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:

Контроллер КПП грузового транспорта

Студент ИУ6И-71Б (Группа) _____ (Подпись, дата) Г. Мэндбаяр (И.О. Фамилия)

Руководитель _____ (Подпись, дата) Е.В. Червяков (И.О. Фамилия)

2024 г.

Лист задания

РЕФЕРАТ

РПЗ 66 страниц, 33 рисунок, 3 таблицы, 6 источников, 3 приложения.

МИКРОКОНТРОЛЛЕР, СИСТЕМА, КОНТРОЛЛЕР КПП, МАТРИЧНАЯ КЛАВИАТУРА, ЗУММЕР

Объектом разработки является контроллер КПП грузового транспорта.

Цель работы – проектирование и моделирование работы, а также разработка устройства контроллера КПП грузового транспорта.

Поставленная цель достигается посредством использования Proteus 8.

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

Результатом проектирования является устройство для контроллера КПП грузового транспорта.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК – микроконтроллер.

ТЗ – техническое задание.

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

ASCII – таблица кодировки символов.

EEPROM – electrically Erasable Programmable Read-Only Memory – электрически стираемое перепрограммируемое ПЗУ (ЭСППЗУ), один из видов энергонезависимой памяти.

ВВЕДЕНИЕ

В данной работе производится разработка контроллера КПП грузового транспорта.

В процессе выполнения работы проведён анализ технического задания, создана концепция устройства, разработаны электрические схемы, построен алгоритм и управляющая программа для МК.

Система состоит из МК, дисплея, зуммера и матричной клавиатуры. Контроллер КПП предназначен для управления доступом грузового транспорта на территорию, а также для проверки соответствия вводимых данных установленным критериям. Система позволяет вводить данные о транспортных средствах и проверять их на соответствие заранее заданным параметрам.

Актуальность работы заключается в необходимости создания компактных и функциональных решений, которые могут быть использованы в различных областях, таких как аналитика, образование или управление информацией. Предложенное устройство демонстрирует возможность реализации доступного и универсального решения, которое может быть адаптировано для других задач с минимальными изменениями.

1 Анализ требований

Исходя из требований технического задания, результатом работы устройства является устройство для контроллера КПП, включающее в себя контроллер, дисплей, устройство ввода и средство хранения информации.

1.1 Принцип работы разрабатываемого устройства

Разработанное устройство предназначено для ввода, обработки и отображения данных о данных грузового транспорта. Основным компонентом устройства является микроконтроллер AtMega8515, который управляет всеми процессами. Работа устройства состоит из нескольких этапов:

1) Ввод данных:

Пользователь вводит информацию с помощью матричной клавиатуры, подключенной к портам PC0–PC6 микроконтроллера. Клавиатура позволяет вводить гос. номер транспорта, его массу с грузом и без груза. Каждый символ обрабатывается микроконтроллером с учетом механизма подавления дребезга контактов, чтобы гарантировать корректность данных.

2) Обработка данных:

После ввода данные анализируются. Если пользователь вводит гос номер неправильно, устройство перестает работать, не позволяя пользователю перейти к следующему вводу. Если входные данные верны, данные сохраняются во время выполнения и готовы к сравнению с данными в базе данных.

3) Вывод информации:

Результаты обработки отображаются на ЖК-дисплее. Например, пользователю выводится сообщение про что совпадают ли данные или нет. Дисплей подключен к портам PA0-PA2 PA4–PA7 и управляется сигналами RS и E, что позволяет визуализировать данные в удобной форме.

4) Связь с внутренней базой данных:

Для хранения и обработки данных устройство взаимодействует с внутренней базой данных через интерфейс EEPROM. Данные записываются в базу данных с момента запуска программы и считываются после этого действии.

1.2 Разработка структурной схемы устройства

Из анализа требований можно выделить следующие структурные элементы устройства: матричная клавиатура, микроконтроллер ATMega8515, LCD-дисплей и зуммер.

Микроконтроллер получает данные, который были введены с матричной клавиатуры, далее отправляет или запрашивает данные от сервера с помощью eeprom, которые впоследствии выводит на LCD-дисплей. Структурная схема устройства представлена на рисунке 1:

Рисунок 1 – структурная схема устройства

2 Проектирование функциональной схемы устройства

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

2.1 Микроконтроллер ATMega8515

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

- 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 1.

Таблица 1 – 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.

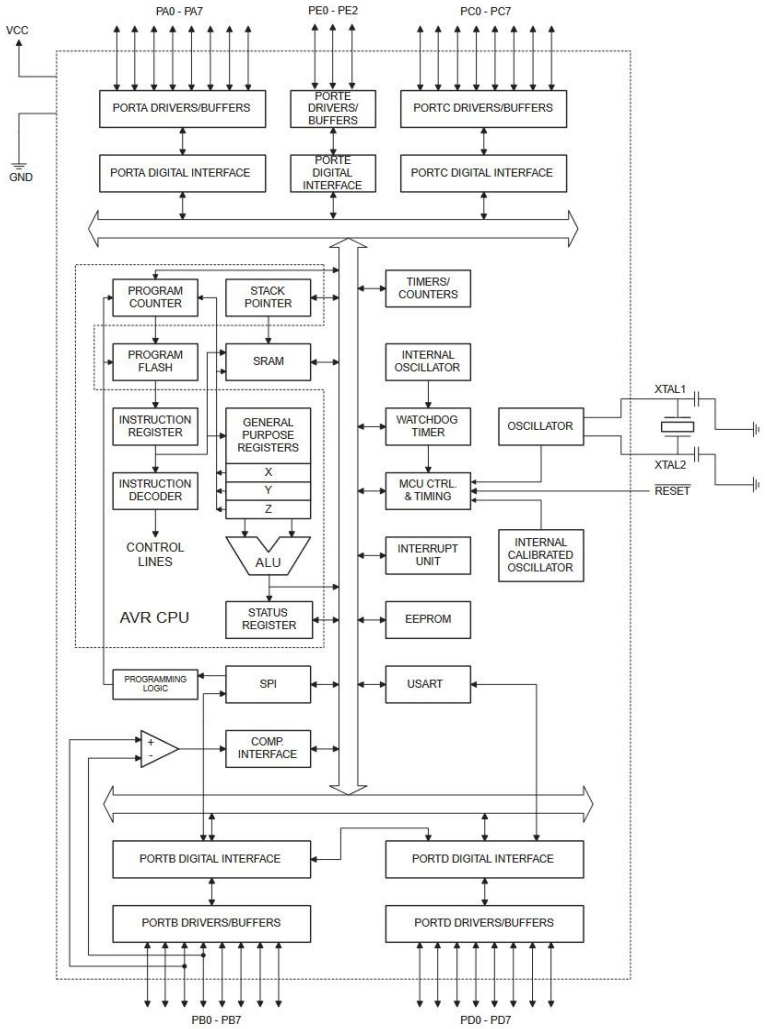

Рисунок 2 – структурная схема ATMega8515

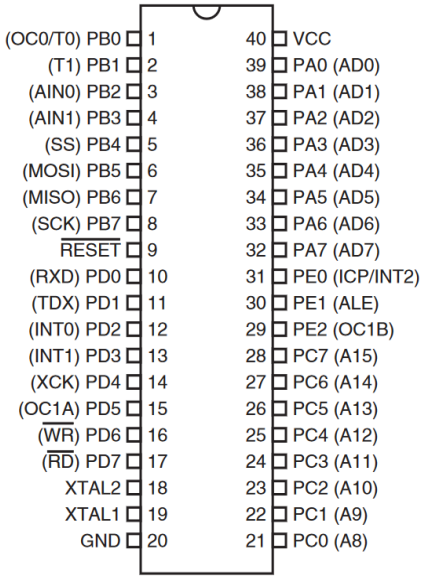

Рисунок 3 – УГО МК ATmega8515

Он обладает следующими характеристиками:

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.

2.1.1 Используемые элементы

1) Порты ввода-вывода (GPIO)

- PC0–PA6 были задействованы для подключения клавиатурной матрицы. Входы и выходы портов использовались для сканирования строк и столбцов клавиатуры, обеспечивая считывание символов.
- PA0-PA2 PA4-PA7 подключены к ЖК-дисплею (соответственно для сигналов RS и Enable).
- PD7 был выделен для управления зуммером. PD0 и PD1 были выделены для RS232.

- PB5 – PB7 были выделены деля программатора.

2) Память EEPROM

EEPROM используется для памяти МК, в котором хранится загруженная в него программа.

3) Таймеры и счетчики

- Один из таймеров используется для реализации программного подавления дребезга клавиш.
- Таймер отслеживает временные интервалы между нажатиями, чтобы избежать случайного многократного ввода.
- Также таймер применяется для измерения времени между последовательными нажатиями клавиш с целью изменения отображаемого символа.

4) SRAM используется для временного хранения данных о вводимых символах, текущем состоянии устройства.

5) SPI – интерфейс для связи МК с другими внешними устройствами. В проекте используется только для прошивки МК.

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

7) Регистр команд – содержит исполняемую в текущий момент (или следующий) команду, то есть команду.

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

9) Программный счетчик – регистр процессора, который указывает, какую команду нужно выполнять следующей.

10) Указатель стека – указатель, хранящий значение вершины стека, используется при вызове подпрограмм.

2.1.2 EEPROM

Микроконтроллер Atmega8515 также оснащён встроенной EEPROM (Electrically Erasable Programmable Read-Only Memory), которая позволяет сохранять данные даже при отключении питания. EEPROM в Atmega8515 имеет свои особенности и функции, которые мы рассмотрим ниже.

1. Структура EEPROM:

EEPROM в Atmega8515 имеет объём 512 байт, что позволяет хранить небольшие объёмы данных, такие как настройки, конфигурации или другие важные параметры, которые необходимо сохранять между перезагрузками.

2. Регистры управления EEPROM:

EECR (EEPROM Control Register): Этот регистр управляет операциями чтения и записи в EEPROM. Он содержит управляющие биты и флаги, необходимые для выполнения операций.

EEDR (EEPROM Data Register): Этот регистр используется для хранения данных, которые будут записаны в EEPROM или которые были считаны из него.

EEAR (EEPROM Address Register): Этот регистр используется для указания адреса ячейки EEPROM, с которой будет производиться чтение или запись.

3. Процесс записи данных:

Подготовка к записи: Перед записью данных в EEPROM необходимо установить адрес ячейки, в которую будет производиться запись, в регистре EEAR.

Запись данных: Данные, которые нужно записать, помещаются в регистр EEDR. Затем необходимо установить бит EEWE в регистре EECR, чтобы начать процесс записи. Это инициирует запись данных в указанную ячейку EEPROM.

Завершение записи: После завершения записи бит EEWE сбрасывается, и процесс записи считается завершённым. Важно отметить, что запись в EEPROM занимает некоторое время (обычно несколько миллисекунд), и в это время нельзя выполнять другие операции записи.

4. Процесс чтения данных:

Подготовка к чтению: Для чтения данных из EEPROM необходимо установить адрес ячейки в регистре EEAR.

Чтение данных: После установки адреса можно считать данные из EEPROM, считывая значение из регистра EEDR. Это значение будет содержать данные, хранящиеся в указанной ячейке.

Обработка данных: После чтения данные могут быть использованы в программе, например, для настройки параметров работы микроконтроллера.

5. Ошибки и ограничения:

Количество циклов записи: EEPROM имеет ограниченное количество циклов записи (обычно около 100 000), поэтому следует избегать частой записи в одну и ту же ячейку.

Время записи: Запись в EEPROM занимает больше времени, чем запись в оперативную память, поэтому необходимо учитывать это при проектировании системы.

2.2 Матричная клавиатура

Матричная клавиатура в нашем проекте используется для ввода данных, таких как гос номер транспорта, его масса с грузом и без груза. Она позволяет

эффективно использовать ограниченное количество портов микроконтроллера для обработки большого числа кнопок. В данном проекте клавиатура состоит из 12 кнопок, которые расположены в матрице 3×4.

Матричная клавиатура представляет собой сетку из пересекающихся строк и столбцов. Каждая кнопка находится на пересечении строки и столбца. Нажатие кнопки замыкает соответствующую цепь, соединяя одну из строк с одним из столбцов.

В проекте используется следующая конфигурация:

- Столбцы: Подключены к портам PC4–PC6 микроконтроллера AtMega8515.
- Строки: Подключены к портам PC0–PC2.

Для устранения эффекта "дребезга контактов" используется программный дебаунсинг. После обнаружения нажатия кнопки вводится задержка (2 мс), чтобы убедиться в стабильности сигнала.

Для ввода текста, как в старых кнопочных телефонах, каждая кнопка (за исключением специальных, таких как *, 0, и #) соответствует нескольким символам. Например:

- Кнопка 2: A, B, C
- Кнопка 3: D, E, F
- Кнопка 4: G, H, I
- Кнопка 5: J, K, L
- Кнопка 6: M, N, O
- Кнопка 7: P, Q, R, S
- Кнопка 8: T, U, V
- Кнопка 9: W, X, Y, Z
- Кнопка 0: Пробел

Микроконтроллер сканирует клавиатуру, поочередно устанавливая строки в состояние LOW (низкий уровень) и считывая состояние столбцов. Если кнопка нажата, между строкой и столбцом появляется замыкание, и микроконтроллер фиксирует нажатие.

Кнопка “*” используется для подтверждения пользовательского нажатия.

2.3 LCD-дисплей LM044L

LCD-дисплей используется для отображения вводимых данных и сообщение сравнения. Его структурная схема представлена на рисунке 4:

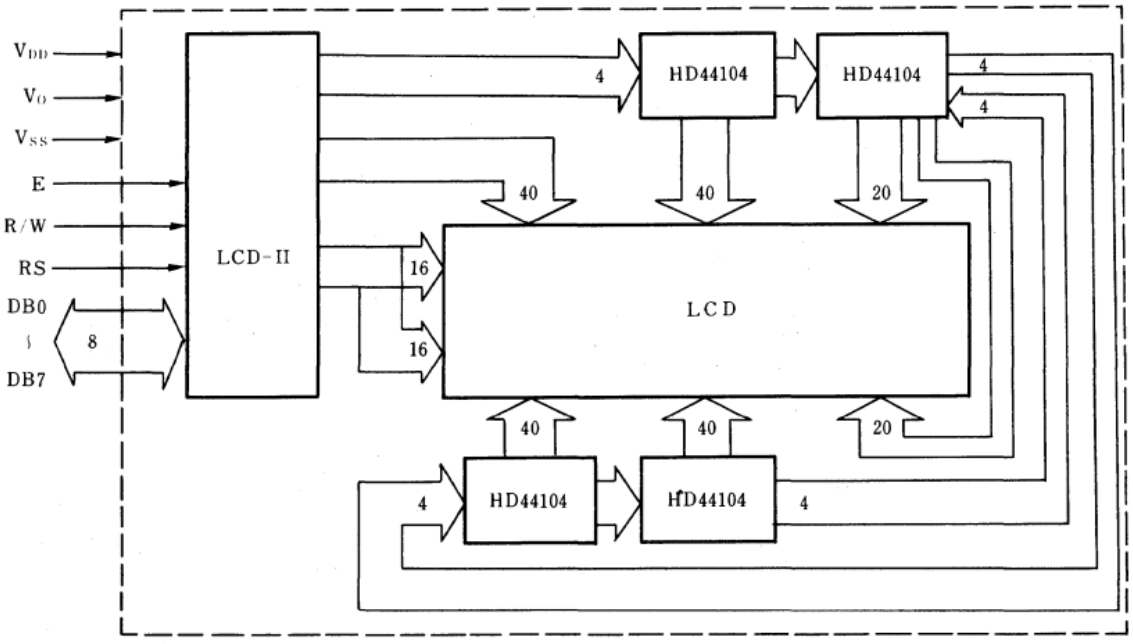

Рисунок 4 – Структурная схема LCD-дисплея LM044L

LCD-дисплей LM044L является важным компонентом нашего проекта, обеспечивающим вывод информации для пользователя. Это устройство предназначено для отображения текстовых и символьных данных и представляет собой модуль с жидкокристаллическим экраном, который можно легко интегрировать с микроконтроллерами, включая AtMega8515. Основным преимуществом дисплея является простота взаимодействия с ним через

стандартный интерфейс, позволяющий управлять отображением данных с минимальными затратами ресурсов.

LM044L поддерживает работу в двух режимах: 4-битном и 8-битном. В рамках нашего проекта используется 4-битный режим, что позволяет экономить выводы микроконтроллера. Для передачи данных и управления используются следующие сигнальные линии: линии данных (D0-D7), линия выбора регистра (RS), линия чтения/записи (RW) и линия включения (E). Линия RS отвечает за выбор типа передаваемой информации: команды или данных. Линия RW определяет направление передачи — чтение или запись, но в нашем проекте используется только режим записи, так как данные считывать с дисплея не требуется. Линия E служит для синхронизации передачи информации. HD44100 — это драйверы дисплея, которые управляют сегментами LCD. В данном случае четыре блока HD44100, каждый из которых контролирует определенную часть дисплея (по 40 сегментов каждый). Эти драйверы работают параллельно, позволяя дисплею отображать больше информации. LCD II — это контроллер дисплея, который управляет передачей данных между микроконтроллером и блоками драйверов.

Для работы с дисплеем LM044L необходимо выполнить несколько этапов настройки. Сначала дисплей инициализируется, чтобы установить нужный режим работы, например, включить 4-битный режим передачи данных, задать количество отображаемых строк и активировать курсор, если это необходимо. После инициализации можно отправлять команды для управления дисплеем, такие как очистка экрана, перемещение курсора или включение/выключение отображения. Далее данные передаются в дисплей для вывода символов, которые пользователь видит на экране.

Физически LM044L представляет собой небольшой экран с 4 строками. Для управления дисплеем в проекте используется библиотека программных функций, обеспечивающая удобное взаимодействие с модулем. Основные команды передаются через функцию, которая задаёт состояние сигнальных

линий и отправляет данные с использованием определённых задержек, чтобы обеспечить стабильность работы устройства.

Отображение символов на дисплее происходит за счёт управления внутренним буфером данных. Каждый символ имеет свой код в таблице ASCII, который передаётся в дисплей. Специальные команды позволяют не только выводить стандартные символы, но и создавать пользовательские, что может быть полезно для специфических задач.

2.4 Зуммер

Звуковой сигнализатор (buzzer) используется для подачи звуковых оповещений микроконтроллером. Он может быть как пассивным, так и активным, в зависимости от принципа работы.

Пассивный зуммер:

- Для работы требуется сигнал с определённой частотой, генерируемый микроконтроллером.
- Микроконтроллер подаёт меандр на вывод буззера, задавая тем самым частоту звука.
- Частота звука может изменяться программно для генерации различных тонов.

Активный зуммер:

- Встроенный генератор обеспечивает фиксированную частоту звука при подаче постоянного напряжения.
- Для активации микроконтроллер подаёт высокий уровень (логическую «1») на вывод буззера.
- Не требует сложного управления – достаточно включить/выключить сигнал.

Вывод зуммера подключается к цифровому пину микроконтроллера. При подаче логической «1» зуммер издаёт звук. Для большей мощности можно использовать транзистор и подключить буззер к внешнему питанию.

2.5 Преобразователь MAX232

Приём данных от ПЭВМ происходит с использованием протокола RS232 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. Обеспечивает передачу данных и некоторых специальных сигналов между терминалом и устройством приема.

Структурная схема драйвера представлена на рисунке 5:

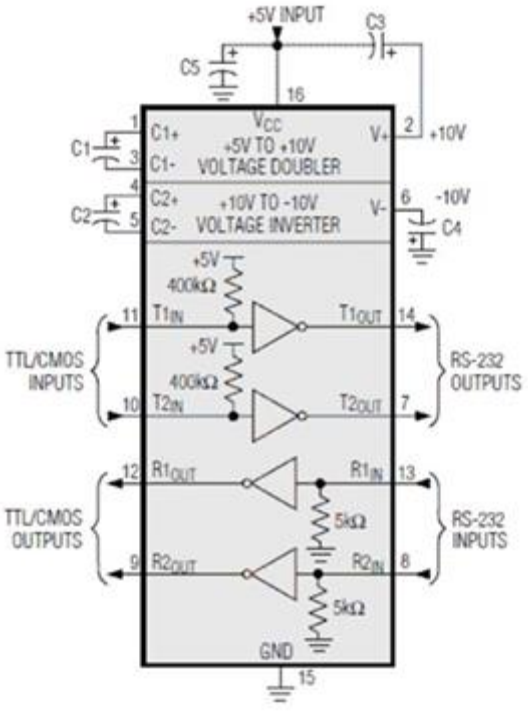

Рисунок 5 – структурная схема MAX232

Преобразователь MAX232 является широко используемым устройством, предназначенным для обеспечения совместимости логических уровней микроконтроллеров с интерфейсом RS-232. Основная задача

MAX232 заключается в преобразовании логических уровней TTL/CMOS, используемых микроконтроллерами, в уровни, соответствующие стандарту RS-232, а также в обратном преобразовании. Это устройство играет ключевую роль в системах, где требуется соединение микроконтроллера с компьютерами или другими устройствами, работающими по интерфейсу RS-232, такими как принтеры, модемы и другие коммуникационные устройства.

Основой работы MAX232 является встроенный генератор, который создает необходимые напряжения для передачи данных по стандарту RS-232. Логические уровни микроконтроллера (0 и 5 В или 0 и 3.3 В) преобразуются в уровни ±12 В, которые требуются для корректного функционирования RS-232. Это достигается за счет встроенного зарядового насоса, который повышает напряжение питания, создавая положительные и отрицательные уровни. Аналогичным образом, при приеме данных устройство понижает уровни RS-232 до логических уровней микроконтроллера. Зарядовые насосы работают благодаря использованию внешних конденсаторов, которые формируют цепь повышения напряжения.

MAX232 поддерживает работу в обоих направлениях: преобразование уровня передачи (TX) и приема (RX). Это означает, что сигнал, поступающий с микроконтроллера через линию передачи данных, усиливается и преобразуется в формат RS-232, в то время как сигнал, полученный с линии приема данных, преобразуется обратно в логический уровень, который может быть обработан микроконтроллером. Таким образом, MAX232 обеспечивает двустороннюю связь, необходимую для большинства приложений.

2.6 Программатор AVR-ISP500

Для программирования МК используется программатор, для его подключения необходим специальный разъем. Будет использован разъем AVR-ISP500. Подключение программатора осуществляется при помощи интерфейса SPI, под что на МК ATmega64 задействован порт PB. 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.

На основе всех вышеописанных сведений была спроектирована функциональная схема разрабатываемого устройства. Полученная схема представлена в приложении Б.

3 Проектирование принципиальной схемы устройства

3.1 Подключение цепи питания

Для подключения микроконтроллера к питанию использовано 1 полярных конденсаторов емкостью 100 нанофарад, 2 емкостью 15 микрофарад, 1 емкостью 10 микрофарад и 2 емкостью 0.1 микрофарад. Больший конденсатор сглаживает низкочастотные колебания напряжения питания, а меньшие конденсаторы более эффективно фильтруют высокочастотный шум на линии питания. Количество меньших конденсаторов подобрано в соответствии с рекомендацией, что один такой конденсатор фильтрует шум двух устройств схемы. Схема подключения изображена на рисунке 6:

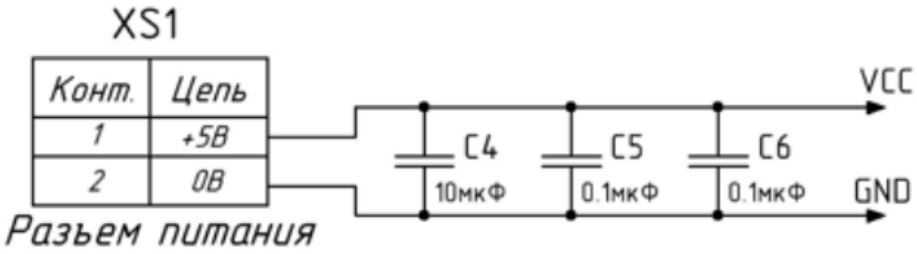

Рисунок 6 – Схема источника питания

3.2 Схема подключения к ПЭВМ

Драйвер последовательного канала MAX232 подключен в соответствии с рекомендуемой схемой включения, представленной в документации к драйверу. Схема подключения представлена на рисунке 7:

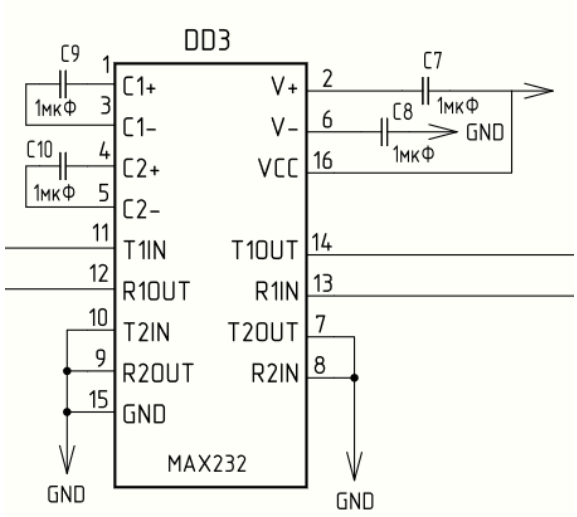

Рисунок 7 - схема подключения MAX232

Драйвер MAX232 используется для обеспечения совместимости уровней напряжения, так как RS-232 требует более высоких напряжений по сравнению с логическими уровнями микроконтроллера.

Чип MAX232 питается от источника напряжения Vcc, подключенного к 16-й ножке микросхемы, а 15-я ножка соединена с общим проводом (землей). Важной особенностью драйвера MAX232 является наличие встроенного преобразователя напряжения, который генерирует более высокие и отрицательные напряжения, необходимые для работы интерфейса RS-232. Для его работы используются внешние конденсаторы.

В схеме видно, что конденсаторы C7 и C8 подключены между ножками Vcc, VS+, и VS−. Конденсатор C7, номиналом 1 мкФ, соединен между выводом VS+ и Vcc, тогда как конденсатор C8, аналогичного номинала, подключен между VS− и общим проводом. Эти конденсаторы стабилизируют напряжение преобразователя.

Дополнительно конденсаторы C9 и C10, также номиналом 1 мкФ, подключены между выводами C1+, C1− и C2+, C2−, что обеспечивает корректную работу внутреннего генератора напряжения.

Для передачи и приема данных используются выводы T1IN, T1OUT, R1IN и R1OUT. Вывод T1IN принимает сигналы с уровнями TTL от

микроконтроллера, а T1OUT выдает сигналы в уровнях RS-232, которые могут быть отправлены на компьютер. Аналогично, R1IN принимает сигналы от компьютера в уровнях RS-232, а R1OUT передает их обратно в уровнях TTL. Данные линии соединяются с последовательным портом компьютера через стандартный разъем, обеспечивая обмен информацией.

Для подключения устройства к ПЭВМ используется COM-порт стандарта DB-9, который на схеме изображен как элемент с условным обозначением XP2, рисунок 8:

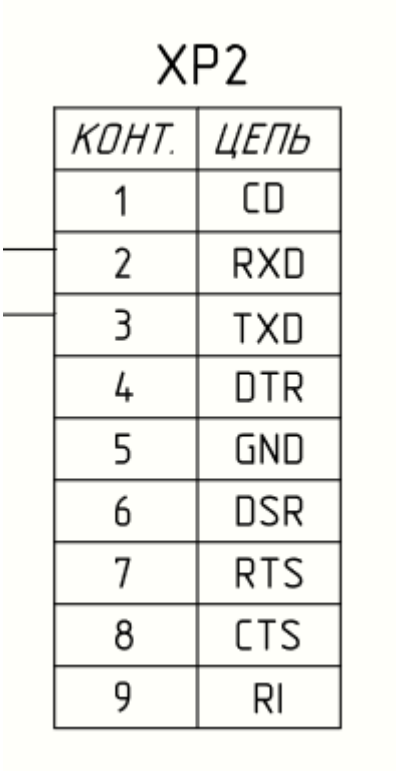

Рисунок 8 – Разъём DB-9

Таблица 2 – Выводы разъема DB-9

Аббревиатура	Расшифровка
DCD	Data Carrier Detect
RXD	Receive Data (I SERIAL INPUT)

TXD	Transmit Data (I SERIAL OUTPUT)
DTR	Data Terminal Ready
GND	Ground
DSR	Data Set Ready
RTS	Request To Send
CTS	Clear To Send
RI	Ring Indicator

3.3 Схема подключения дисплея

Схема подключения дисплея представлена на рисунке 9:

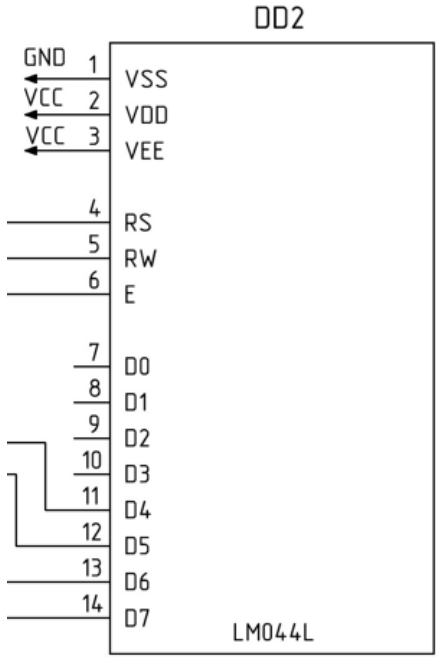

Рисунок 9 – Схема подключения дисплея

При данном подключении используется 4 входов по данным – при таком соединении обеспечивается наивысшая скорость передачи данных.

Входы управления E, отвечающий за разрешение работы дисплея, а также RS, определяющий тип входных данных: код символа или команда, управляются микроконтроллером. Вход управления RW заземлен, так как дисплей используется только в одном режиме – записи. На схеме также изображен потенциометр, меняя сопротивление, которого можно регулировать напряжения, подаваемое на ЖКИ, а также, соответственно, и яркость.

3.4 Схема подключения к зуммеру

Зуммер подключен в соответствии с рекомендуемой схемой включения, представленной в документации. Схема подключения представлена на рисунке 10:

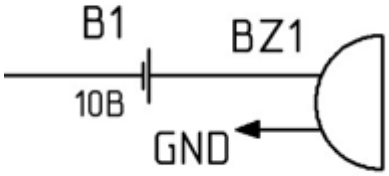

Рисунок 10 – схема подключения зуммера

Один вывод буззера подключен к источнику питания (через компонент B1), а второй вывод соединён с GND (землёй). Это стандартное подключение для активного буззера, который начинает издавать звук сразу после подачи питания.

3.5 Расчёт 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. Мощность для резисторов рассчитывается по формуле: $P = I^2 * R$, где R – сопротивление резистора; I – ток, проходящий через резистор.

Расчет потребляемого напряжения для каждой микросхемы показан в таблице 3.

Таблица 3 – Потребляемая мощность

Микросхема	Ток потребления, мА	Потребляемая мощность, мВт	Количество устройств	Суммарная потребляемая мощность, мВт
Atmega8515	40	200	1	200
Батерей	5	10	1	10
MAX232	8	40	1	40
LM044L	3	15	1	15
Матричная клавиатура	20	12	1	12

Также в схеме используются 7 резисторов МО-100 номиналом 1 кОм и 1 резистор номиналом 10кОм.

$P_{суммарная} = P_{Atmega8515} + P_{батерей} + P_{max232} + P_{LM044L} + P_{клавиатура} + P_{резисторов} = 200 + 10 + 40 + 15 + 12 + 17 = 294$ мВт

Суммарная потребляемая мощность системы равна 294 мВт.

4 Разработка алгоритмов

4.1 Алгоритм выполнения основной программы

В основной программе происходит настройка портов ввода-вывода для матричной клавиатуры, LCD и EEPROM. Далее проводится инициализация дисплея и EEPROM. В 3 бесконечных циклах:

1) Вводится гос номер, используя матричную клавиатуру и дисплей.
2) Вводится масса без груза, используя матричную клавиатуру и дисплей.
3) Вводится масса с грузом, используя матричную клавиатуру и дисплей.

Схема алгоритма основной программы представлена на рисунке 11:

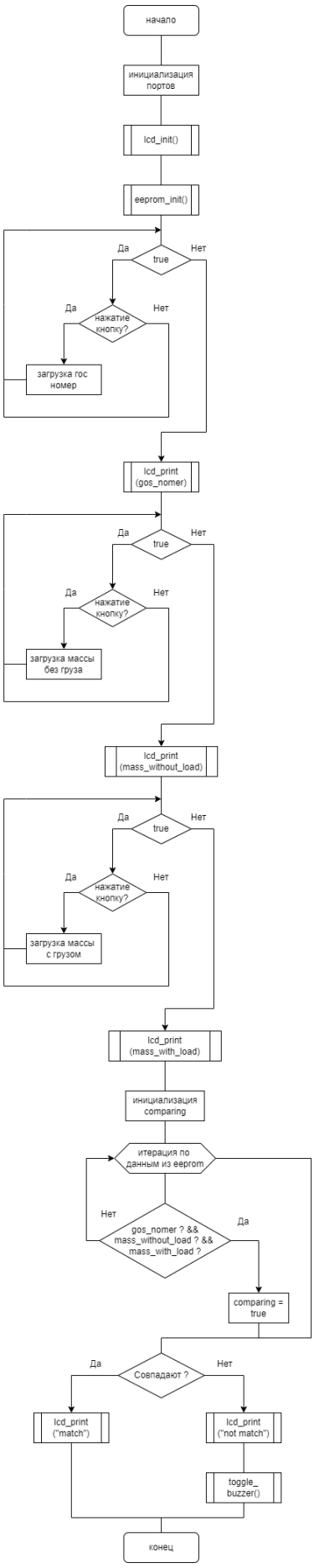

Рисунок 11 – Схема алгоритма основной программы

4.2 Алгоритм функции работы с матричной клавиатурой

Соответствующая строка клавиатуры устанавливается в активное состояние путем установки значения на порт PORTC.

Далее проверяется, какая клавиша в активной строке нажата, анализируя значения на входном порте PINC.

Схема алгоритма функции общей работы с матричной клавиатурой представлена на рисунке 12:

Рисунок 12 – Схема алгоритма функции работы с матричной клавиатурой

4.2.1 Алгоритм инициализации с матричной клавиатуры

Функция keypad_init() обозначен для инициализации портов в МК.

Схема алгоритма функции инициализации матричной клавиатурой представлена на рисунке 13:

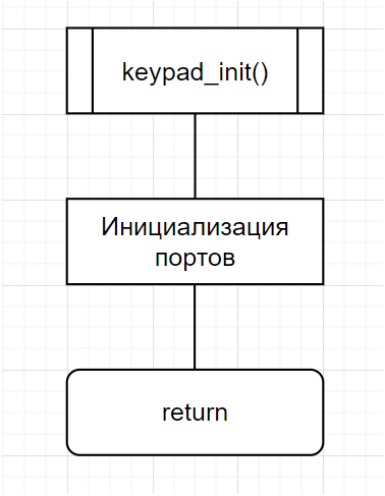

Рисунок 13 – Схема алгоритма функции работы с матричной клавиатурой

4.2.2 Алгоритм чтения нажатых данных с матричной клавиатуры

Функция keypad_read() обозначен для чтеных нажатых данных. Итерируя строки и столбцы, если нажатый ключ соответствует какому-либо элементу в матрице, он возвращает соответствующее значение.

Схема алгоритма функции инициализации матричной клавиатурой представлена на рисунке 14:

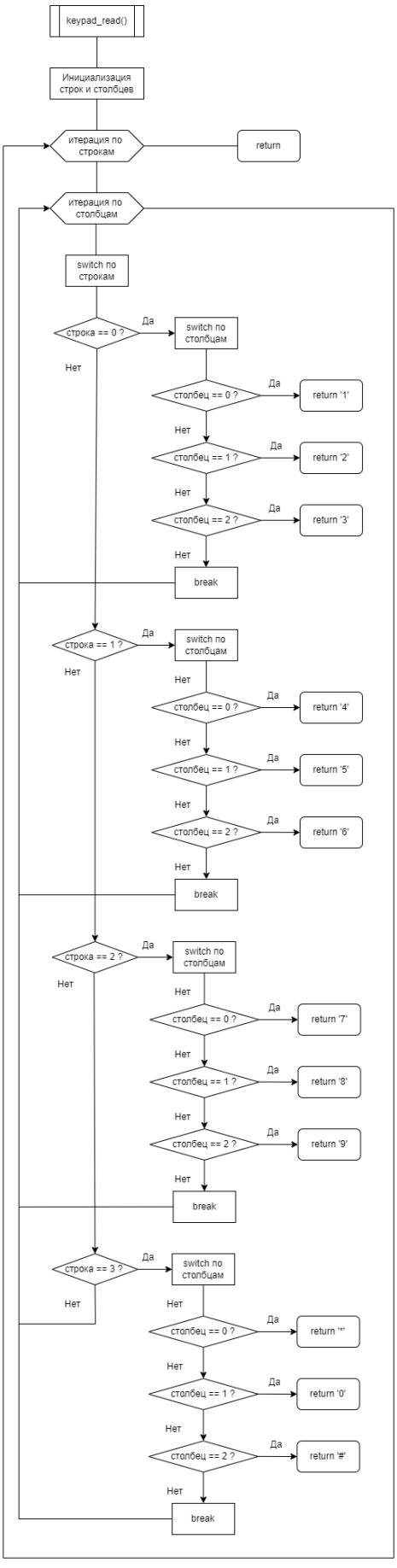

Рисунок 14 – Схема алгоритма функции чтения с матричной клавиатурой

4.2.3 Алгоритм преобразования номеров к буквам с матричной клавиатуры

Функция convert_to_letters() обозначен для преобразования номеров. Функция принимает массив нажатых клавиш и его длину в качестве параметра, проверяя, повторяются ли нажатые клавиши, если это так, итерируя его по его длине, превращая его в соответствующие буквы и возвращая его.

Схема алгоритма функции инициализации матричной клавиатурой представлена на рисунке 15:

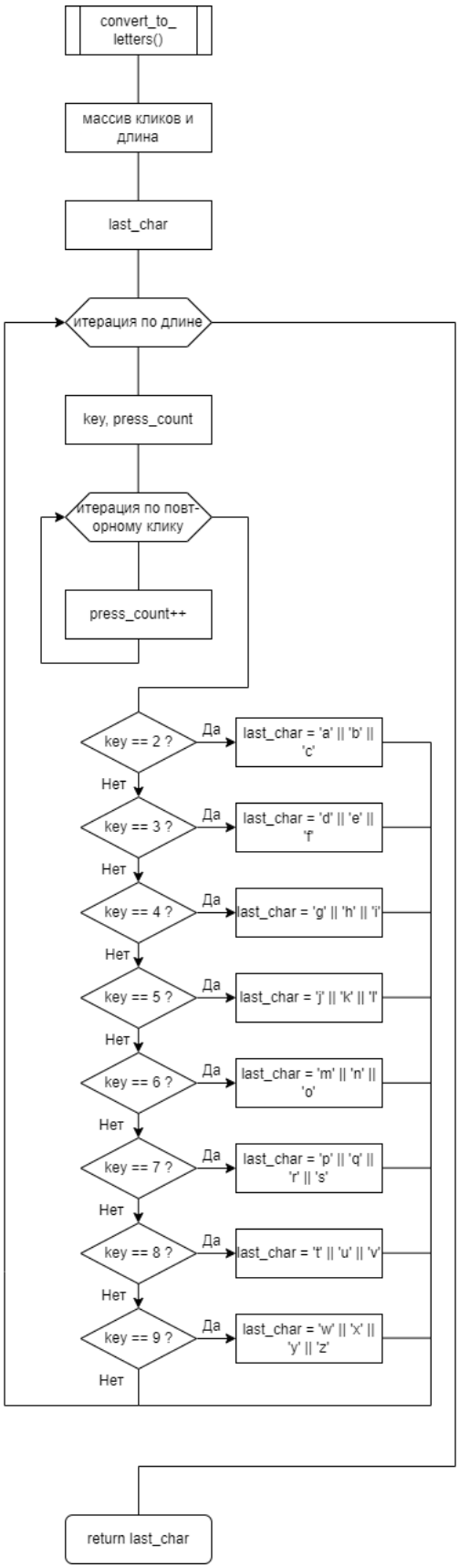

Рисунок 15 – Схема алгоритма функции преобразования номеров к буквам с матричной клавиатурой

4.3 Алгоритм работы EEPROM

Для реализации работы с EEPROM задействовано 4 функции: write_truck_to_eeprom(), write_all_trucks_to_eeprom(), read_truck_from_eeprom(), read_all_trucks_from_eeprom ().

4.3.1 Алгоритм записи в EEPROM

Функция write_truck_to_eeprom() назначен для записи данных в EEPROM, используя встроенную функцию eeprom_write_byte(). Получая адрес памяти и данные об грузовых транспортах в качестве параметров, данные (гос номер, масса без груза и масса с грузом) записываются.

Схема алгоритма записи в EEPROM представлена на рисунке 16:

Рисунок 16 – Схема алгоритма записи в EEPROM

4.3.2 Алгоритм записи данных всех грузовых транспортов в EEPROM

Функция write_all_trucks_to_eeprom() назначен для записи данных всех грузовых транспортов (в нашем случае 5 транспортов) в EEPROM, используя предыдущую функцию write_truck_to_eeprom().

Схема алгоритма записи транспортов в EEPROM представлена на рисунке 17:

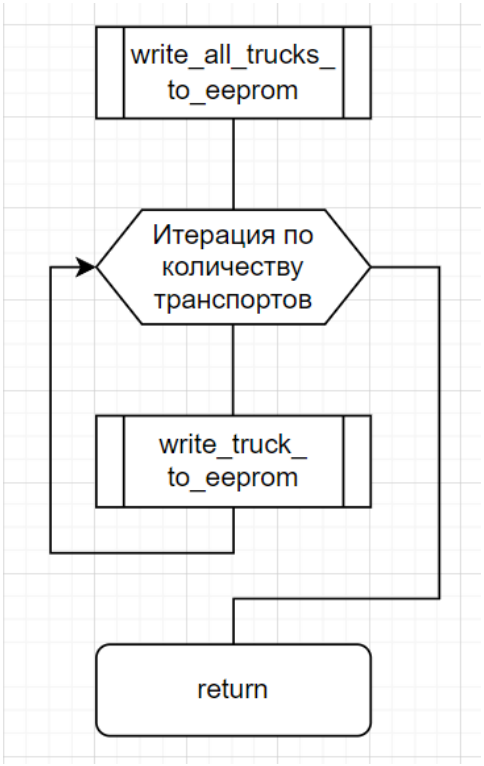

Рисунок 17 – Схема алгоритма записи всех транспортов в EEPROM

4.3.3 Алгоритм чтения из EEPROM

Функция read_truck_from_eeprom() назначен для чтения данных в EEPROM, используя встроенную функцию eeprom_read_byte(). Получая адрес памяти и данные об грузовых транспортах в качестве параметров, данные (гос номер, масса без груза и масса с грузом) считываются.

Схема алгоритма чтения из EEPROM представлена на рисунке 18:

Рисунок 18 – Схема алгоритма чтения из EEPROM

4.3.4 Алгоритм чтения данных всех грузовых транспортов в EEPROM

Функция read_all_trucks_from_eeprom() назначен для чтения данных всех грузовых транспортов (в нашем случае 5 транспортов) из EEPROM, используя предыдущую функцию read_truck_from_eeprom().

Схема алгоритма чтения транспортов из EEPROM представлена на рисунке 19:

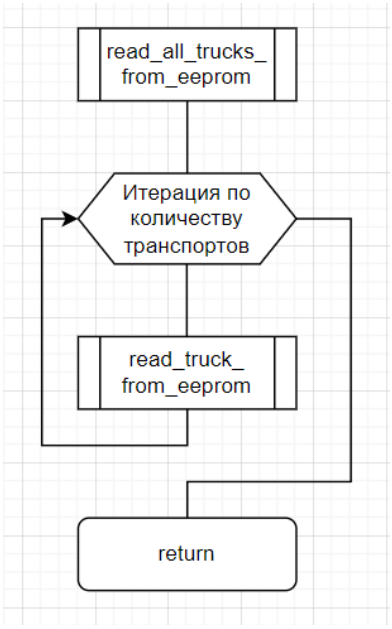

Рисунок 19 – Схема алгоритма чтения всех транспортов из EEPROM

4.4 Алгоритмы работы с дисплеем

Для реализации алгоритмов работы с дисплеем было создано 4 функции: command(), lcd_init(), lcd_data(), lcd_print(). Рассмотрим их подробнее.

4.4.1 Алгоритм отправки команд на дисплей

Функция lcd_command отправляет команду на устройство, управляемое с помощью портов микроконтроллера. Процесс передачи команды включает отправку старшего и младшего полубайтов с использованием сигналов управления.

Схема алгоритма отправки команд на дисплей представлена на рисунке 20:

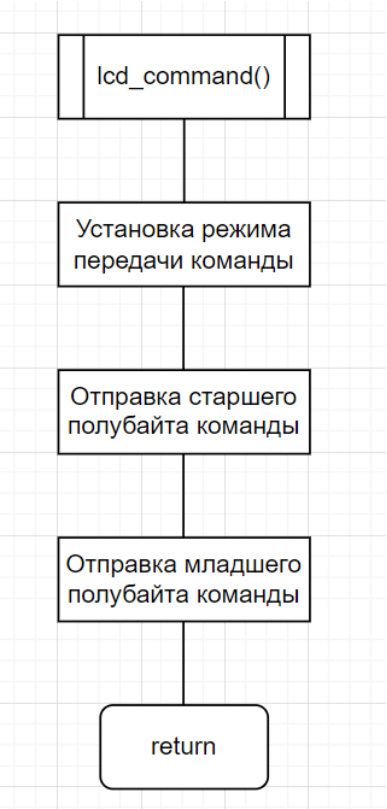

Рисунок 21 – Схема алгоритма отправки команд на дисплей

4.4.2 Алгоритм отправки данных на дисплей

Функция lcd_data отправляет данные (символы) на устройство, управляемое с помощью портов микроконтроллера. Процесс передачи включает отправку старшего и младшего полубайтов символа с использованием сигналов управления. Различность функции данных от функции команд является установкой режима. В режим данных RS=1.

Схема алгоритма отправки данных на дисплей представлена на рисунке 22:

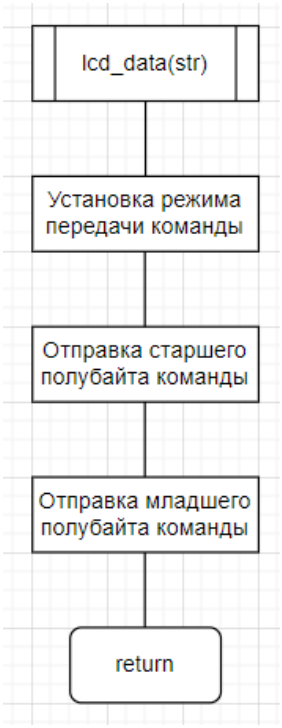

Рисунок 22 – Схема алгоритма отправки данных на дисплей

4.4.3 Алгоритм инициализации дисплея

Функция lcd_init выполняет начальную инициализацию LCD-дисплея для его корректной работы. Она настраивает микроконтроллер и отправляет команды дисплею, чтобы включить его и настроить рабочий режим.

Схема алгоритма инициализации дисплея представлена на рисунке 23:

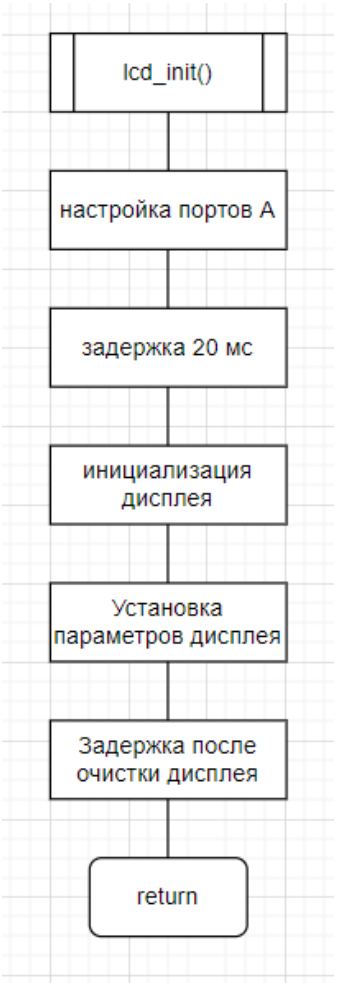

Рисунок 23 – Схема алгоритма инициализации дисплея

4.4.4 Алгоритм вывода строки на дисплей

Функция lcd_print выполняет вывод строки символов на дисплей. Строка передается в функцию в виде указателя на массив символов.

Схема алгоритма инициализации дисплея представлена на рисунке 24:

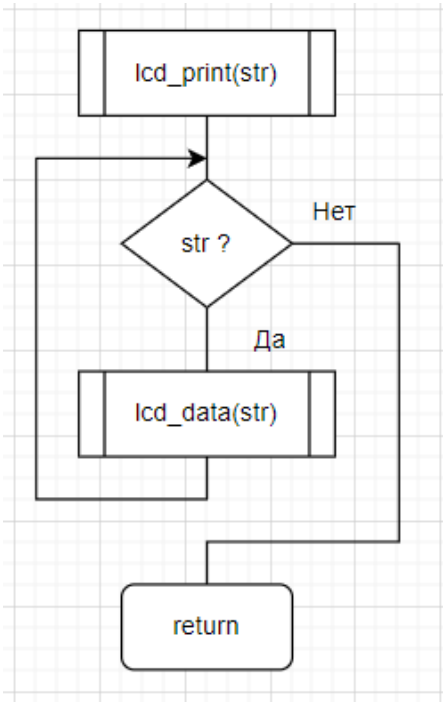

Рисунок 24 – Схема алгоритма вывода строки на дисплей

4.5 Алгоритмы работы с зуммером

Для реализации алгоритмов работы с зуммером было создано 2 функции: setup(), toggle_buzzer(). Рассмотрим их подробнее.

1) setup() – Инициализация выхода для управления буззером.
   - Назначение: Настройка пина микроконтроллера как выхода.
   - Действия: Установка PD7 как выходной пин.
2) toggle_buzzer() – Переключение состояния буззера.
   - Назначение: Включение/выключение буззера путем инверсии состояния пина.
   - Действия: Используется операция XOR для инверсии текущего состояния пина. Если пин был высоким (1), он станет низким (0) и наоборот.

5 Технологическая часть

5.1 Описание систем разработки

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

1) Proteus 8 Professional – многофункциональная программа для автоматизированного проектирования электронных схем.
2) AVR Studio – оптимизирующий компилятор программ, написанных на языках С для AVR.

5.2 Отладка и тестирование устройства

Схема устройства разработана, отлажена и протестирована в программе Proteus.

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;
- avr/eeprom.h – это библиотека, предназначенная для работы с энергонезависимой памятью EEPROM в МК AVR. Она предоставляет функции для чтения и записи данных в EEPROM, что позволяет сохранять информацию даже после отключения питания. EEPROM используется для хранения конфигурационных данных,

параметров и другой информации, которая должна сохраняться между перезагрузками устройства;

- stdlib.h – это стандартная библиотека C общего назначения, включающая в себя функции, занимающиеся распределением памяти, управлением процессами, преобразованием и др;
- stdbool.h – это библиотека, которая позволяет работать с данными типа bool;
- string.h – это стандартная библиотека языка C, которая предоставляет функции для работы со строками и массивами символов. Она включает в себя различные функции для манипуляции строками.

Схема устройства представлена на рисунке 25:

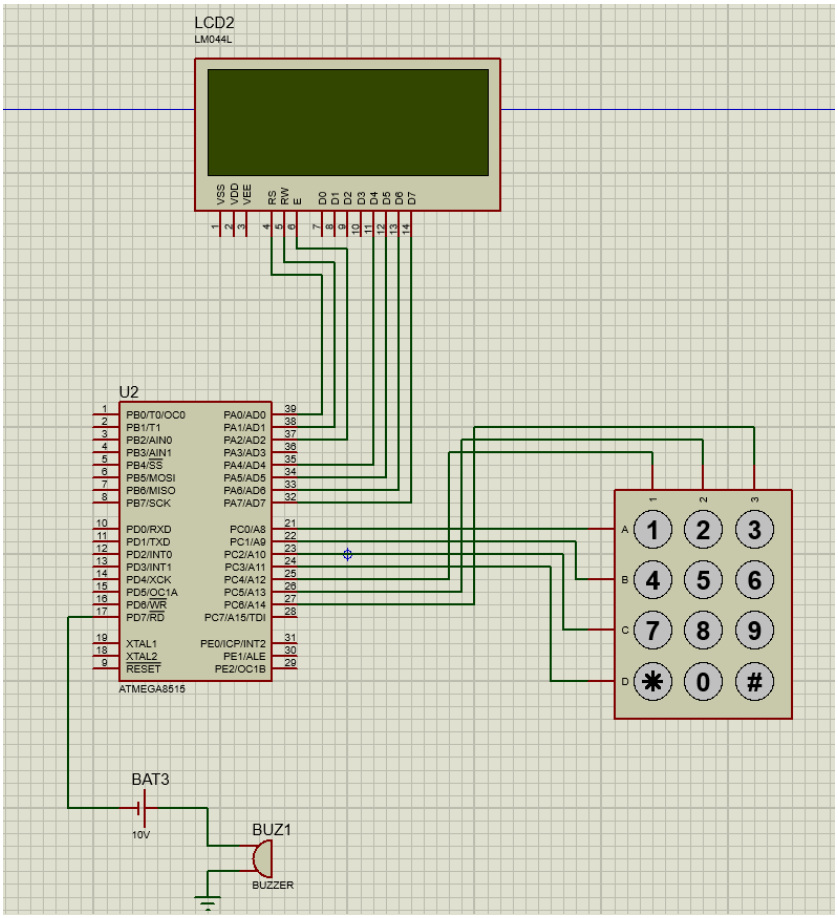

Рисунок 25 – схема устройства в Proteus

5.3 Симуляция работы устройства

Каждый ввод данных подтвержден с нажатием кнопки “*”. После ввода данных дисплей задерживается 2 секунды для отображения данных.

На рисунке 26 показан ввод гос номера:

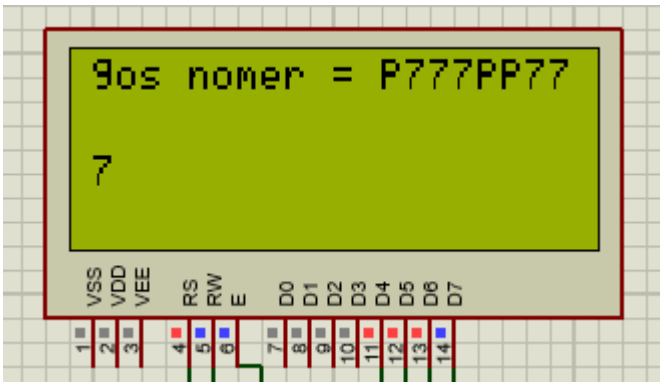

Рисунок 26 – ввод гос номера

На рисунке 27 показан ввод массы без груза:

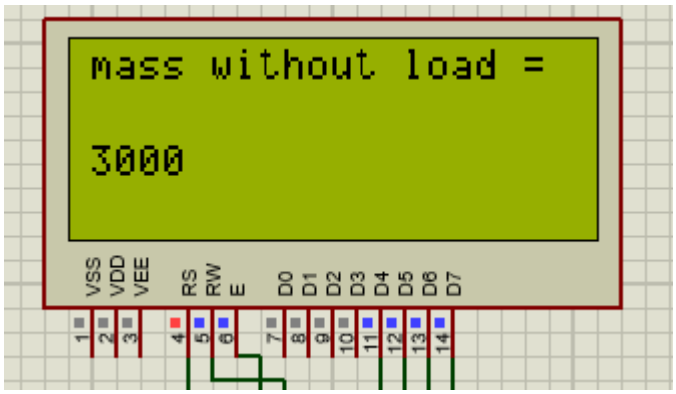

Рисунок 27 – ввод массы без груза

На рисунке 28 показан ввод массы с грузом:

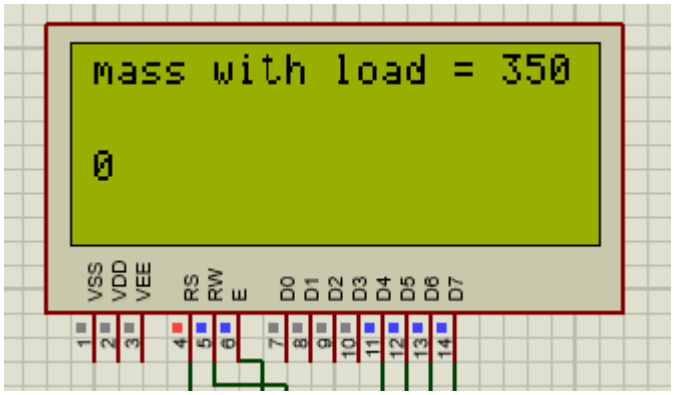

Рисунок 28 – ввод массы с грузом

На рисунке 29 показано сообщение совпадания пользовательского данных с данными из базы данных:

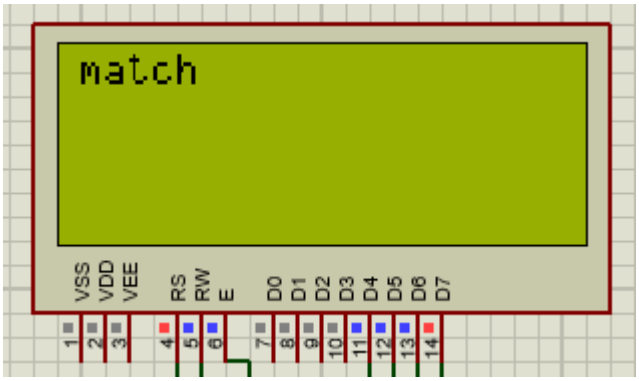

Рисунок 29 – вывод информации о совпадании данных

На рисунке 30 показано сообщение несовпадания пользовательского данных с данными из базы данных:

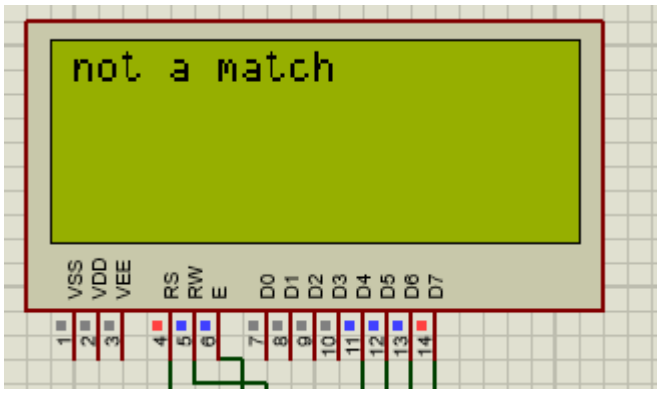

Рисунок 30 – вывод информации о несовпадании данных

5.4 Способы программирования памяти 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:

- 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. Визуальное представление показано на рисунке 31:

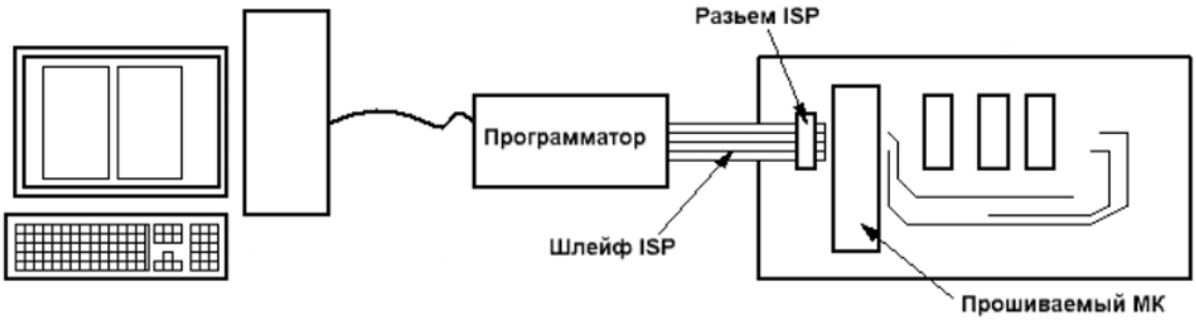

Рисунок 31 – Программирование МК с помощью SPI

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. Для программирования и проверки Atmega8515 в режиме последовательного программирования SPI рекомендуется следующая последовательность:

1) Необходимо подать питание между VCC и GND, а также выполнить сброс микроконтроллера путем подачи низкого напряжения на инверсный вход RESET.

2) После паузы не менее 20 мс необходимо включить последовательное программирование SPI, отправив команду Programming Enable serial на pin MOSI.

3) Инструкции по последовательному программированию SPI не будут работать, если связь не синхронизирована. Когда они синхронизированы второй байт ($53), будет отражен при выдаче третьего байта команды включения программирования.

4) Flash программируется по одной странице за раз. Страница памяти загружается по одному байту за раз путем ввода 6 LSB адреса и данных вместе с инструкцией загрузки страницы памяти программы. Чтобы обеспечить правильную загрузку страницы, младший байт данных должен быть загружен до того, как для данного адреса будет применен старший байт данных.

5) EEPROM программируется по одному байту за раз путем ввода адреса и данных вместе с соответствующей инструкцией записи. Ячейка памяти EEPROM сначала автоматически стирается перед записью новых данных.

6) Любая ячейка памяти может быть проверена с помощью инструкции чтения, которая возвращает содержимое по выбранному адресу при последовательном выводе MISO.

7) В конце сеанса программирования сброс может быть установлен на высокий уровень, чтобы начать нормальную работу.

После выполнения указанных шагов программатор может быть отключен и устройство готово к работе.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был создан проект – система контроллера КПП грузового транспорта. Устройство реализовано на базе микроконтроллера AtMega8515, оснащено матричной клавиатурой для ввода данных, LCD-дисплеем для отображения информации, интегрировано с базой данных EEPROM и с зуммером для указания неправильного ввода. Устройство разработано в соответствии с ТЗ.

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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 2.743-91 ЕСКД ОБОЗНАЧЕНИЯ БУКВЕННО-ЦИФРОВЫЕ В ЭЛЕКТРИЧЕСКИХ СХЕМАХ [Электронный ресурс]. – Режим доступа: https://docs.cntd.ru/document/1200001985 (дата обращения: 17.11.2024)

2. ГОСТ 2.721-74 ЕСКД ОБОЗНАЧЕНИЯ УСЛОВНЫЕ ГРАФИЧЕСКИЕ В СХЕМАХ [Электронный ресурс]. – Режим доступа: https://docs.cntd.ru/document/1200007058 (дата обращения: 17.11.2024)

3. Методические указания по выполнению курсовой работы по дисциплине “Микропроцессорные системы” [Электронный ресурс]. – Режим доступа: https://e-learning.bmstu.ru/iu6/mod/resource/view.php?id=9847 (дата обращения: 17.11.2024)

4. Atmega8515 Datasheet [Электронный ресурс]. – Режим доступа: https://www.alldatasheet.com/datasheet-pdf/pdf/166872/ATMEL/ATMEGA8515.html (дата обращения: 17.11.2024)

5. LM044L Datasheet [Электронный ресурс]. – Режим доступа: https://www.datasheet4u.com/datasheet-pdf/Hitachi/LM044L/pdf.php?id=540804 (дата обращения: 17.11.2024)

6. Matrix Keypad Datasheet [Электронный ресурс]. – Режим доступа: https://www.alldatasheet.com/datasheet-pdf/pdf/73047/MAXIM/matrix-keypad.html (дата обращения: 17.11.2024)

Приложение А. Исходный текст программы со стороны МК

Листов 9

Объем исполняемого кода – 522 строки.

Main.c

```
#include <avr/io.h>
#include <avr/eeprom.h>
#include <util/delay.h>
#include <stdbool.h>

#include "lcd.h"
#include "keypad.h"
#include "eeprom.h"

#define BUZZER_PIN PD7

void setup() {
  // Set PD7 as an output
  DDRD |= (1 << BUZZER_PIN);
}

void toggle_buzzer() {
  PORTD ^= (1 << BUZZER_PIN);
}

char convert_to_letters(char* key_store, int count) {
  char last_char = '\0'; // Variable to hold the last generated
character

  int i = 0;
  for (i = 0; i < count; i++) {
    char key = key_store[i];
    int press_count = 1;

    // Count how many times the same key is pressed
    while (i + 1 < count && key_store[i + 1] == key) {
      press_count++;
      i++;
    }

    // Map the key and press count to the corresponding letter
    if (key == '2') {
      last_char = 'A' + (press_count - 1) % 3; // a, b, c
    } else if (key == '3') {
      last_char = 'D' + (press_count - 1) % 3; // d, e, f
    } else if (key == '4') {
      last_char = 'G' + (press_count - 1) % 3; // g, h, i
    } else if (key == '5') {
      last_char = 'J' + (press_count - 1) % 3; // j, k, l
    } else if (key == '6') {
      last_char = 'M' + (press_count - 1) % 3; // m, n, o
    } else if (key == '7') {
```

```
      last_char = 'P' + (press_count - 1) % 4; // p, q, r, s
    } else if (key == '8') {
      last_char = 'T' + (press_count - 1) % 3; // t, u, v
    } else if (key == '9') {
      last_char = 'W' + (press_count - 1) % 4; // w, x, y, z
    }
  }

  return last_char; // Return the last generated character
}

// start
int main(void) {
  setup();

  write_all_trucks_to_eeprom();
  read_all_trucks_from_eeprom();

  lcd_init(); // Initialize the LCD
  lcd_print("Input gos nomer: "); // Print an initial message
  _delay_ms(2000);

  lcd_command(0x01); // Clear the LCD
  _delay_ms(2);

  keypad_init();
  char gos_nomer[10] = {0}; // Array to hold the car number
  int len_gos_nomer = 0;
  int i = 0;

  char key;
  char last_key = '\0'; // Variable to track the last key pressed
  char key_store[10] = {0}; // Temporary key store
  int len_key_store = 0;


  // gos nomer loop
  while(1) {
    key = keypad_read(); // Read the key press

    // Check if a key is pressed and it's different from the last key
    if (key && key != last_key) {

      // Insert key into key_store
      if (key != '*' && key != '#') {
        if (len_key_store < sizeof(key_store) - 1) {
          key_store[len_key_store] = key; // Store the key
          len_key_store++;
        }
      }

      // Submit to gos_nomer
      else {
        // Check if we can add to gos_nomer
        if (len_gos_nomer < sizeof(gos_nomer) - 1) {

          if (len_gos_nomer == 0 || len_gos_nomer == 4 ||
len_gos_nomer == 5) {
```

```
            char that_letter = convert_to_letters(key_store,
len_key_store);
            gos_nomer[len_gos_nomer] = that_letter;
          }

          else {
            char that_number = key_store[0]; // Get the first
character from key_store
            gos_nomer[len_gos_nomer] = that_number; // Add to
gos_nomer
          }

          len_gos_nomer++; // Increment the length
          gos_nomer[len_gos_nomer] = '\0'; // Null-terminate the
string
        }

        // Clear key_store
        len_key_store = 0;

        // Update the LCD
        lcd_command(0x01);
        _delay_ms(2);

     char buffer[3]; // Buffer to hold the string representation (2
digits + null terminator)
        sprintf(buffer, "%d", len_gos_nomer);

        lcd_print(gos_nomer);

        if (buffer[0] == '9') {
          break;
        }
      }

      last_key = key; // Update last_key to the current key
    } else if (!key) {
      // Reset last_key when no key is pressed
      last_key = '\0';
    }
  }

  ////////////////////// end of gos_nomer ////////////////////

  _delay_ms(2000);
  lcd_command(0x01);
  _delay_ms(2);

  lcd_print("Input mass without load: ");
  _delay_ms(2000);
  lcd_command(0x01);
  _delay_ms(2);

  char mass_without_load[6] = {0};
  len_key_store = 0;
  strcpy(key_store, "");

  // mass without load loop
```

```
  while (1) {
    key = keypad_read();

    if (key && key != last_key) {
      if (key != '*' && key != '#') {
        if (len_key_store < sizeof(key_store) - 1) {
          key_store[len_key_store] = key;
          len_key_store++;

          lcd_command(0x01);
          _delay_ms(2);
          lcd_print(key_store);
        }
      }
      else {
        memcpy(mass_without_load, key_store, len_key_store);
        mass_without_load[len_key_store] = '\0';

        lcd_command(0x01);
        _delay_ms(500);
        char print_str[30];
        sprintf(print_str, "mass without load = %s",
mass_without_load);
        lcd_print(print_str);
        break;
      }

    last_key = key;
    } else if (!key) {
      last_key = '\0';
    }
  }

  ////////////////////// end of mass without load ////////////////////

  _delay_ms(2000);
  lcd_command(0x01);
  _delay_ms(2);

  lcd_print("Input mass with load: ");
  _delay_ms(2000);
  lcd_command(0x01);
  _delay_ms(2);

  char mass_with_load[6] = {0};
  len_key_store = 0;

  for (i = 0; i < sizeof(key_store); i++) {
    key_store[i] = '\0'; // Set each element to null character
  }

  // mass with load loop
  while (1) {
    key = keypad_read();

    if (key && key != last_key) {
      if (key != '*' && key != '#') {
        if (len_key_store < sizeof(key_store) - 1) {
```

```
          key_store[len_key_store] = key;
          len_key_store++;

          lcd_command(0x01);
          _delay_ms(2);
          lcd_print(key_store);
        }
      }
      else {
        memcpy(mass_with_load, key_store, len_key_store);
        mass_with_load[len_key_store] = '\0';

        lcd_command(0x01);
        _delay_ms(500);
        char print_str[30];
        sprintf(print_str, "mass with load = %s", mass_with_load);
        lcd_print(print_str);
        break;
      }

    last_key = key;
    } else if (!key) {
      last_key = '\0';
    }
  }

  ////////////////////// end of mass with load ////////////////////

  // if data matches or not, print the corresponding message
  bool comparing = false;
  lcd_command(0x01);
  _delay_ms(2);

  for (i = 0; i < 5; i++) {
    if ((strcmp(gos_nomer, trucks[i].gos_nomer) == 0) &&
        (strcmp(mass_without_load, trucks[i].mass_without_load) == 0)
&&
        (strcmp(mass_with_load, trucks[i].mass_with_load) == 0)) {
          comparing = true;
          break;
    }
  }

  if (comparing) {
    lcd_command(0x01);
    _delay_ms(250);
    lcd_print("match");
  }
  else {
    lcd_command(0x01);
    _delay_ms(250);
    lcd_print("not a match");
    toggle_buzzer();
    _delay_ms(500);
  }

  return;
}
```

Lcd.c

```
// lcd.c
#include "lcd.h"
#include <util/delay.h>

void lcd_command(unsigned char cmd) {
    PORTA &= ~(1 << LCD_RS); // RS = 0 for command
    PORTA &= ~(1 << LCD_RW); // RW = 0 for write
    PORTA |= (1 << LCD_E);   // Enable high

    // Send higher nibble
    PORTA = (PORTA & 0x0F) | (cmd & 0xF0); // Mask lower nibble
    PORTA &= ~(1 << LCD_E); // Enable low
    _delay_us(100);         // Delay for command execution

    PORTA |= (1 << LCD_E);  // Enable high
    // Send lower nibble
    PORTA = (PORTA & 0x0F) | ((cmd << 4) & 0xF0); // Mask lower nibble
    PORTA &= ~(1 << LCD_E); // Enable low
    _delay_us(100);         // Delay for command execution
}

void lcd_data(unsigned char data) {
    PORTA |= (1 << LCD_RS);  // RS = 1 for data
    PORTA &= ~(1 << LCD_RW); // RW = 0 for write
    PORTA |= (1 << LCD_E);   // Enable high

    // Send higher nibble
    PORTA = (PORTA & 0x0F) | (data & 0xF0); // Mask lower nibble
    PORTA &= ~(1 << LCD_E); // Enable low
    _delay_us(100);         // Delay for data execution

    PORTA |= (1 << LCD_E);  // Enable high
    // Send lower nibble
    PORTA = (PORTA & 0x0F) | ((data << 4) & 0xF0); // Mask lower
nibble
    PORTA &= ~(1 << LCD_E); // Enable low
    _delay_us(100);         // Delay for data execution
}

void lcd_init() {
    DDRA = 0xFF; // Set PORTA as output
    _delay_ms(20); // Wait for LCD to power up

    // Initialize LCD in 4-bit mode
    lcd_command(0x33); // Initialize
    lcd_command(0x32); // Set to 4-bit mode
    lcd_command(0x28); // 2 lines, 5x7 matrix
    lcd_command(0x0C); // Display ON, Cursor OFF
    lcd_command(0x06); // Increment cursor
    lcd_command(0x01); // Clear display
    _delay_ms(2);      // Wait for clear command to finish
}

void lcd_print(const char *str) {
```

```
    while (*str) {
        lcd_data(*str++);
    }
}
```

Lcd.h

```
// lcd.h
#ifndef LCD_H
#define LCD_H

#include <avr/io.h>

#define LCD_RS PA0
#define LCD_RW PA1
#define LCD_E  PA2
#define LCD_D4 PA4
#define LCD_D5 PA5
#define LCD_D6 PA6
#define LCD_D7 PA7

void lcd_init(void);
void lcd_command(unsigned char cmd);
void lcd_data(unsigned char data);
void lcd_print(const char *str);

#endif // LCD_H
```

Keypad.c

```
#include <avr/io.h>
#include <util/delay.h>
#include "keypad.h" // Include the header file for function prototypes

#define KEYPAD_PORT PORTC
#define KEYPAD_DDR DDRC
#define KEYPAD_PIN PINC

// Function to initialize the keypad
void keypad_init() {
    KEYPAD_DDR = 0x0F; // Set PC0-PC3 as output (columns), PC4-PC6 as
input (rows)
    KEYPAD_PORT = 0xF0; // Enable pull-up resistors on PC4-PC6
}

// Function to read the keypad
char keypad_read() {
    uint8_t row, col; // Declare loop variables outside the loop

    for (row = 0; row < 4; row++) {
        KEYPAD_PORT = ~(1 << row); // Set one column low at a time

        // Delay to stabilize the signal
        _delay_ms(10);

        // Check each row
        for (col = 0; col < 3; col++) {
```

```
            if (!(KEYPAD_PIN & (1 << (col + 4)))) { // Check if row is
low
                // Debounce
                _delay_ms(10);
                if (!(KEYPAD_PIN & (1 << (col + 4)))) {
                    // Return the key pressed based on the row and
column
                    // Map rows and columns to actual numbers
                    switch (row) {
                        case 0: // First row
                            switch (col) {
                                case 0: return '1'; // Row 0, Col 0
                                case 1: return '2'; // Row 0, Col 1
                                case 2: return '3'; // Row 0, Col 2
                            }
                            break;
                        case 1: // Second row
                            switch (col) {
                                case 0: return '4'; // Row 1, Col 0
                                case 1: return '5'; // Row 1, Col 1
                                case 2: return '6'; // Row 1, Col 2
                            }
                            break;
                        case 2: // Third row
                            switch (col) {
                                case 0: return '7'; // Row 2, Col 0
                                case 1: return '8'; // Row 2, Col 1
                                case 2: return '9'; // Row 2, Col 2
                            }
                            break;
                        case 3: // Third row
                            switch (col) {
                                case 0: return '*'; // Row 2, Col 0
                                case 1: return '0'; // Row 2, Col 1
                                case 2: return '#'; // Row 2, Col 2
                            }
                            break;
                    }
                }
            }
        }
    }
    return 0; // No key pressed
}
```

Keypad.h

```
#ifndef KEYPAD_H
#define KEYPAD_H

#include <avr/io.h>

void keypad_init();
char keypad_read();

#endif // KEYPAD_H
```

Eeprom.c

```
#include "eeprom.h"

// Define the Truck array (this should match the `extern` declaration
in the header)
struct Truck trucks[5] = {
    {"A123BC077", "1000", "1500"},
    {"B456DE088", "2000", "2500"},
    {"P777PP777", "3000", "3500"},
    {"C789FG099", "4000", "4500"},
    {"D012GH000", "5000", "5500"}
};

struct Truck read_trucks[5] = {};

// Function to write a truck to EEPROM
void write_truck_to_eeprom(uint16_t addr, struct Truck* truck) {
    int i = 0;
    for (i = 0; i < 9; i++) {
        eeprom_write_byte((uint8_t*) (addr + i), truck->gos_nomer[i]);
    }

    for (i = 0; i < 4; i++) {
        eeprom_write_byte((uint8_t*) (addr + 9 + i), truck-
>mass_without_load[i]);
    }

    for (i = 0; i < 4; i++) {
        eeprom_write_byte((uint8_t*) (addr + 13 + i), truck-
>mass_with_load[i]);
    }
}

// Function to write all trucks to EEPROM
void write_all_trucks_to_eeprom(void) {
    int i = 0;
    for (i = 0; i < 5; i++) {
        write_truck_to_eeprom(i * (17), &trucks[i]);
    }
}

// Function to read a truck from EEPROM
void read_truck_from_eeprom(uint16_t addr, struct Truck* truck) {
    int i = 0;
    for (i = 0; i < 9; i++) {
         char data = eeprom_read_byte((uint8_t*) (i + addr));
         truck->gos_nomer[i] = data;
    }
    truck->gos_nomer[10] = '\0';

    for (i = 0; i < 4; i++) {
        char data = eeprom_read_byte((uint8_t*) (i + addr + 9));
        truck->mass_without_load[i] = data;
    }
    truck->mass_without_load[5] = '\0';
```

```
    for (i = 0; i < 4; i++) {
        char data = eeprom_read_byte((uint8_t*) (i + addr + 13));
        truck->mass_with_load[i] = data;
    }
    truck->mass_with_load[5] = '\0';
}

// Function to read all trucks from EEPROM
void read_all_trucks_from_eeprom(void) {
    int i = 0;
    for (i = 0; i < 5; i++) {
        read_truck_from_eeprom(i * (17), &read_trucks[i]);
    }
}
```

Eeprom.h

```
#ifndef EEPROM_H
#define EEPROM_H

#include <avr/io.h>
#include <avr/eeprom.h>
#include <string.h>

// Define the Truck structure
struct Truck {
    char gos_nomer[10];         // License plate number (9 characters)
    char mass_without_load[5]; // Mass without load (as string)
    char mass_with_load[5];    // Mass with load (as string)
};

// Declare an array of 10 trucks
extern struct Truck trucks[5];

extern struct Truck read_trucks[5];

// Function to write a truck to EEPROM
void write_truck_to_eeprom(uint16_t addr, struct Truck* truck);

// Function to write all trucks to EEPROM
void write_all_trucks_to_eeprom(void);

// Function to read a truck from EEPROM
void read_truck_from_eeprom(uint16_t addr, struct Truck* truck);

// Function to read all trucks from EEPROM
void read_all_trucks_from_eeprom(void);

#endif // EEPROM_H
```

Приложение Б

Графическая часть

На 2 листах

Электрическая схема функциональная

Электрическая схема принципиальная

Приложение В

На 2 листах

Перечень элементов