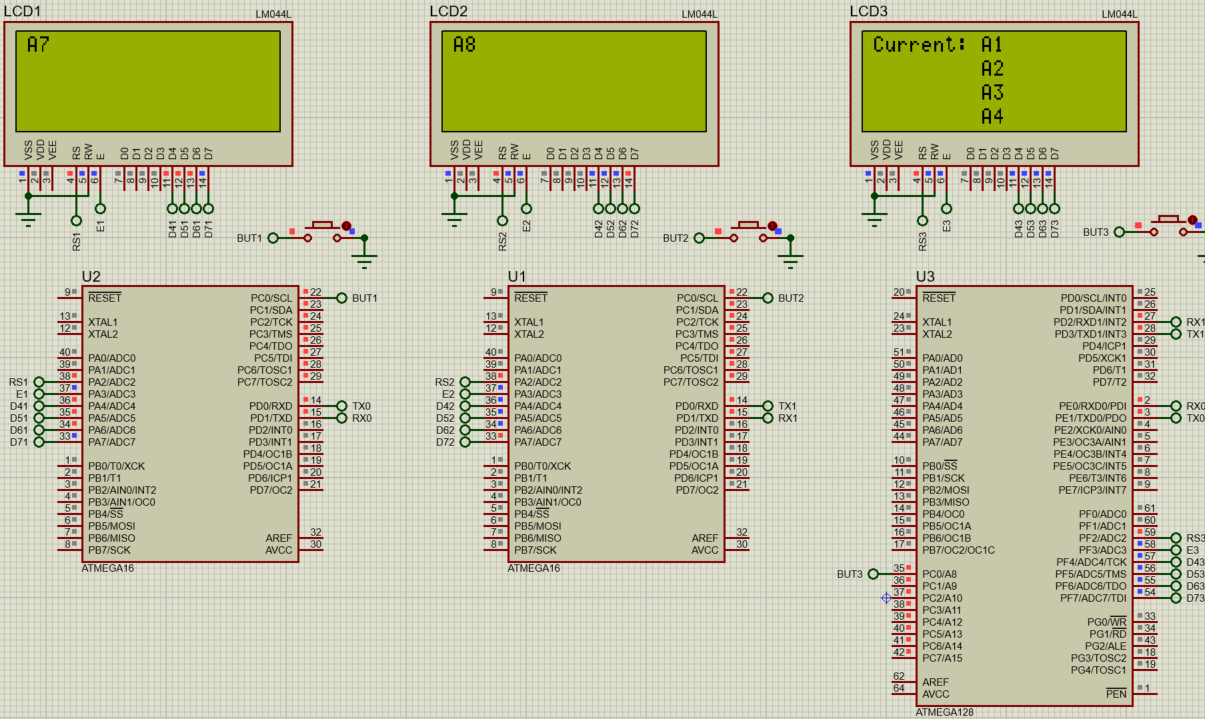


Рисунок 8 – Выдача талонов

На рисунке 9 изображено движение очереди на ведомом МК.

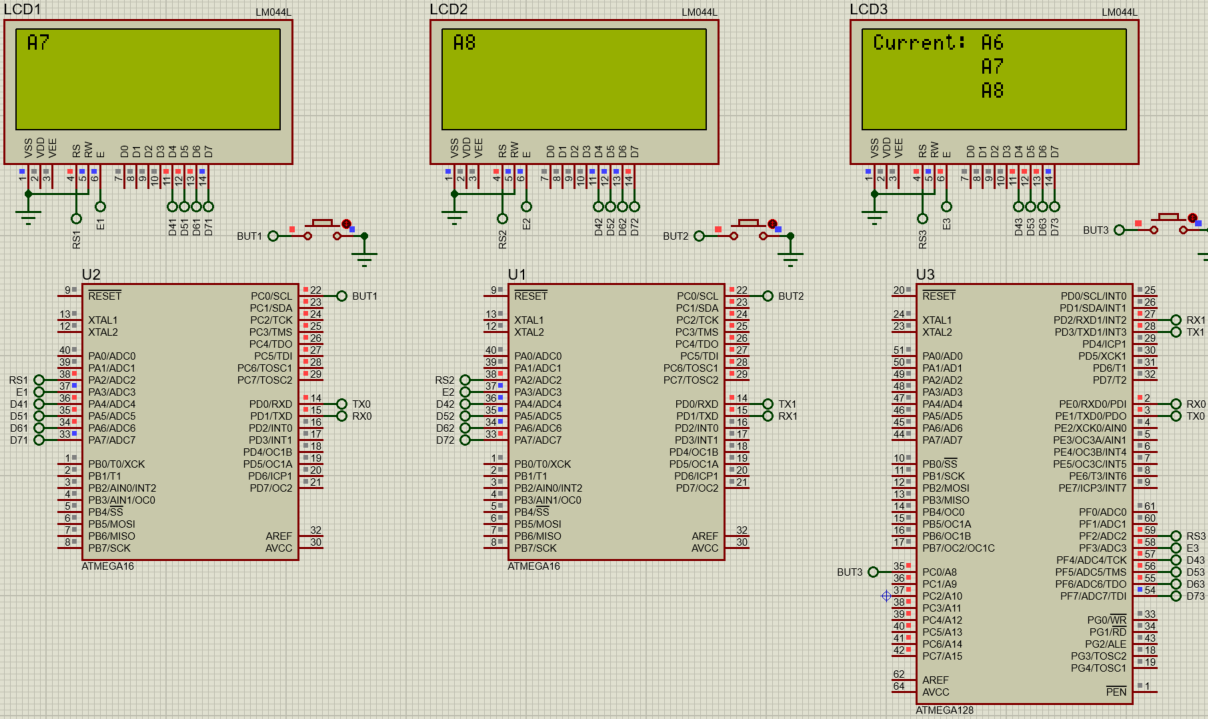


Рисунок 9 – Движение очереди

### 2.3 Программирование микроконтроллера

Самым популярным способом запрограммировать современный микроконтроллер является внутрисхемное программирование (ISP). Внутрисхемным метод называют из-за того, что микроконтроллер в момент программирования находится в схеме целевого устройства.

У МК AVR прошивка передается по интерфейсу SPI, а для работы программатора нужно четыре линии и питание. Разъем для программирования должен иметь следующие контакты для соединения с МК:

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

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.

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 10. Данные команды являются универсальными для микроконтроллеров AVR.

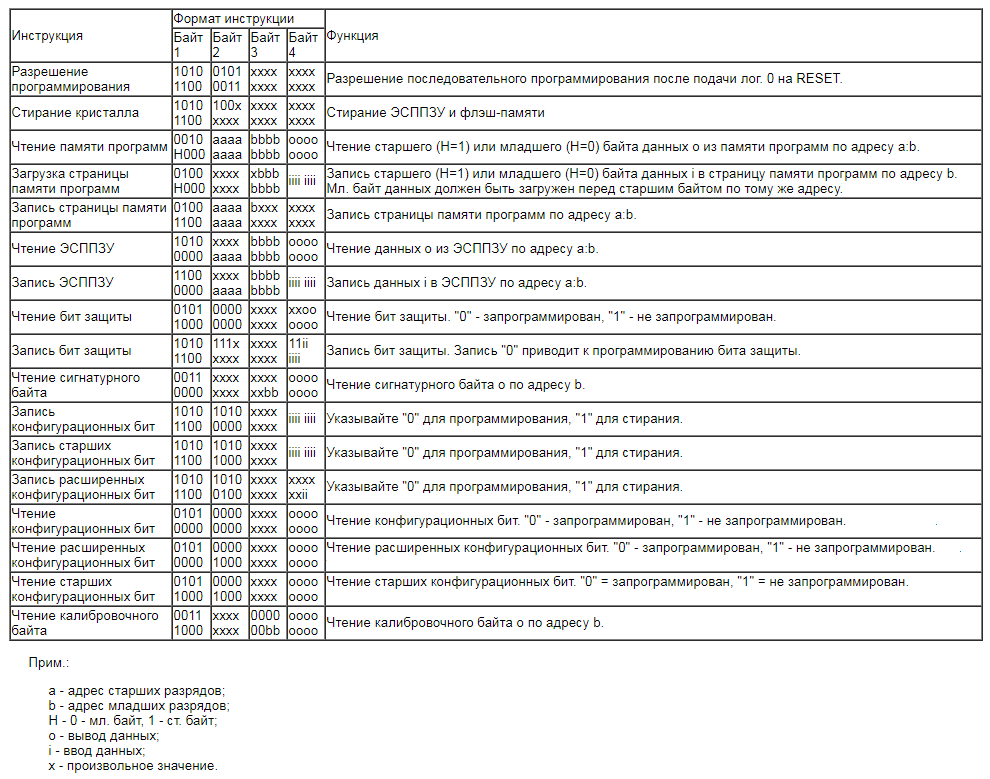


Рисунок 10 – Набор инструкций

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 ProgrammingEnable 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. Страница памяти программ сохраняется путем загрузки инструкции страницы памяти WriteProgram с 7MSB адреса.

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 была реализована система управления потоком посетителей на основе микроконтроллеров ATmega128 и ATmega16. Система состоит из трех устройств, расположенных в разных местах здания.

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 курсовой проект, и может использоваться в качестве системы управления потоком посетителей.

# 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. ГОСТ 2.702-2011 Правила выполнения электрических схем
4. ГОСТ 2.721-74 Обозначения условные графические в схемах. Обозначения общего применения

# ПРИЛОЖЕНИЕ А. Текст исходной программы

Ведомый МК

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

#define BAUDRATE 9600L //скорость передачи данных по usart

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define LCD\_DDR DDRF //порт дисплея

#define LCD\_PORT PORTF //порт дисплея

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

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

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

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

#define B\_DDR DDRC

#define B\_PORT PORTC

#define B\_PIN PINC

#define BUT 0

char data0[32]; //массив символов для приема по usart0

char data1[32]; //массив символов для приема по usart1

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

char queue[100][3] = {0}; //массив для хранения очереди

uint8\_t countCoupons = 0; //кол-во талонов

uint8\_t currentCoupon = 0; //текущий талон

const char commandGetCount[] = "get count\r";

static int LCD\_putchar(char c, FILE \*stream);

static int USART0\_putchar(char c, FILE \*stream);

static int USART1\_putchar(char c, FILE \*stream);

static FILE lcd = FDEV\_SETUP\_STREAM(LCD\_putchar, NULL, \_FDEV\_SETUP\_WRITE);

static FILE usart0 = FDEV\_SETUP\_STREAM(USART0\_putchar, NULL, \_FDEV\_SETUP\_WRITE);

static FILE usart1 = FDEV\_SETUP\_STREAM(USART1\_putchar, NULL, \_FDEV\_SETUP\_WRITE);

//отправка полбайта в дисплей

void LCD\_sendhalfbyte(unsigned char c)

{

c <<= 4;

E1; //включение линии Е

\_delay\_us(50);

LCD\_PORT &= 0b00001111;

LCD\_PORT |= c;

E0; //выключение линии Е

\_delay\_us(50);

}

//отправка байта в дисплей

void LCD\_sendbyte(unsigned char c, unsigned char mode)

{

if (mode == 0)

{

RS0;

}

else

{

RS1;

}

unsigned char hc = 0;

hc = c >> 4;

LCD\_sendhalfbyte(hc);

LCD\_sendhalfbyte(c);

}

//отправка символа в дисплей

void LCD\_sendchar(unsigned char c)

{

LCD\_sendbyte(c, 1);

}

//отправка символа по usart

static int LCD\_putchar(char c, FILE \*stream)

{

if (c == '\n')

{

LCD\_putchar('\r', stream);

}

LCD\_sendchar(c);

return 0;

}

//установка координат каретки на дисплее

void LCD\_setpos(unsigned char x, unsigned y)

{

switch(y)

{

case 0:

LCD\_sendbyte(x | 0x80, 0);

break;

case 1:

LCD\_sendbyte((0x40 + x) | 0x80, 0);

break;

case 2:

LCD\_sendbyte((0x14 + x) | 0x80, 0);

break;

case 3:

LCD\_sendbyte((0x54 + x) | 0x80, 0);

break;

}

}

//инициализация дисплея

void LCD\_init(void)

{

stdout = &lcd;

\_delay\_ms(15);

LCD\_sendhalfbyte(0b00000011);

\_delay\_ms(4);

LCD\_sendhalfbyte(0b00000011);

\_delay\_us(100);

LCD\_sendhalfbyte(0b00000011);

\_delay\_ms(1);

LCD\_sendhalfbyte(0b00000010);

\_delay\_ms(1);

LCD\_sendbyte(0b00101000, 0);

\_delay\_ms(1);

LCD\_sendbyte(0b00001100, 0);

\_delay\_ms(1);

LCD\_sendbyte(0b00000110, 0);

\_delay\_ms(1);

}

//отчистка дисплея

void LCD\_clear(void)

{

LCD\_sendbyte(0b00000001, 0);

\_delay\_us(1500);

}

//Инициализация USART0

void USART0\_init()

{

stderr = &usart0;

UBRR0L=51;

UCSR0B=0x18;

UCSR0C=0x06;

UCSR0B |= (1<<RXCIE);

}

//Инициализация USART1

void USART1\_init()

{

stdin = &usart1;

UBRR1L=51;

UCSR1B=0x18;

UCSR1C=0x06;

UCSR1B |= (1<<RXCIE);

}

//вывод символа по usart0

static int USART0\_putchar(char c, FILE \*stream)

{

if (c == '\n')

USART0\_putchar('\r', stream);

while(!(UCSR0A & (1<<UDRE)));

UDR0 = c;

return 0;

}

//вывод символа по usart1

static int USART1\_putchar(char c, FILE \*stream)

{

if (c == '\n')

USART1\_putchar('\r', stream);

while(!(UCSR1A & (1<<UDRE)));

UDR1 = c;

return 0;

}

//прием данных по usart0

void USART0\_receiving()

{

memset(data0, 0, sizeof data0);

int i = 0;

do

{

while(!(UCSR0A&(1<<RXC)));

data0[i] = UDR0;

i++;

} while (data0[i-1] != '\r');

}

//прием данных по usart0

void USART1\_receiving()

{

memset(data1, 0, sizeof data1);

int i = 0;

do

{

while(!(UCSR1A&(1<<RXC)));

data1[i] = UDR1;

i++;

} while (data1[i-1] != '\r');

}

//прерывание usart0

ISR(USART0\_RX\_vect)

{

USART0\_receiving();

if (strcmp(data0, commandGetCount) == 0)

{

fprintf(stderr, "%d%d\r", countCoupons / 10, countCoupons % 10);

}

else

{

queue[countCoupons][0] = data0[0];

queue[countCoupons][1] = data0[1];

queue[countCoupons][2] = data0[2];

countCoupons++;

if (countCoupons == 99)

{

countCoupons = 0;

}

}

}

//прерывание usart1

ISR(USART1\_RX\_vect)

{

USART1\_receiving();

if (strcmp(data1, commandGetCount) == 0)

{

fprintf(stdin, "%d%d\r", countCoupons / 10, countCoupons % 10);

}

else

{

queue[countCoupons][0] = data1[0];

queue[countCoupons][1] = data1[1];

queue[countCoupons][2] = data1[2];

countCoupons++;

if (countCoupons == 99)

{

countCoupons = 0;

}

}

}

//отображение очереди на дисплее

void show\_queue()

{

LCD\_setpos(0, 0);

printf("Current: %c%c%c", queue[currentCoupon][0], queue[currentCoupon][1], queue[currentCoupon][2]);

for (int i = 1; i < 4; i++)

{

LCD\_setpos(0, i);

printf("%10c%c%c", queue[currentCoupon + i][0], queue[currentCoupon + i][1], queue[currentCoupon + i][2]);

}

}

//нажатие кнопки

void button()

{

if (!(B\_PIN&(0x01<<BUT)))

{

while (!(B\_PIN&(0x01<<BUT)));

currentCoupon++;

if (currentCoupon == 99)

{

currentCoupon = 0;

}

}

}

int main(void)

{

USART0\_init();

USART1\_init();

LCD\_DDR = 0b11111100;

LCD\_PORT = 0x00;

LCD\_init();

B\_DDR = 0x00;

B\_PORT = 0xff;

sei();

while (1)

{

button();

show\_queue();

}

}

Ведущий МК

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

#define BAUDRATE 9600L //скорость передачи данных по usart

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define LCD\_DDR DDRA //порт дисплея

#define LCD\_PORT PORTA //порт дисплея

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

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

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

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

#define B\_DDR DDRC

#define B\_PORT PORTC

#define B\_PIN PINC

#define BUT 0

char data[32]; //массив символов для приема по usart

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

uint8\_t count; //полученное кол-во талонов

static int LCD\_putchar(char c, FILE \*stream);

static int USART\_putchar(char c, FILE \*stream);

static FILE lcd = FDEV\_SETUP\_STREAM(LCD\_putchar, NULL, \_FDEV\_SETUP\_WRITE);

static FILE usart = FDEV\_SETUP\_STREAM(USART\_putchar, NULL, \_FDEV\_SETUP\_WRITE);

//отправка полбайта в дисплей

void LCD\_sendhalfbyte(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\_sendbyte(unsigned char c, unsigned char mode)

{

if (mode == 0)

{

RS0;

}

else

{

RS1;

}

unsigned char hc = 0;

hc = c >> 4;

LCD\_sendhalfbyte(hc);

LCD\_sendhalfbyte(c);

}

//отправка символа в дисплей

void LCD\_sendchar(unsigned char c)

{

LCD\_sendbyte(c, 1);

}

//отправка символа по usart

static int LCD\_putchar(char c, FILE \*stream)

{

if (c == '\n')

{

LCD\_putchar('\r', stream);

}

LCD\_sendchar(c);

return 0;

}

//установка координат каретки на дисплее

void LCD\_setpos(unsigned char x, unsigned y)

{

switch(y)

{

case 0:

LCD\_sendbyte(x | 0x80, 0);

break;

case 1:

LCD\_sendbyte((0x40 + x) | 0x80, 0);

break;

case 2:

LCD\_sendbyte((0x14 + x) | 0x80, 0);

break;

case 3:

LCD\_sendbyte((0x54 + x) | 0x80, 0);

break;

}

}

//инициализация дисплея

void LCD\_init(void)

{

stdout = &lcd;

\_delay\_ms(15);

LCD\_sendhalfbyte(0b00000011);

\_delay\_ms(4);

LCD\_sendhalfbyte(0b00000011);

\_delay\_us(100);

LCD\_sendhalfbyte(0b00000011);

\_delay\_ms(1);

LCD\_sendhalfbyte(0b00000010);

\_delay\_ms(1);

LCD\_sendbyte(0b00101000, 0);

\_delay\_ms(1);

LCD\_sendbyte(0b00001100, 0);

\_delay\_ms(1);

LCD\_sendbyte(0b00000110, 0);

\_delay\_ms(1);

}

//отчистка дисплея

void LCD\_clear(void)

{

LCD\_sendbyte(0b00000001, 0);

\_delay\_us(1500);

}

//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)|(3<<UCSZ0); //8 бит

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

}

//отправка символа по usart

static int USART\_putchar(char c, FILE \*stream)

{

if (c == '\n')

{

USART\_putchar('\r', stream);

}

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

UDR = c;

return 0;

}

//прием данных по usart

void USART\_receiving()

{

memset(data, 0, sizeof data);

int i = 0;

do

{

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

data[i] = UDR;

i++;

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

}

//прерывание usart

ISR(USART\_RXC\_vect)

{

USART\_receiving();

count = ((data[0]&0b00001111) \* 10) + (data[1]&0b00001111);

fprintf(stderr, "A%d\r", count + 1);

LCD\_clear();

LCD\_setpos(0, 0);

fprintf(stdout, "A%d", count + 1);

}

//нажатие кнопки

void button()

{

if (!(B\_PIN&(0x01<<BUT)))

{

while (!(B\_PIN&(0x01<<BUT)));

fprintf(stderr, "get count\r");

}

}

int main(void)

{

LCD\_DDR = 0b11111100;

LCD\_PORT = 0x00;

B\_DDR = 0x00;

B\_PORT = 0xff;

USART\_init();

LCD\_init();

sei();

while (1)

{

button();

}

}

# ПРИЛОЖЕНИЕ Б. Спецификация радиоэлементов схемы

