



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:

***Балансировщик нагрузки
распределенной сетит***

Студент

ИУ6-73Б

(Группа)

(Подпись, дата)

В.С. Нейбауэр

(И.О. Фамилия)

Руководитель

(Подпись, дата)

И.Б. Трамов

(И.О. Фамилия)

2024 г.

ЗАДАНИЕ

РЕФЕРАТ

РПЗ 59 страниц, 27 рисунков, 7 таблиц, 10 источников, 4 приложения.

МИКРОКОНТРОЛЛЕР, СИСТЕМА, УСТРОЙСТВО,
БАЛАНСИРОВЩИК НАГРУЗКИ, РАСПРЕДЕЛЕННАЯ СИСТЕМА, AVR,
ROUND ROBIN

Объектом разработки является балансировщик нагрузки распределенной системы.

Цель работы – разработка и создание МК-системы на базе семейства AVR, балансирующей нагрузку в распределенной системы.

Поставленная цель достигается посредством использования Proteus 8 и sPlan 8.0.

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 программы МК, а также написание сопутствующей документации.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	7
1.1 Конструкторская часть	8
1.1 Анализ требований и принцип работы системы	8
1.1.1 Изучение технического задания	8
1.1.2 Выбор способа связи микроконтроллеров в распределённой системе и её архитектура	8
1.1.3 Выбор параметров оптимизации и алгоритма оптимизации	10
1.1.4 Внешние компоненты	12
1.1.5 Выбор элементной базы	13
1.1.6 Проектирование структурной схемы	15
1.2 Проектирование функциональной схемы.....	16
1.2.1 Микроконтроллер ATmega16	16
1.2.1.1 Используемые элементы	23
1.2.1.2 Распределение портов.....	24
1.2.1.3 Организация памяти	25
1.2.2 Внешняя периферия.....	26
1.2.2.1 Интерфейс RS-232 и модуль MAX232	26
1.2.2.2 Bluetooth-модуль HC-05	29
1.2.2.3 Интерфейс I2C.....	30
1.2.2.4 LCD дисплей.....	31
1.2.3 Подключение двух устройств, работающих по UART	32
1.2.4 Настройка передачи данных по каналу UART	34
1.2.5 Генератор тактовых импульсов	36
1.2.6 Сброс микроконтроллера	37
1.2.8 Построение функциональной схемы.....	38
1.3 Проектирование принципиальной схемы.....	39
1.3.1 Разъем программатора	39
1.3.2 Подключение цепи питания	39
1.3.3 Расчет сопротивления резисторов	40
1.3.4 Расчет потребляемой мощности	41
1.3.5 Построение принципиальной схемы.....	42
1.4 Алгоритмы работы системы	43
2 Технологическая часть	47

2.1 Отладка и тестирование программы.....	47
2.2 Симуляция работы системы	49
2.3 Способы программирования МК	56
ЗАКЛЮЧЕНИЕ	58
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	59
Приложение А	60
Приложение Б	69
Приложение В.....	71
Приложение Г	73

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК – микроконтроллер.

ТЗ – техническое задание.

РС – распределенная система.

RR – Round-robin, циклический алгоритм распределения нагрузки в распределенной сети.

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

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

I2C – Inter-Integrated Circuit – интерфейс для связи МК с другими внешними устройствами.

ПК – персональный компьютер.

ПЭВМ – персональная электронвычислительная машина.

EEPROM – Electrically Erasable Programmable Read-Only Memory, электрически стираемое перепрограммируемое постоянное запоминающее устройство.

Flash – разновидность программируемой памяти.

SRAM – Static Random Access Memory, статическая память с произвольным доступом.

LCD / ЖКД – Liquid Crystal Display / жидкокристаллический дисплей

ВВЕДЕНИЕ

В данной работе производится разработка устройства, решающего задачу балансировки нагрузки в распределённых системах. В процессе выполнения работы проведён анализ технического задания, создана концепция устройства, разработаны электрические схемы, построен алгоритм и управляющая программа для микроконтроллера, выполнено интерактивное моделирование устройства.

Система состоит из микроконтроллера AVR, который осуществляет обработку данных, поступающих через UART-интерфейс либо с помощью эмуляции COM-порта на компьютере, либо с помощью подключения по Bluetooth через телефон к HC-05 и графического ЖК-дисплея, предназначенного для визуализации результатов распределения нагрузки в распределённой системе. Ввод данных реализуется с использованием ПК и телефона, а обработка включает проверку корректности конфигурации, парсинг входных данных и выполнение балансировки нагрузки на основе заранее определённых алгоритмов. Вывод результатов осуществляется как на дисплей, так и обратно на ПК или виртуальный терминал посредством UART-интерфейса, а также на телефон через модуль HC-05 по Bluetooth.

Актуальность разрабатываемой системы для балансировки нагрузки в распределённых системах заключается в её применении для оптимизации использования ресурсов и повышения производительности системы, что критически важно при проектировании и эксплуатации масштабируемых цифровых устройств и систем управления. Использование алгоритмов балансировки нагрузки позволяет эффективно распределять рабочие нагрузки, делая систему более эффективной и надёжной. Созданное устройство обеспечивает удобный и наглядный способ демонстрации работы алгоритмов балансировки, что особенно ценно в инженерной практике.

1.1 Конструкторская часть

1.1 Анализ требований и принцип работы системы

1.1.1 Изучение технического задания

Исходя из требований, изложенных в техническом задании, необходимо разработать микроконтроллерную систему (МК-систему) на базе семейства AVR для функционирования в качестве балансирующей нагрузки в распределённой системе (РС). Основная задача системы — оптимизация работы сети, состоящей из более чем 100 микроконтроллеров, путём эффективного распределения нагрузки между ними. Помимо этого, система должна обеспечивать передачу всей необходимой информации на персональный компьютер (ПК) и мобильный телефон.

Распределённая система (РС) представляет собой совокупность взаимосвязанных вычислительных узлов (в данном случае микроконтроллеров), которые совместно выполняют задачи для достижения общих целей. В распределённых системах ресурсы и задачи распределены между несколькими узлами, что обеспечивает повышенную гибкость, отказоустойчивость и масштабируемость. Преимущества распределённых систем включают параллельную обработку данных, снижение времени отклика и улучшение общей производительности.

1.1.2 Выбор способа связи микроконтроллеров в распределённой системе и её архитектура

Для обеспечения надёжной и эффективной коммуникации между микроконтроллерами в распределённой системе был выбран интерфейс I2C. Этот выбор обусловлен следующими причинами:

- 1) двухпроводной интерфейс;
- 2) мастер-слейв архитектура;
- 3) поддержка большого числа устройств;
- 4) простота реализации.

I2C использует всего две линии для связи — SCL (Serial Clock) и SDA (Serial Data), что значительно упрощает схему подключения при большом количестве узлов. Мастер-Слейв архитектура позволяет одному главному контроллеру (мастеру) управлять несколькими подчинёнными устройствами (слейвами), что идеально подходит для балансировки нагрузки. Также, I2C способен поддерживать до 112 устройств на одной шине, что соответствует требованиям проекта по подключению 112 микроконтроллеров-слейвов. А главное - интерфейс I2C широко поддерживается в микроконтроллерах AVR и имеет хорошо задокументированную спецификацию, что облегчает разработку и отладку системы.

I2C представляет собой синхронный последовательный интерфейс, предназначенный для связи микроконтроллеров с периферийными устройствами. Разработанный компанией Philips (ныне NXP Semiconductors), этот интерфейс получил широкое распространение благодаря своей простоте и эффективности. Основными характеристиками I2C являются двунаправленная двухпроводная связь, состоящая из линии тактовых импульсов SCL (Serial Clock), генерируемых мастером для синхронизации передачи данных, и линии данных SDA (Serial Data), используемой для передачи информации между устройствами. I2C работает по модели "мастер-слейв", где мастер — это устройство, инициирующее передачу данных и генерирующее тактовые импульсы, обычно это главный микроконтроллер, а слейвы — устройства, реагирующие на команды мастера, такие как датчики, EEPROM, ADC и другие периферийные устройства. Каждое слейв-устройство имеет уникальный адрес (7 или 10 бит), что позволяет мастеру выбирать, с каким устройством взаимодействовать. Скорость передачи данных в I2C варьируется от стандартного режима до 100 кбит/с, быстрого режима до 400 кбит/с и высокоскоростного режима до 3.4 Мбит/с. Простота масштабирования является одной из ключевых особенностей I2C, позволяющей легко добавлять новые устройства к шине без значительных изменений в схеме. Таким образом, использование I2C обеспечивает

эффективную и надёжную связь между микроконтроллерами и периферийными устройствами с минимальным количеством проводов и высокой гибкостью в масштабировании системы.

В соответствии с техническим заданием, система должна поддерживать подключение 112 микроконтроллеров-слейвов. Такая конфигурация требует тщательно продуманной архитектуры связи и управления нагрузкой. Основные компоненты архитектуры системы:

- 1) главный микроконтроллер (мастер);
- 2) 112 микроконтроллеров-слейвов;
- 3) I2C-шина;
- 4) питание;
- 5) интерфейсы для связи с ПК и телефоном.

Мастер управляет всей системой, обрабатывает входящие задачи и распределяет их между 112 мк-слейвами. Они выполняют полученные задачи, сообщают о своём статусе и готовности к приёму новых задач. I2C-шина предоставляет связь между мастером и слейвами, позволяя эффективную передачу данных и управление устройствами. Питание обеспечивает стабильную работу всех микроконтроллеров и периферийных устройств. Интерфейсы для связи с ПК и телефоном предоставляют ввод и вывод данных, мониторинг состояния системы и управление балансировкой нагрузки.

1.1.3 Выбор параметров оптимизации и алгоритма оптимизации

Основным параметром оптимизации в данной системе является время обработки поступающих сообщений. Цель — минимизация задержки при распределении задач между микроконтроллерами, что достигается за счёт эффективного распределения нагрузки. Для этого необходимо:

- оптимизация времени обработки;
- эффективное распределение нагрузки.

Сокращение времени, необходимого для получения, обработки и распределения каждой входящей задачи.

Эффективное распределение нагрузки обеспечивает равномерное распределение задач между микроконтроллерами для предотвращения перегрузки отдельных узлов и максимального использования ресурсов всей системы.

Для реализации эффективной балансировки нагрузки был выбран алгоритм Round Robin. Этот алгоритм заключается в последовательном распределении задач между доступными микроконтроллерами, обеспечивая равномерное распределение нагрузки и предотвращая перегрузку отдельных узлов.

Алгоритм Round Robin обладает рядом значительных преимуществ, которые делают его оптимальным выбором для балансировки нагрузки в распределённой системе. Во-первых, он отличается простой реализацией и не требует сложных вычислений или хранения дополнительных данных, что существенно упрощает разработку и обслуживание системы. Благодаря своей природе, Round Robin обеспечивает равномерное распределение нагрузки между всеми подключёнными микроконтроллерами, предотвращая перегрузку отдельных узлов и способствуя эффективному использованию ресурсов всей сети. Более того, последовательный подход алгоритма обеспечивает прозрачность и предсказуемость распределения задач, что облегчает процесс отладки и мониторинга системы. Отсутствие необходимости в постоянном учёте текущего состояния каждого узла снижает общую сложность системы и повышает её надёжность.

В контексте данной системы применение алгоритма Round Robin реализуется следующим образом: при поступлении новой задачи главный микроконтроллер (мастер) направляет её к следующему микроконтроллеру-слейву в циклическом порядке. Это гарантирует равномерное распределение задач между всеми 112 микроконтроллерами, обеспечивая, что ни один из них не будет перегружен, пока остальные находятся в состоянии ожидания или выполняют свои задачи. Такой подход способствует оптимизации общего времени обработки задач и повышению производительности распределённой

системы, обеспечивая эффективную балансировку нагрузки и максимальное использование потенциала всех подключённых устройств.

1.1.4 Внешние компоненты

Исходя из функциональных требований системы, разработанной для балансировки нагрузки в распределённой системе, необходимо обеспечить удобный и надёжный ввод данных как через персональный компьютер (ПК), так и через мобильный телефон. Для реализации этого функционала используется интерфейс UART (Universal Asynchronous Receiver/Transmitter) микроконтроллера AVR, что обосновано его широкими возможностями по организации последовательной связи и простотой интеграции с различными устройствами. UART позволяет эффективно передавать данные между микроконтроллером и внешними устройствами, обеспечивая необходимую скорость и надёжность передачи информации.

Для связи с внешними устройствами МК использует UART-интерфейс, реализованный через последовательный порт RS-232. Для преобразования сигналов уровня TTL, используемых микроконтроллером, в стандартизированные сигналы RS-232, применяется драйвер MAX232. Этот компонент играет ключевую роль в обеспечении надёжной передачи данных между системой балансировки нагрузки и внешними устройствами, такими как компьютер или смартфон. MAX232 преобразует низковольтные логические сигналы микроконтроллера в высоковольтные сигналы RS-232, что соответствует требованиям стандартов последовательной связи и предотвращает возможные повреждения микроконтроллера из-за несоответствия уровней напряжений.

Дополнительно для обеспечения беспроводного взаимодействия с мобильным устройством используется Bluetooth-модуль HC-05. Этот модуль позволяет принимать команды от пользователя и передавать данные о состоянии системы без необходимости использования проводных соединений. HC-05 интегрируется с микроконтроллером через интерфейс UART,

обеспечивая стабильную и безопасную беспроводную связь. Это значительно расширяет функциональные возможности системы, позволяя пользователю управлять балансировщиком нагрузки и получать актуальную информацию о состоянии распределённой системы непосредственно со своего смартфона.

Для отображения отладочной информации и мониторинга состояния системы необходима возможность подключения графического ЖК-дисплея. ЖК-дисплей обеспечивает визуализацию результатов балансировки нагрузки, текущего состояния микроконтроллеров и других важных параметров системы в реальном времени. Наличие дисплея облегчает отладку и мониторинг работы системы, позволяя инженерам и пользователям быстро выявлять и устранять возможные проблемы, а также получать наглядную информацию о работе распределённой системы без необходимости постоянного обращения к ПК или мобильному устройству.

Кроме того, для обеспечения связи между микроконтроллерами в распределённой системе, как уже сказано ранее необходимо наличие интерфейса I2C.

Также, для обеспечения возможности быстрого и безопасного восстановления работы системы в случае сбоя предусмотрена кнопка сброса (Reset). Кнопка Reset позволяет пользователю быстро перезагрузить микроконтроллер, возвращая систему в исходное состояние без необходимости отключения питания.

Таким образом, интеграция UART с драйвером MAX232, Bluetooth-модулем HC-05, графическим ЖК-дисплеем, интерфейсом I2C и кнопкой Reset обеспечивает гибкость и надёжность взаимодействия микроконтроллерной системы с внешними устройствами.

1.1.5 Выбор элементной базы

Для выполнения поставленной задачи было решено использовать микроконтроллер ATmega16 от компании Atmel (ныне часть компании Microchip Technology). Данный микроконтроллер обладает широкими

возможностями по вводу и выводу данных, что делает его идеальным выбором для реализации балансировщика нагрузки в распределённой системе. ATmega16 оснащён четырьмя параллельными восьмиразрядными портами (PORTA, PORTB, PORTC, PORTD), что обеспечивает гибкость в подключении различных периферийных устройств. Все линии портов могут программироваться на ввод или вывод данных независимо друг от друга и имеют возможность подключения ко всем входам внутренних подтягивающих резисторов сопротивлением от 20 до 50 кОм. Это позволяет эффективно управлять состояниями входов и выходов без необходимости использования дополнительных внешних компонентов.

Данный микроконтроллер также оснащён мощным контроллером прерываний, который позволяет быстро и удобно обрабатывать как внешние прерывания (например, нажатие кнопки), так и внутренние прерывания (например, от таймера). Это критически важно для своевременного реагирования системы на изменения нагрузки и обеспечения стабильной работы распределённой системы. Кроме того, ATmega16 поддерживает встроенный канал UART, который необходим для связи с компьютером и со смартфоном. Это обеспечивает надёжную и эффективную передачу данных между микроконтроллером и внешними устройствами, такими как персональный компьютер или мобильный телефон, что является ключевым для ввода и вывода необходимой информации в рамках проекта.

Таким образом, делаем вывод, что выбранный микроконтроллер ATmega16 обладает всем необходимым функционалом для реализации проекта балансировщика нагрузки в распределённой системе. Его многофункциональные порты ввода/вывода, мощный контроллер прерываний и поддержка UART-интерфейса обеспечивают высокую гибкость и надёжность работы системы. Более подробное описание и характеристики используемого микроконтроллера представлены в разделе 1.2.1.

1.1.6 Проектирование структурной схемы

Учитывая вышесказанное, имеем следующие устройства/внешние компоненты, необходимые для реализации балансировщика нагрузки в распределённой системе:

- микроконтроллер ATmega16;
- жидкокристаллический дисплей для отображения необходимой информации;
- разъем RS-232 и драйвер MAX232, необходимые для связи с внешними ПЭВМ;
- модуль HC-05, необходимый для управления по Bluetooth;
- шина I2C, обеспечивающая связь в РС.

В этом списке нет компонента, благодаря которому все устройство будет получать напряжение для его работы. На этапе проектирования структурной схемы обобщим всю схему питания в один блок под названием «Блок питания». Также в этом списке нет всех необходимых компоненты, которые необходимы для работы итогового устройства, однако, этого достаточно для проектирования структурной схемы. Обобщенная структурная схема представлена на рисунке 1.

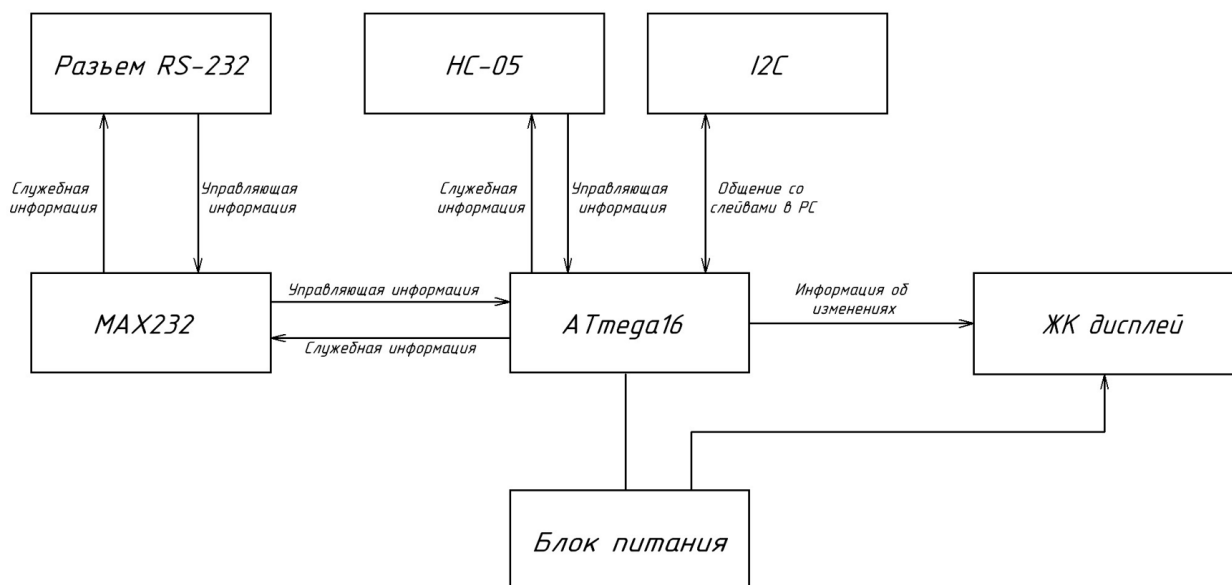


Рисунок 1 - Структурная схема устройства

Представленная структурная схема, показывает общий принцип работы итогового устройства.

1.2 Проектирование функциональной схемы

1.2.1 Микроконтроллер ATmega16

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

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

Сравнение семейств показано в таблице 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 инструкцию	4 такта на инструкцию	1 такт на инструкцию	1 такт на инструкцию
Память	ROM, SRAM, FLASH	SRAM, FLASH	Flash, SRAM, EEPROM	Flash, SDRAM, EEPROM
Энергопотребление	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 проекте минимизации булевых функций важна скорость обработки данных и выполнения операций, что могут предоставить МК серии AVR. Также они обладают достаточным объемом

памяти и встроенными периферийными устройствами, необходимыми для работы с UART и графическим дисплеем. Опыт работы с МК семейства AVR дополнительно упрощает разработку системы, так как нет необходимости изучать функционал нового микроконтроллера.

AVR в свою очередь делятся на семейства TinyAVR, MegaAVR, XMEGA AVR.

TinyAVR имеют 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.

MegaAVR, имеют 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.

XMEGA AVR, имеют 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).

Было выбрано подсемейство MegaAVR, так как оно обладает достаточным объемом памяти и количеством портов ввода-вывода для реализации проекта. XMEGA AVR, несмотря на свои расширенные возможности, избыточны для данной задачи, а TinyAVR недостаточны по ресурсам.

В подсемействе MegaAVR семейства AVR был выбран микроконтроллер ATmega16, обладающий всем необходимым функционалом для реализации проекта:

- интерфейс UART для связи с ПК через виртуальный терминал;
- достаточное количество пинов ввода-вывода для подключения и управления графическим ЖК-дисплеем;
- 1 Кбайт оперативной памяти (RAM), необходимой для обработки данных и временного хранения информации;
- 26 Кбайт Flash-памяти для размещения программы, включая алгоритмы минимизации и функции визуализации;
- поддержка внешних и внутренних прерываний, что позволяет организовать управление пошаговым режимом через кнопки;
- число пинов ввода-вывода: 32, что обеспечивает гибкость при подключении периферийных устройств;
- частота работы до 16 МГц, что позволяет обеспечивать быструю обработку данных.

Выбор микроконтроллера ATmega16 обусловлен его сбалансированными характеристиками, которые соответствуют требованиям проекта по балансировке нагрузки в распределённой системе. ATmega16 предлагает достаточный объем Flash-памяти и оперативной памяти для хранения и выполнения необходимых алгоритмов распределения нагрузки, а также управления взаимодействием с внешними устройствами через UART и I2C интерфейсы. 32 пина ввода-вывода позволяют гибко подключать графический ЖК-дисплей, модуль Bluetooth HC-05, интерфейс I2C для связи со слейвами в распределённой системе и другие периферийные устройства без необходимости использования дополнительных микроконтроллеров. Поддержка внешних и внутренних прерываний обеспечивает оперативную реакцию системы на внешние команды и внутренние события, что критично для эффективной балансировки нагрузки и поддержания стабильной работы системы. Кроме того, ATmega16 отличается низким энергопотреблением и

высокой производительностью благодаря тактовой частоте до 16 МГц, что позволяет системе быстро обрабатывать входящие данные и распределять задачи между микроконтроллерами-слейвами. Опыт работы с AVR микроконтроллерами упрощает разработку и отладку системы, позволяя сосредоточиться на реализации алгоритмов балансировки нагрузки и интеграции внешних компонентов.

Таким образом, ATmega16 предоставляет оптимальный баланс между производительностью, доступностью и необходимыми ресурсами для реализации поставленных задач. Структурная схема МК показана на рисунке 2 и УГО на рисунке 3 [2].

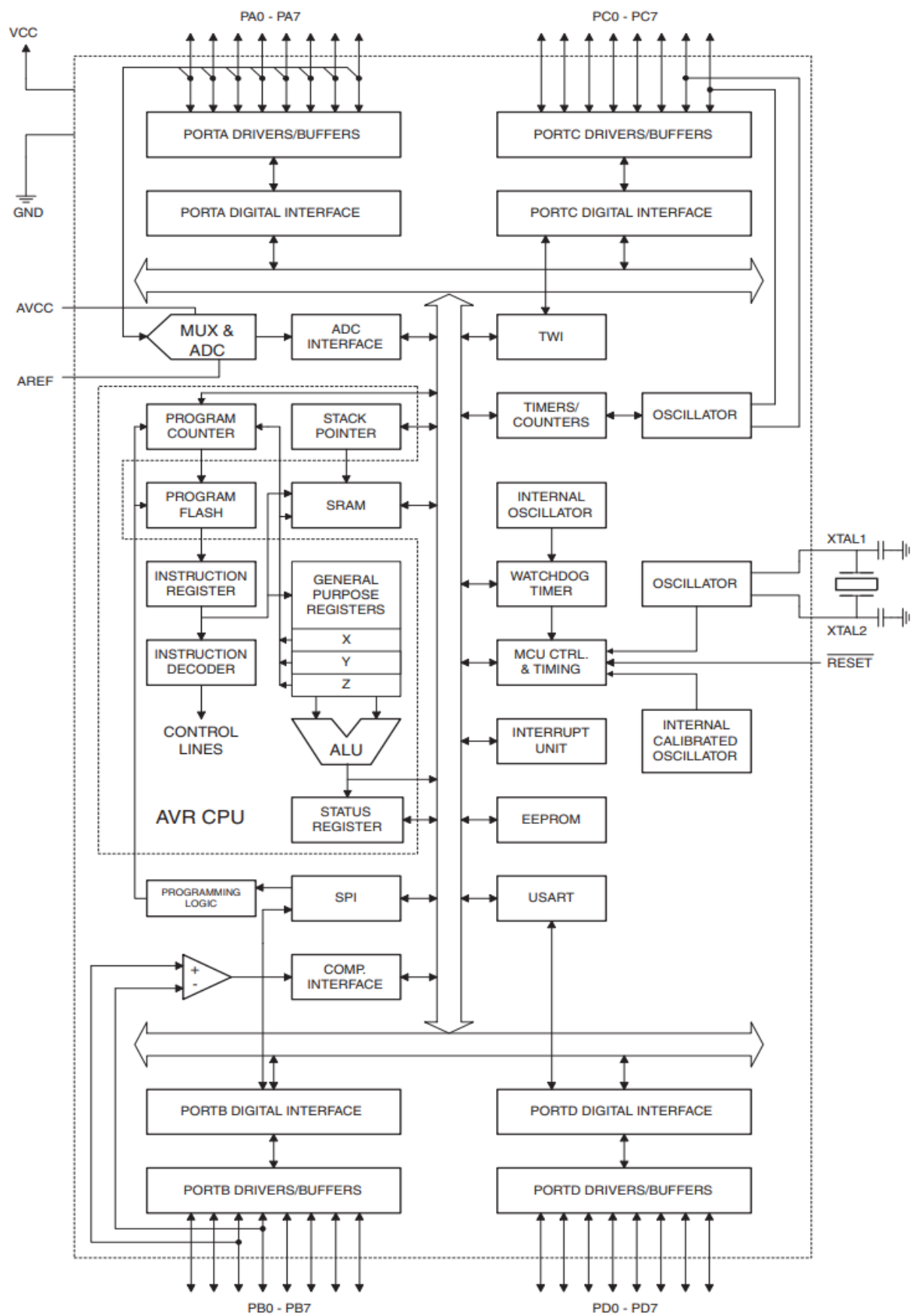


Рисунок 2 - Структурная схема МК АТmega16

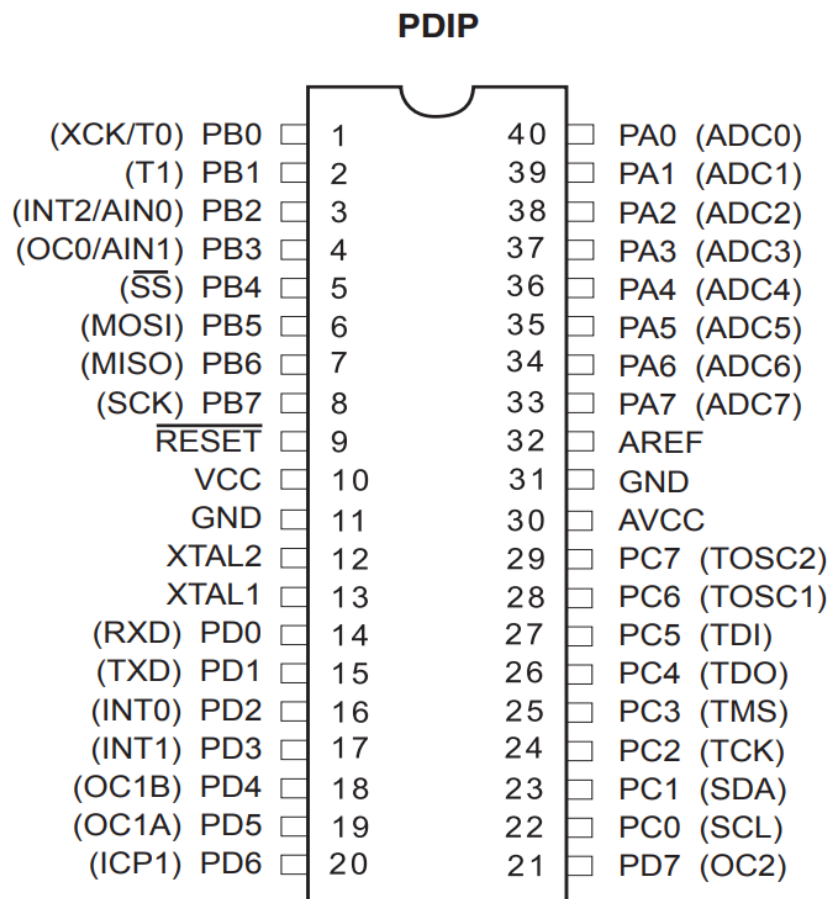


Рисунок 3 - УГО МК ATmega16

Он обладает следующими характеристиками [2]:

1. RISC архитектура:

- поддержка мощной системы команд с 131 инструкцией, большинство из которых выполняются за один машинный цикл;
- 32 восьмиразрядных рабочих регистра общего назначения;
- производительность до 16 миллионов операций в секунду при частоте 16 МГц;
- встроенное умножение за 2 цикла.

2. Энергонезависимые память программ и данных:

- 16 Кбайта внутрисхемно-программируемой Flash-памяти с возможностью самозаписи;
- 512 байта EEPROM с циклом записи/чтения до 100 000 раз;
- 1 Кбайта внутреннего SRAM;
- возможность записи и чтения данных без перезаписи всей памяти;

- поддержка блокировки программного обеспечения для повышения безопасности.

3. Периферийные устройства:

- два 8-разрядных таймера-счетчика с отдельными предделителями и режимами компаратора;
- один 16-разрядный таймер-счетчик с отдельным предделителем, режимами компаратора и захвата;
- реальный таймер с отдельным генератором;
- четыре канала широтно-импульсной модуляции (ШИМ);
- 8-канальный 10-битный АЦП, поддерживающий 8 однополюсных каналов и 2 дифференциальных канала с программируемым усилением;
- программируемый последовательный USART для асинхронной и синхронной передачи данных;
- интерфейсы SPI и I2C;
- программируемый сторожевой таймер с отдельным генератором;
- встроенный аналоговый компаратор.

4. Специальные функции микроконтроллера:

- сброс при подаче питания и программируемый супервизор питания;
- встроенный калиброванный RC-генератор;
- источники внешних и внутренних прерываний;
- шесть режимов энергосбережения: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, Extended Standby.

5. Ввод-вывод:

- 32 программируемых линий ввода-вывода;
- 40 регистров ввода/вывода.

6. Напряжение питания: 4.5 – 5.5В.

7. Рабочая частота: 0 – 16 МГц.

1.2.1.1 Используемые элементы

Для функционирования проектируемого устройства в МК ATmega16 задействованы не все элементы его архитектуры. Выделим и опишем те, что используются для его функционирования [2].

Порты A, B, C, D – обеспечивают подключение таких периферийных устройств, как ЖК-дисплей и кнопки, а также используются для реализации обмена данными с ПЭВМ, телефоном и виртуальным терминалом по UART. Назначения каждого из портов описано в разделе 1.2.1.2.

Указатель стека – используется для работы со стеком, при вызове процедур и функций, используемых в программном коде.

SRAM – статическая память МК, где хранятся объявленные в программе переменные.

EEPROM – энергонезависимая память МК, предназначенная для хранения постоянных данных программы.

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

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

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

Программный счетчик – используется для указания следующей команды выполняемой программы.

Память Flash – память МК, в которой хранится выполняемая программа.

Регистры команды – содержит исполняемую в текущий момент (или следующий) команду, то есть команду, адресуемую счетчиком команд.

Декодер – блок, выделяющий код операции и операнды команды, а затем вызывающий микропрограмму, которая исполняет данную команду.

Сигналы управления – синхронизируют обработку данных.

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

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

Прерывания – контроллер прерываний обрабатывает внешние прерывания и прерывания от периферийных устройств МК (таймеров, портов ввода/вывода). В проекте запрос на внешнее прерывание возникает по нажатию кнопок управления.

UART – через этот интерфейс реализован обмен данными между МК с ПЭВМ и виртуальным терминалом.

Генератор – внешний кварцевый резонатор с частотой 8 МГц обеспечивает стабильную синхронизацию работы микроконтроллера, что необходимо для высокой скорости выполнения операций, включая работу алгоритма минимизации и управления дисплеем.

1.2.1.2 Распределение портов

Микроконтроллер ATmega16 содержит четыре порта – А, В, С, D. Опишем назначение каждого из них, используемого в данной системе для её функционирования.

Порт А:

- PA0–PA7 – вывод данных на линии DA0–DA7 графического дисплея.

Порт С:

- PC0 – линия SCL I2C, отвечает за синхронизацию передачи данных между устройствами;

- PC1 – линия SDA I2C, отвечает за передачу данных между устройствами;

- PC2 – линия RS дисплея, определяет, какой тип данных будет передаваться на дисплей (инструкции или данные);

- PC4 – линия E дисплея, активирует дисплей для передачи данных.

Порт D:

- PD0 – линия RXD UART, используется для получения данных от ПЭВМ или телефона;

- PD1 – линия TXD UART, используется для передачи данных ПЭВМ или телефону;

Распределение портов обеспечивает корректную работу всех компонентов системы: ввод и обработку данных через UART, взаимодействие с графическим дисплеем.

1.2.1.3 Организация памяти

Микроконтроллер ATmega16 обладает развитой системой памяти, разделенной на память программ и память данных. Схема организации памяти представлена на рисунке 4.

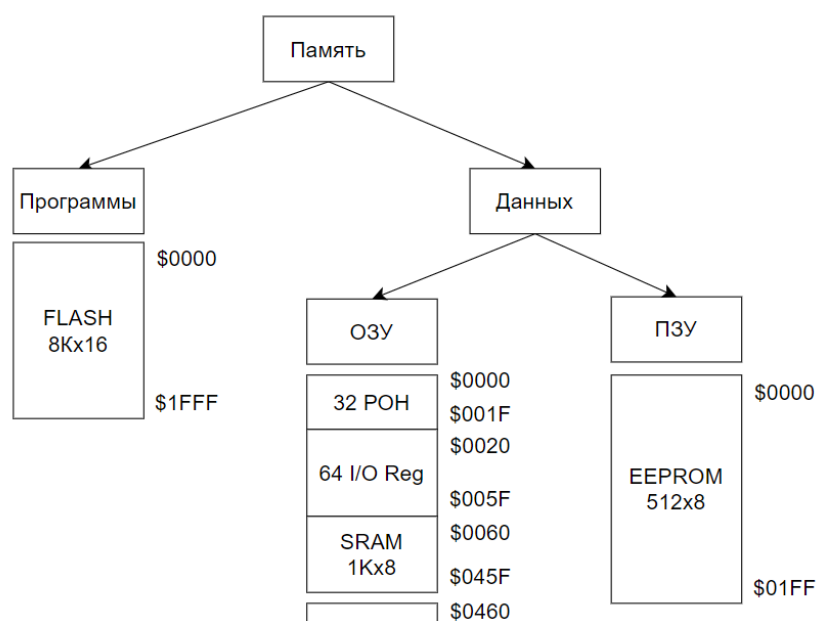


Рисунок 4 - Организация памяти МК ATmega16

Память программ предназначена для хранения последовательности команд, управляющих функционированием микроконтроллера. Она имеет 16-битную организацию, что означает длину одной команды в 16 разрядов.

МК ATmega16 имеет 16 Кбайта Flash-памяти. Информацию в flash-память можно мгновенно стереть и записать. В память микроконтроллера программа записывается с помощью.

Память данных делится на три основные части:

1. Регистровая память – 32 регистра общего назначения и служебные регистры ввода/вывода.

2. Оперативная память (ОЗУ) – МК ATmega16 имеет объем внутреннего SRAM 1 Кбайт (с адреса \$0060 до \$045F). Число циклов чтения и записи в RAM не ограничено, но при отключении питания вся информация теряется. Также есть возможность подключить к МК ATmega16 внешнее ОЗУ объемом до 64Кбайт.

3. Энергонезависимая память (EEPROM) – эта память доступна МК в ходе выполнения, она предназначена для хранения промежуточных результатов. В выбранном МК ее объем составляет 512 байт. Также в неё могут быть загружены данные через программатор.

1.2.2 Внешняя периферия

Как уже было проанализировано ранее, нам необходим ряд внешней периферии: дисплей, разъем RS-232, Bluetooth-модуль HC-05, I2C. В этом разделе будут подробно рассмотрены внешние компоненты, используемые для работы устройства.

1.2.2.1 Интерфейс RS-232 и модуль MAX232

Прием данных от персонального компьютера осуществляется через последовательный интерфейс UART. В примере реального подключения используется драйвер MAX232, который преобразует уровни сигналов RS-232 в TTL (цифровые сигналы). Это позволяет корректно передавать данные между физическим COM-портом компьютера и микроконтроллером.

RS-232 – стандарт физического уровня для синхронного и асинхронного интерфейса (USART и UART). Обеспечивает передачу данных и некоторых специальных сигналов между терминалом и устройством приема. Сигнал,

поступающий от интерфейса RS-232, через преобразователь передается в микроконтроллер на вход RxD.

Для физического подключения интерфейса RS-232 используется стандартный 9-контактный разъем DB-9. Он обеспечивает передачу данных через контакты TXD и RXD, а также предоставляет общий провод (GND). На рисунке 5 представлен внешний вид порта разъема, назначение каждого контакта разъема описано в таблице 2.

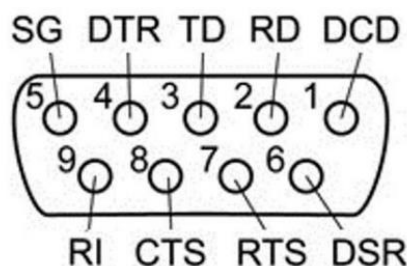


Рисунок 5 - Внешний вид и назначение контактов разъема DB-9

Таблица 2 - Назначение контактов разъема DB-9

Номер	Имя	Описание
1	DCD	Carrie Detect – определение несущей
2	RXD (RD)	Receive Data – принимаемые данные
3	TXD (TD)	Transmit Data – передаваемые данные
4	DTR	Data Terminal Ready – готовность терминала
5	SG (GND)	System Ground – корпус системы, заземление
6	DSR	Data Set Ready – готовность данных
7	RTS	Request To Send – запрос на отправку
8	CTS	Clear To Send – готовность приема
9	RI	Ring Indicator – «входящий вызов» на модеме

Внутреннее изображение MAX232 показано на рисунке 6. Назначение пинов описано в таблице 3 [3].

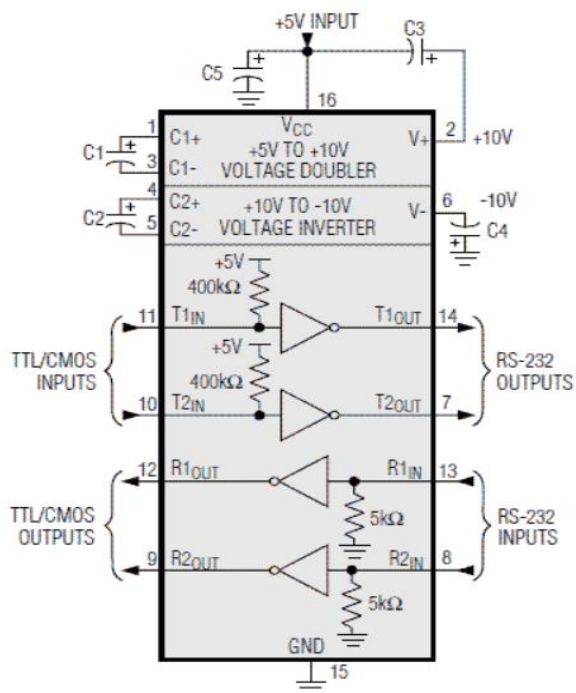


Рисунок 6 - Преобразователь MAX232

Таблица 3 - Назначение пинов MAX232

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

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

1.2.2.2 Bluetooth-модуль HC-05

Для беспроводного обмена данными разрабатываемого устройства и смартфона, с которого будет вестись управление, было решено использовать самый популярный и доступный Bluetooth-модуль – HC-05. На рисунке 7 представлен внешний вид HC-05, а также назначение контактов этого модуля.

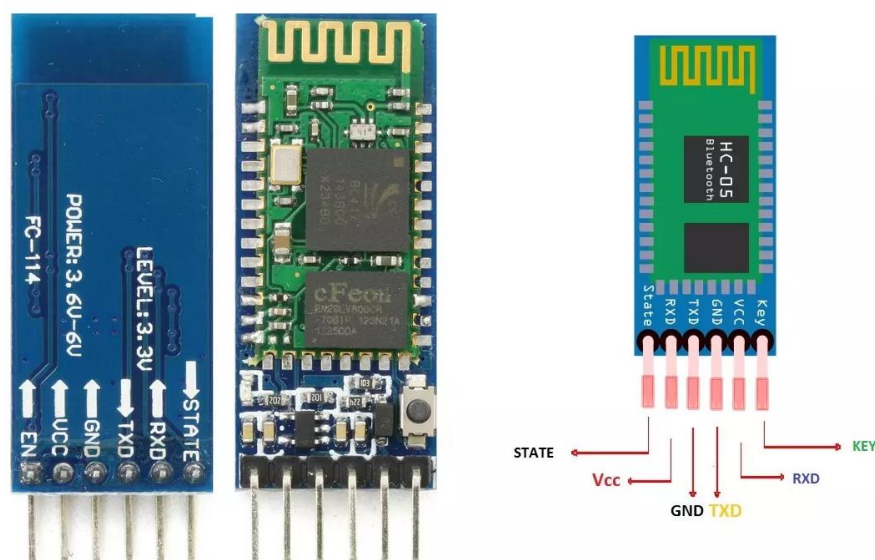


Рисунок 7 – Внешний вид и назначение контактов HC-05

Этот модуль имеет встроенную кнопку, зажатие которой эквивалентно подаче высокого напряжения на контакт EN (или KEY), необходимое для перехода в режим программирования, с помощью которого можно настроить Bluetooth-модуль. Настройка модуля происходит с помощью специальных AT-команд. С помощью таких команд можно установить различные параметры модуля: имя, которое видит устройство, желающее подключиться к этому модулю, пароль, режим подключения и другие.

Данный модуль обменивается сообщениями с подключенным устройством по UART, поэтому его необходимо подключить к контактам

микроконтроллера, отвечающих за передачу данных по UART, по схеме, представленной на рисунке 8.

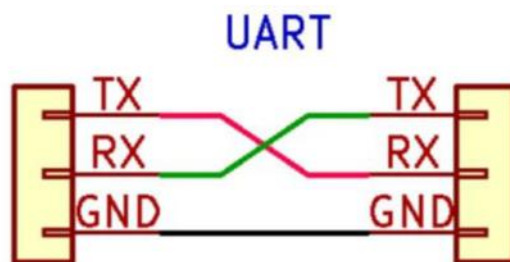


Рисунок 8 – Общая схема подключения устройств, обменивающихся сообщениями по каналу UART

1.2.2.3 Интерфейс I2C

В рамках разработки системы балансировки нагрузки в распределённой сети микроконтроллеров ATmega16 был выбран интерфейс I2C для организации связи между главным микроконтроллером и более чем 100 микроконтроллерами-слейвами.

Для проектирования интерфейса I2C в распределённой сети необходимо, чтобы все устройства в сети имели подключённые линии SCL и SDA. Это достигается объединением соответствующих пинов на главном микроконтроллере ATmega16 и всех микроконтроллеров-слейвов. Также, каждый микроконтроллер-слейв должен иметь уникальный адрес. В AVR микроконтроллерах адресация осуществляется путём настройки адресных пинов или программным образом. Пример I2C сети представлен на рисунке 9.

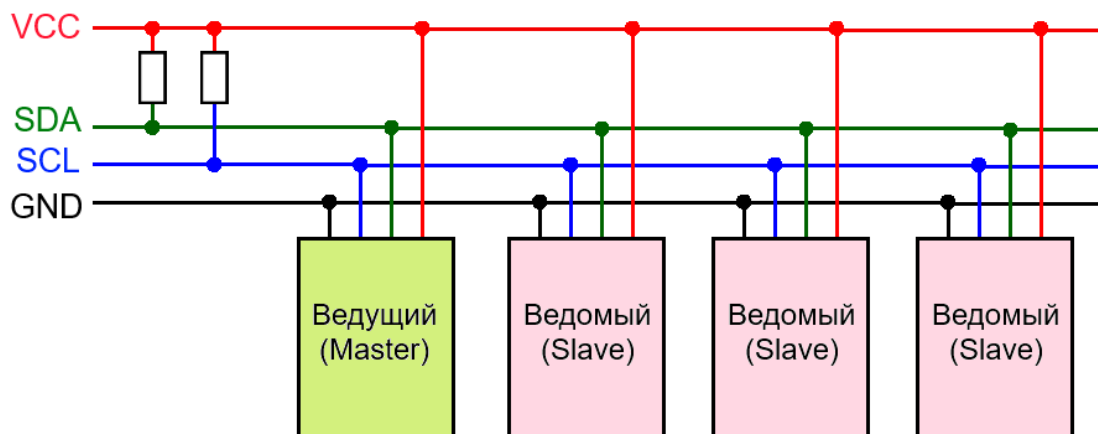


Рисунок 9 – Пример сети, соединенной I2C

1.2.2.4 LCD дисплей

Для отображения необходимой информации в системе балансировки нагрузки была выбрана конкретная модель LCD-дисплея – LM044L. Это жидкокристаллический дисплей (ЖКД) с разрешением 20x4 символов, способный отображать до 80 символов (20 символов в четырёх строках). На рисунке 10 представлена обобщённая структурная схема данного модуля, а на рисунке 11 – внешний вид LM044L и его распиновка.

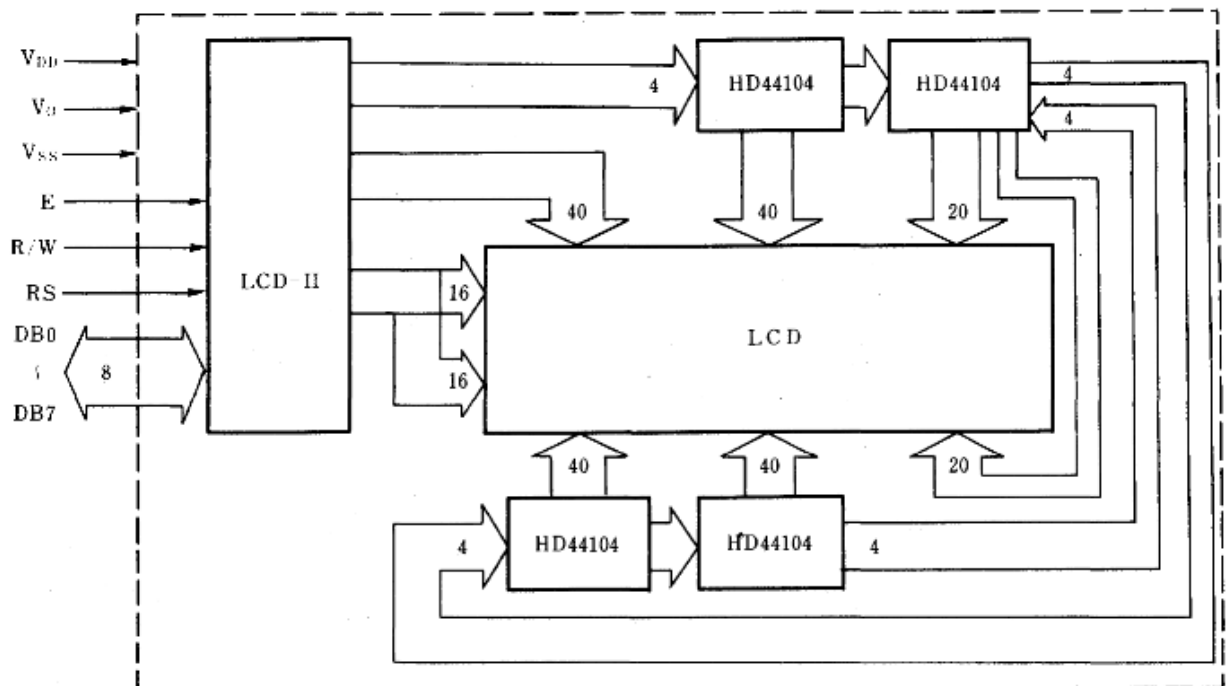


Рисунок 10 – Структурная схема LM044L

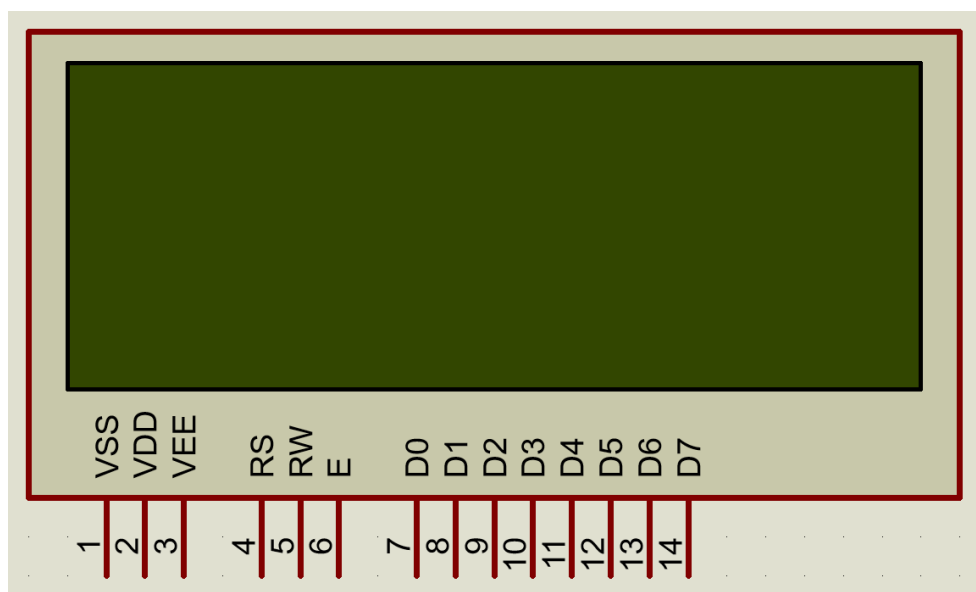


Рисунок 11 – Внешний вид и распиновка LM044L

Рекомендованное напряжение питания данного модуля – 5 В. При таком напряжении устройство способно работать с максимальной яркостью, обеспечивая чёткое и разборчивое отображение информации даже при высоком уровне освещённости в помещении. Рассмотрим назначение каждого пина данного модуля:

- D0-D7 – пины для передачи данных, то есть для управления отображаемыми символами;
- RS (Register Select) – используется для выбора между двумя типами регистров: если $RS = 0$, выбирается регистр инструкций (для записи команд в модуль), если $RS = 1$, выбирается регистр данных (для записи или чтения данных, которые будут отображаться);
- RW (Read/Write) – определяет режим чтения или записи: если $RW = 0$, производится запись в модуль, если $RW = 1$, производится чтение из модуля;
- E (Enable) – используется для активации записи данных или команд в модуль: передача данных происходит при переходе сигнала E с высокого уровня на низкий;
- VSS – этот вывод подключается к земле GND;
- VDD – этот вывод подключается к положительному источнику питания 5 В;
- VEE – этот вывод используется для регулировки контрастности дисплея и часто подключается к переменному резистору или потенциометру для настройки уровня контрастности, но при разработке данного проекта этот вывод не используется, так как мы используем максимальную яркость, которая и устанавливается по умолчанию.

1.2.3 Подключение двух устройств, работающих по UART

К микроконтроллеру может быть подключено два устройства, работающих по каналу UART: Bluetooth-модуль HC-05 и разъем RS-232. Все

устройства, подключенные к шине UART, являются приемопередатчиками, то есть они могут как отправлять сообщения, так и получать их: устройство, подключенное к RS-232 или к Bluetooth-модулю отправляет команды и получает ответ, а микроконтроллер получает команды и отправляет ответы.

Изначально UART предназначался для связи двух устройств по принципу «точка-точка». Впоследствии были созданы физические уровни, которые позволяют связывать более двух UART по принципу «один говорит – несколько слушают» [18].

В разрабатываемом устройстве подключено более двух устройств, поэтому принцип «точка-точка» не работает. Все устройства являются приемопередатчиками, поэтому второй принцип тоже не работает. Таким образом, мы не можем одновременно подключить RS-232 и HC-05 к шине UART, идущей от микроконтроллера. Поэтому было решено использовать переключатель, который будет определять, какое именно устройство сейчас обменивается с микроконтроллером сообщениями. Хотя шина UART состоит из двух линий, было решено использовать переключатель типа SPDT – один полюс, два направления. В данной конфигурации приём данных осуществляется только от одного устройства, что предотвращает возможные конфликты и обеспечивает корректную обработку входящей информации. Одновременно с этим, передача данных от микроконтроллера осуществляется сразу на два устройства. Это решение обусловлено необходимостью обеспечения одновременной доступности данных для двух внешних систем, например, персонального компьютера и мобильного телефона. Таким образом, данные от микроконтроллера доступны обоим устройствам одновременно, что позволяет при переключении управления определить, какое из устройств получило ввод последним. На рисунке 12 представлено схематическое обозначение данного переключателя.

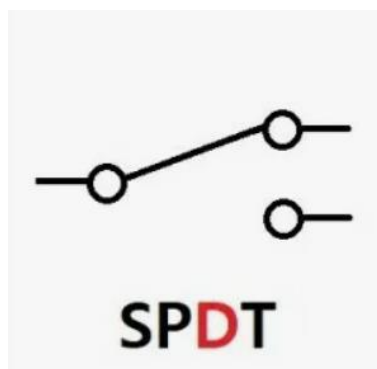


Рисунок 12 – SPDT-переключатель

1.2.4 Настройка передачи данных по каналу UART

Для передачи информации между микроконтроллером ATmega16 и ПЭВМ используется последовательный интерфейс UART. Для его корректной работы необходимо настроить регистры управления UCSRA, UCSRB и UCSRC [2, 4].

UCSRA (регистр управления A). Биты регистра UCSRA показаны в таблице 4.

Таблица 4 – биты регистра UCSRA

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

В процессе обмена данными по UART в нашем проекте используются только биты RXC и UDRE [2].

RXC — флаг завершения приема. Устанавливается в 1 при наличии непрочитанных данных в буфере приемника. Используется для проверки данных перед их считыванием.

UDRE — флаг опустошения регистра данных. Устанавливается в 1, если буфер передатчика пуст и готов к записи новых данных.

UCSRB (регистр управления B). Биты регистра UCSRB показаны в таблице 5.

Таблица 5 – биты регистра UCSRB

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

Во время работы системы, для управления приемом и передачей информации, будут использоваться только биты TXEN и RXEN [2].

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

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

UCSRC (регистр управления C). Биты регистра UCSRC показаны в таблице 6.

Таблица 6 – биты регистра UCSRC

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

Во время работы системы будут использоваться только биты URSEL, UCSZ0 и UCSZ1 [2].

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

UCSZ0 и UCSZ1 – формат посылки. Каждый из них принимает значение 1, для 8 бит данных в посылке.

USBS – задает количество стоп-битов, устанавливаемых в посылке. При значении 0 используется 1 стоп-бит.

Скорость передачи определяется выражением [4]:

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

где BAUD — скорость передачи (бод);

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

UBRR — содержимое регистров UBRR0H, UBRR0L контроллера скорости передачи.

Зададим скорость 9600 бод. Получится:

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

UBRR0L будет принимать значение 51.

Итоговая настройка:

- UBRR0L будет принимать значение 51;
- включены прием и передача данных (RXEN = 1, TXEN = 1);
- формат кадра: 8 бит данных, 1 стоп-бит, без проверки четности.

1.2.5 Генератор тактовых импульсов

Для работы микроконтроллера ATmega16 необходим тактовый генератор, обеспечивающий синхронизацию выполнения команд и работы всех внутренних модулей. Хотя микроконтроллер оснащен встроенным генератором, для большей стабильности и точности работы используется внешний кварцевый резонатор с частотой 8 МГц. Резонатор подключается к выводам XTAL1 и XTAL2 [5], используя конденсаторы для корректной работы генератора.

Внешний резонатор подключен следующим образом:

- один вывод кварцевого резонатора подключается к XTAL1, а другой – к XTAL2;
- к обоим выводам добавлены конденсаторы емкостью 18 пФ, подключенные к земле. Эти конденсаторы необходимы для устойчивости генерации.

Частота 8 МГц выбрана для обеспечения высокой скорости выполнения инструкций и поддержания надежной работы алгоритма минимизации булевых функций.

1.2.6 Сброс микроконтроллера

Для корректного запуска микроконтроллера при включении питания и возможности ручного сброса реализована схема сброса с использованием RC-цепи. В микроконтроллере ATmega16 сигнал RESET подтянут к питанию через внутренний резистор сопротивлением 100 кОм. Однако для повышения надежности используется внешняя схема сброса.

Сигнал RESET дополнительно подключен к внешнему резистору (10 кОм), соединенному с V_{cc} , и конденсатору (100 мкФ) [5], соединенному с землей. Эта RC-цепь выполняет следующие функции:

- при включении питания конденсатор разряжен, и напряжение на RESET близко к нулю. Это удерживает микроконтроллер в состоянии сброса, предотвращая запуск в условиях нестабильного питания;
- по мере зарядки конденсатора через резистор напряжение на RESET постепенно увеличивается, достигая логической 1, после чего микроконтроллер начинает выполнение программы.

Для ручного сброса используется кнопка, подключенная между RESET и землей. При нажатии кнопки конденсатор разряжается, напряжение на RESET падает до 0, и микроконтроллер переходит в состояние сброса. После отпускания кнопки конденсатор вновь заряжается, плавно возвращая микроконтроллер в рабочее состояние.

Преимущества RC-цепи

- защита от скачков напряжения и нестабильности питания при включении;
- создание плавного запуска системы;
- возможность ручного сброса для повторного запуска программы.

1.2.8 Построение функциональной схемы

На основе всех вышеописанных сведений была спроектирована функциональная схема разрабатываемого устройства, показанная в приложении Б [6, 7].

1.3 Проектирование принципиальной схемы

1.3.1 Разъем программатора

Для программирования микроконтроллера ATmega16 используется программатор, для подключения которого необходим специальный разъем. В проекте будет использован разъем IDC-06MS, обеспечивающий подключение программатора через интерфейс SPI. Для реализации интерфейса SPI на микроконтроллере ATmega16 задействован порт PB. На принципиальной схеме, представленной в разделе 1.3.5, разъем будет обозначен как XP1.

Разъем включает следующие линии для подключения к микроконтроллеру:

- MISO (PB6) – для передачи данных от микроконтроллера в программатор;
- SCK (PB7) – тактовый сигнал, синхронизирующий передачу данных;
- MOSI (PB5) – для передачи данных от программатора в микроконтроллер;
- RESET – сигнал сброса, которым программатор переводит микроконтроллер в режим программирования.

Этот разъем обеспечивает надежное подключение программатора и позволяет выполнять программирование микроконтроллера без его извлечения из устройства (In-System Programming).

1.3.2 Подключение цепи питания

Для работы микроконтроллера ATmega16 и подключенных к нему периферийных устройств требуется стабильное напряжение питания. В данном проекте используется внешний блок питания ARDV-15-5B, обеспечивающий необходимую мощность и стабильность.

ARDV-15-5B имеет следующие технические характеристики:

- выходное напряжение: 5 В;
- выходной ток: 3 А;

- входное напряжение: 100–240 В (позволяет подключить блок питания к бытовой электрической сети);

- пусковой ток: 40 А (кратковременный скачок при включении питания).

Для подключения блока питания используется разъем DS-201, имеющий следующие характеристики:

- диаметр центрального проводника: 2 мм;

- диаметр Jack-разъема: 2 мм.

Блок питания подключается к разъему DS-201, который далее распределяет напряжение 5 В на все узлы устройства.

На принципиальной схеме, представленной в разделе разъем питания обозначен как XS1. Через этот разъем питание подается на линии VCC (+5 В) и GND, обеспечивая работу всех компонентов схемы.

1.3.3 Расчет сопротивления резисторов

В проекте используются резисторы для обеспечения стабильной работы схемы, рассчитаем номиналы резисторов, задействованных в проекте.

Резисторы RC-цепи сброса. Для реализации плавного сброса микроконтроллера при включении питания используется RC-цепь. Она включает резистор и конденсатор, подключенные к выводу RESET. Резистор ограничивает ток зарядки конденсатора, создавая задержку.

Стандартный номинал резистора для RC-цепи сброса составляет 10 кОм, что обеспечивает достаточную задержку для стабилизации питания перед запуском микроконтроллера.

Итоговые номиналы резисторов в проекте.

1. Резистор RC-цепи сброса:

- используется для плавного запуска микроконтроллера;

- номинал: 10 кОм.

1.3.4 Расчет потребляемой мощности

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

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

$$P = I * U,$$

где U – напряжение питания (В);

I – ток потребления микросхемы (мА).

Кроме того, в схеме присутствуют резисторы, для них мощность рассчитывается по формуле:

$$P = \frac{U^2}{R} = \frac{25 \text{ В}}{10000 \text{ Ом}} = 2,5 \text{ мВт},$$

где R – сопротивление резистора;

U – напряжение на резисторе.

Расчет потребляемого напряжения для каждой микросхемы показан в таблице 7.

Таблица 7 – Потребляемая мощность

Микросхема	Ток потребления, мА	Потребляемая мощность, мВт	Количество устройств	Суммарная потребляемая мощность, мВт
ATmega16	40	200	1	200
MAX232	8	40	1	40
НС-05	15	75	1	75
ЖК-дисплей LM044L	3	15	1	15

Также в схеме используются 1 резистор МЛТ-0.125 (1 резистор RC-цепи) мощностью 2,5 мВт.

Таким образом, суммарная мощность:

$$\begin{aligned} P_{\text{суммарная}} &= P_{\text{ATmega16}} + P_{\text{MAX232}} + P_{\text{НС-05}} + P_{\text{LM044L}} + P_{\text{резисторов}} = \\ &= 200 + 40 + 75 + 15 + 2,5 = 332,5 \text{ мВт} \end{aligned}$$

Суммарная потребляемая мощность системы равна 332,5 мВт.

1.3.5 Построение принципиальной схемы

На основе всех вышеописанных сведений была спроектирована принципиальная схема разрабатываемой системы, показанная в приложении Б [6, 7].

1.4 Алгоритмы работы системы

Перед разработкой алгоритмов работы разрабатываемого программного кода, необходимо определить, общий принцип работы устройства, а именно то, каким образом пользователь может управлять разрабатываемым устройством.

Как уже было сказано ранее, всего 112 МК в РС, пользователь управляет балансировщиком, который уже управляет 112 МК-слейвами. Итак, РС настроена заранее – на балансировщике уже есть 112 адресов слейвов. Пользователь же отправляет с ПЭВМ или телефона сообщение-задачу, которое балансировщик направляет слейвам. Также, пользователь может настроить РС – отключить ненужные слейвы или включить нужные слейвы. Также, пользователь может сохранить текущую конфигурацию системы в постоянную память и может очистить память, ещё он может перезапустить работу балансировщика, тогда при запуске конфигурация возьмется из памяти МК, либо создаться конфигурация по умолчанию. Ещё пользователь может выводить адрес текущего слейва – слейва на который балансировщик отправит ближайшую задачу. Также, пользователь может выводить список слейвов. И так, вот полный синтаксис команд, который МК принимает по UART:

- print [N] – отображает все активные адреса слейвов или следующие N слейвов, без указания N отображаются все активные адреса;
- enable N – включить слейв с номером N;
- disable N – отключить слейв с номером N;
- current – показать текущий активный адрес слейва;
- save – сохранить текущую конфигурацию в EEPROM;
- clear – очистить данные EEPROM;
- reset – программно сбросить микроконтроллер;
- help – показать это справочное сообщение;

- любой другой текст – по умолчанию, любая введенная строка будет трактоваться как сообщение для отправки текущему слайду, система будет балансировать нагрузку.

После определения принципа работы устройства были построены схемы алгоритмов работы микроконтроллера, которые полностью описывают логику работы микроконтроллера.

На рисунке 13 представлена схема алгоритма основной программы, запускаемой при включении устройства.

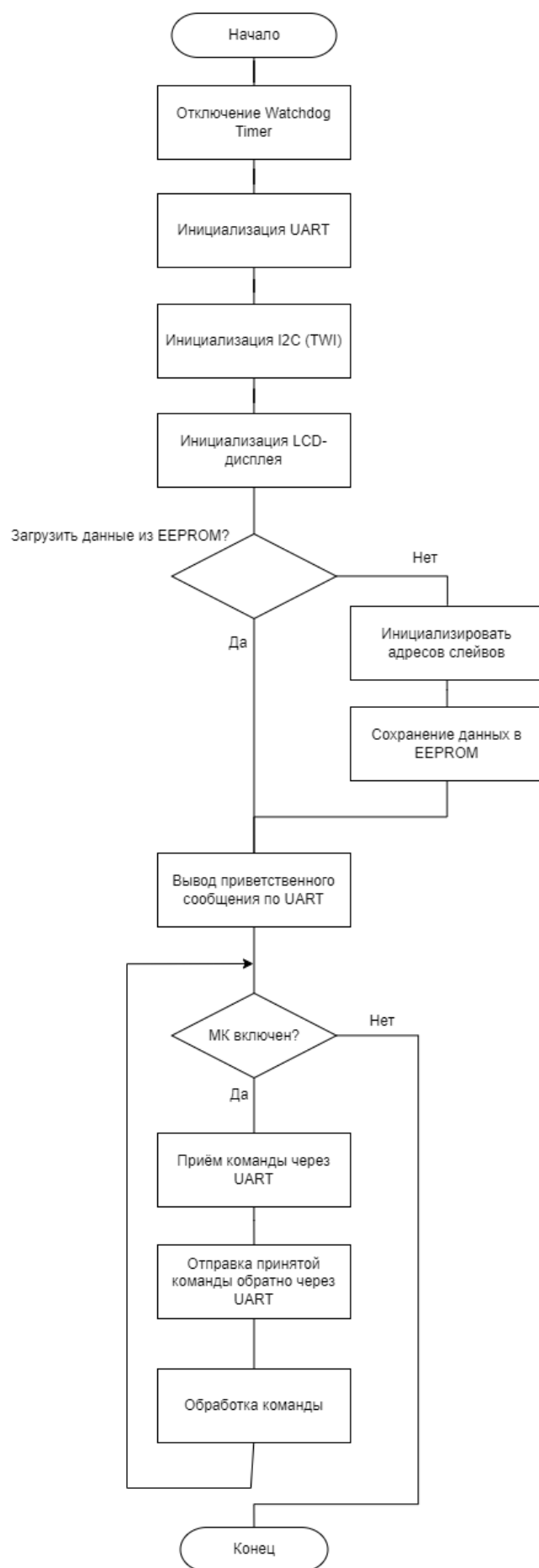


Рисунок 13 – Схема алгоритма основной программы

На основе составленных схем алгоритмов написан программный код на языке программирования Си, который представлен в приложении А (Исходный код программы микроконтроллера).

2 Технологическая часть

2.1 Отладка и тестирование программы

Программа была отлажена с использованием среды разработки Proteus 8 Professional. Это приложение предназначено для выполнения различных видов моделирования аналоговых и цифровых устройств, что позволило протестировать взаимодействие микроконтроллера с внешними устройствами, такими как ЖК-дисплей, ПЭВМ и телефон.

В Proteus нет встроенной поддержки модуля HC-05. В Интернете есть неофициальная библиотека с этим модулем, которую можно подключить к используемой среде, но она лишь является «оберткой» RS-232, то есть подключаемый модуль не отличается настройками и функционалом от RS-232, а отличается только внешним видом, который, в свою очередь, меняется обратно на внешний вид разъема RS-232 при запуске симуляции. Поэтому было решено использовать оригинальный эмулятор разъема RS-232, настроенный на работу по Bluetooth. Поэтому было принято решение использовать альтернативный способ передачи данных через TCP/IP соединение. Для этого на стороне ПК используется программа TCP2COM, которая создает виртуальный COM-порт и выступает в качестве TCP-сервера, пересылая все полученные по сети данные на виртуальный порт и обратно. На мобильном устройстве используется приложение TCP Client, позволяющее устанавливать TCP-соединение с сервером и обмениваться данными. Такой подход позволил обойти проблемы с Bluetooth-подключением и обеспечить надежный канал связи между мобильным приложением и эмулируемым устройством в среде Proteus.

Также, в качестве симуляции РС я использовал I2C-debugger, в него балансировщик отправляет данные слейвам.

В ходе отладки наглядно демонстрировалось, как микроконтроллер принимает данные через UART, обрабатывает их, выводит результаты на дисплей и отправляет данные на ПЭВМ и телефон.

При написании кода для микроконтроллера ATmega16 были использованы следующие библиотеки:

- `avr/io.h` – стандартная библиотека ввода/вывода, которая позволяет настраивать порты микроконтроллера, их функционал и режим работы;

- `util/delay.h` – библиотека, предоставляющая функции программных задержек. Функция `_delay_ms()` использовалась для создания временных пауз между инструкциями, что необходимо для корректной работы с ЖК-дисплеем и обработки входных данных;

- `util/twi.h` – библиотека, обеспечивающая низкоуровневую поддержку протокола I2C, включая функции для инициализации, отправки и приёма данных по шине I2C;

- `eeprom.h` – библиотека для работы с энергонезависимой памятью EEPROM. Она была использована для хранения текущего адреса слайва и текущих включенных и отключенных слайвов в PC№;

- `avr/pgmspace.h` – библиотека для работы с данными, хранящимися в памяти программы (Flash). Она была использована для хранения константных данных, таких как строки сообщений, в Flash-памяти вместо SRAM, экономя оперативную память микроконтроллера. Для чтения данных из Flash использовалась функция `pgm_read_byte()`;

- `stdio.h` – стандартная библиотека ввода-вывода языка C. Использовалась для форматирования строк с помощью функций, таких как `sprintf()`, что облегчает формирование сообщений для передачи через UART или отображения на LCD-дисплее;

- `stdlib.h` – стандартная библиотека ввода-вывода языка C. Предоставляет функции для работы с памятью, преобразования типов данных, генерации случайных чисел и другие общие утилиты. В данном коде использовалась, функция `atoi()` для преобразования строк в числа;

- `string.h` – стандартная библиотека ввода-вывода языка C. Обеспечивает функции для манипуляции строками, такие как `strcmp`, `strncmp`, `strcat`, что необходимо для обработки и сравнения команд, полученных через UART;

– `avr/wdt.h` – функции для управления Watchdog Timer. Использовалась для предотвращения зависаний микроконтроллера путем автоматического сброса системы в случае сбоя. В коде реализованы функции для включения и отключения Watchdog Timer.

В ходе отладки и тестирования программа проверялась в среде Proteus. Были протестированы:

- 1) корректность приема и обработки данных через UART;
- 2) вывод результатов на ЖК-дисплей;

Итогом работы стал стабильный программный код, который полностью соответствует техническому заданию. Подробности симуляции и проверки функционала приведены в разделе 2.2 Симуляция системы.

2.2 Симуляция работы системы

Для имитации работы системы использовалась программа Proteus 8 Professional. Схема, демонстрирующая взаимодействие микроконтроллера ATmega16 с внешними устройствами, такими как ЖК-дисплей LM044L, виртуальный терминал, COMPI и I2C-debugger, представлена на рисунке 14.

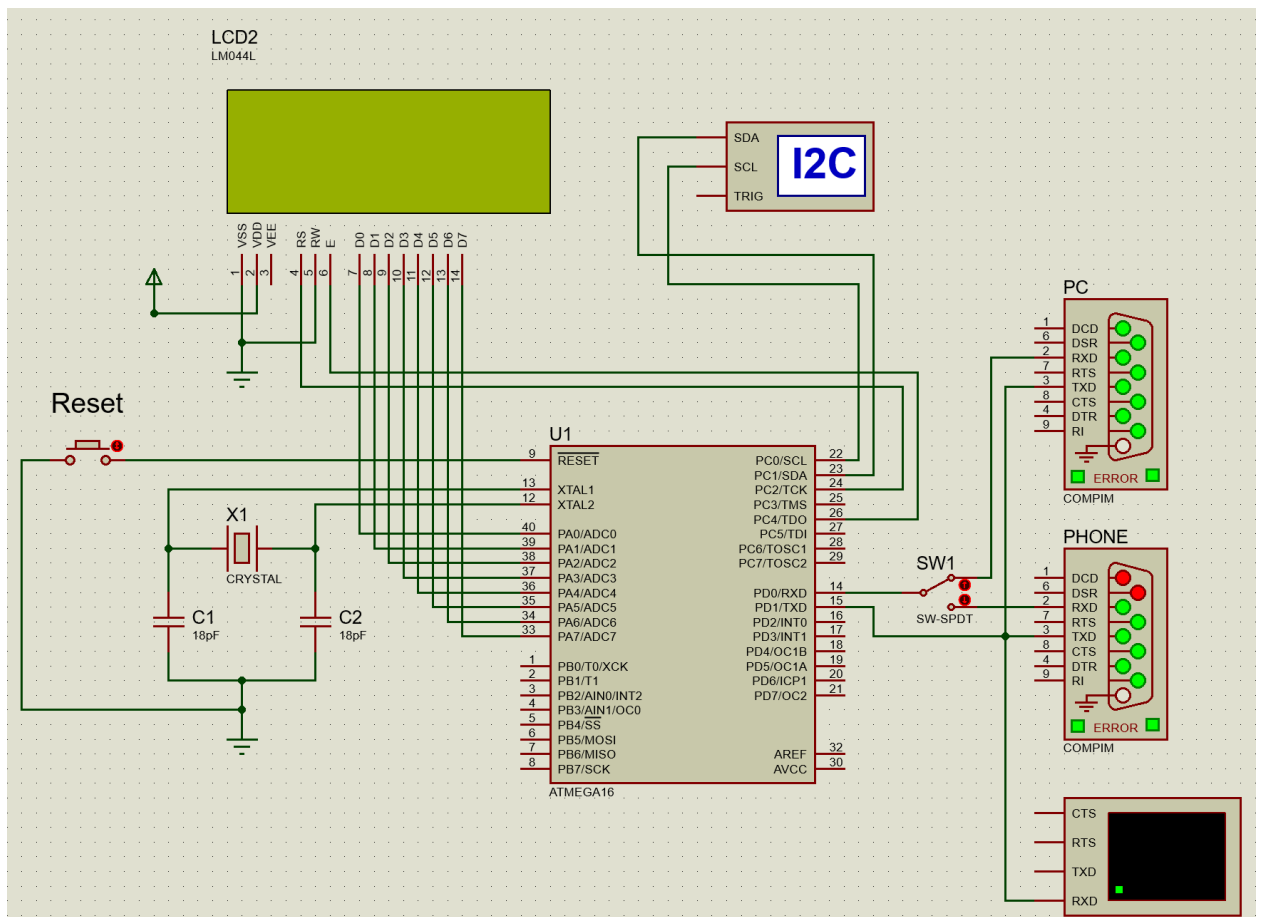


Рисунок 14 - Модель устройства

После начала работы на все устройства МК отправляет приветственное сообщение, данный этап показан на рисунке 15.

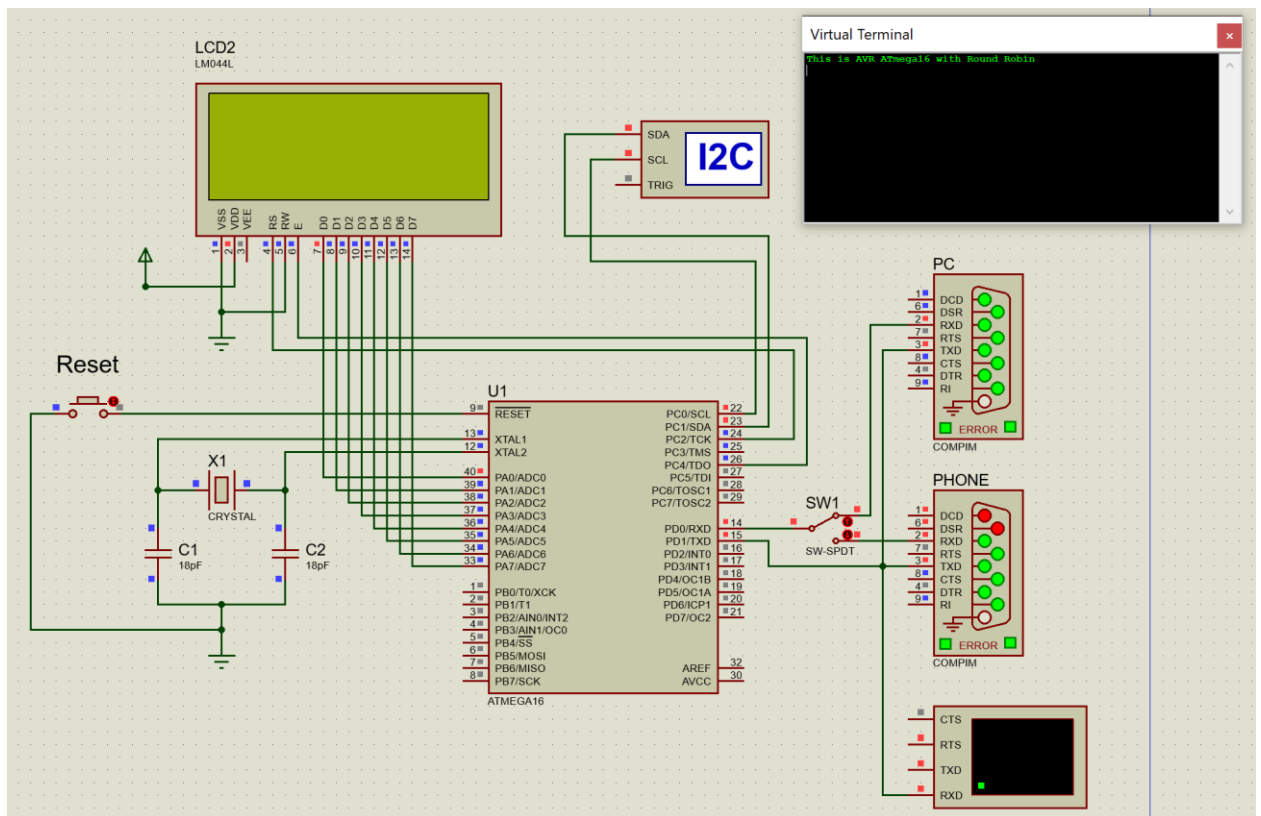


Рисунок 15 - Этап приветственного сообщения

Далее, всё взаимодействие происходит через терминал, поэтому на последующих рисунках будет только терминал.

Теперь пользователь ввёл команду «help», она выводит все доступные команды. Результат ввода показана на рисунке 16.

```

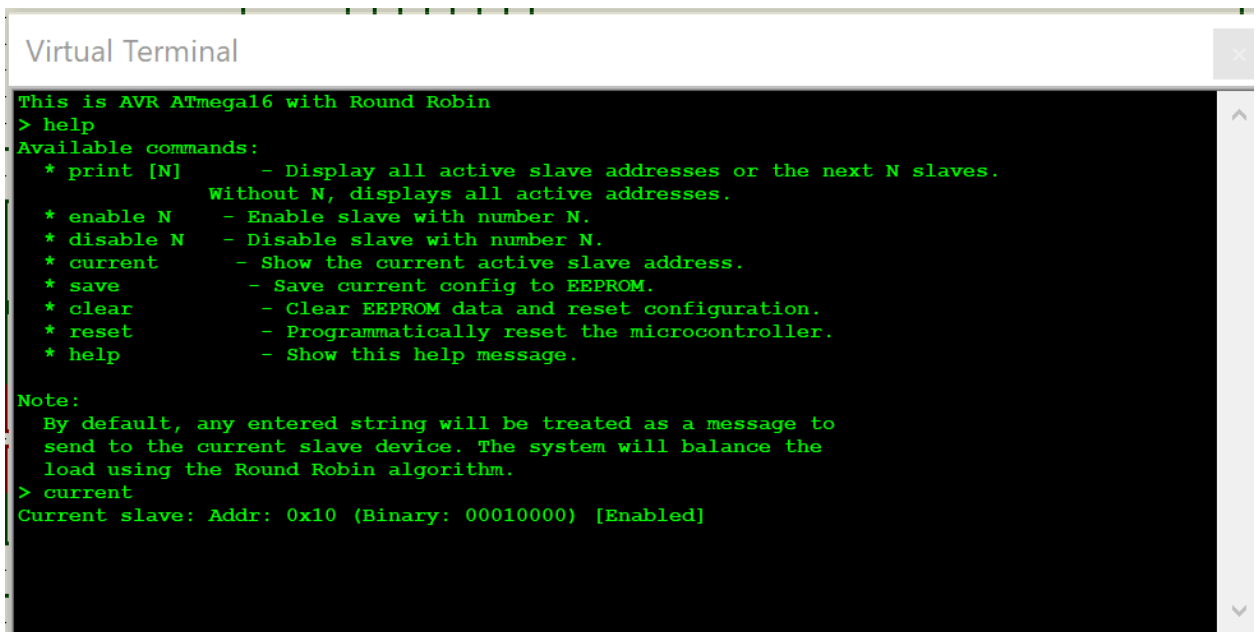
Virtual Terminal
This is AVR ATmega16 with Round Robin
> help
Available commands:
* print [N] - Display all active slave addresses or the next N slaves.
              Without N, displays all active addresses.
* enable N - Enable slave with number N.
* disable N - Disable slave with number N.
* current - Show the current active slave address.
* save - Save current config to EEPROM.
* clear - Clear EEPROM data and reset configuration.
* reset - Programmatically reset the microcontroller.
* help - Show this help message.

Note:
By default, any entered string will be treated as a message to
send to the current slave device. The system will balance the
load using the Round Robin algorithm.

```

Рисунок 16 - Ввод команды «help»

Теперь пользователь ввёл команду «current», она выводит адрес текущего слейва. Результат ввода показана на рисунке 17.

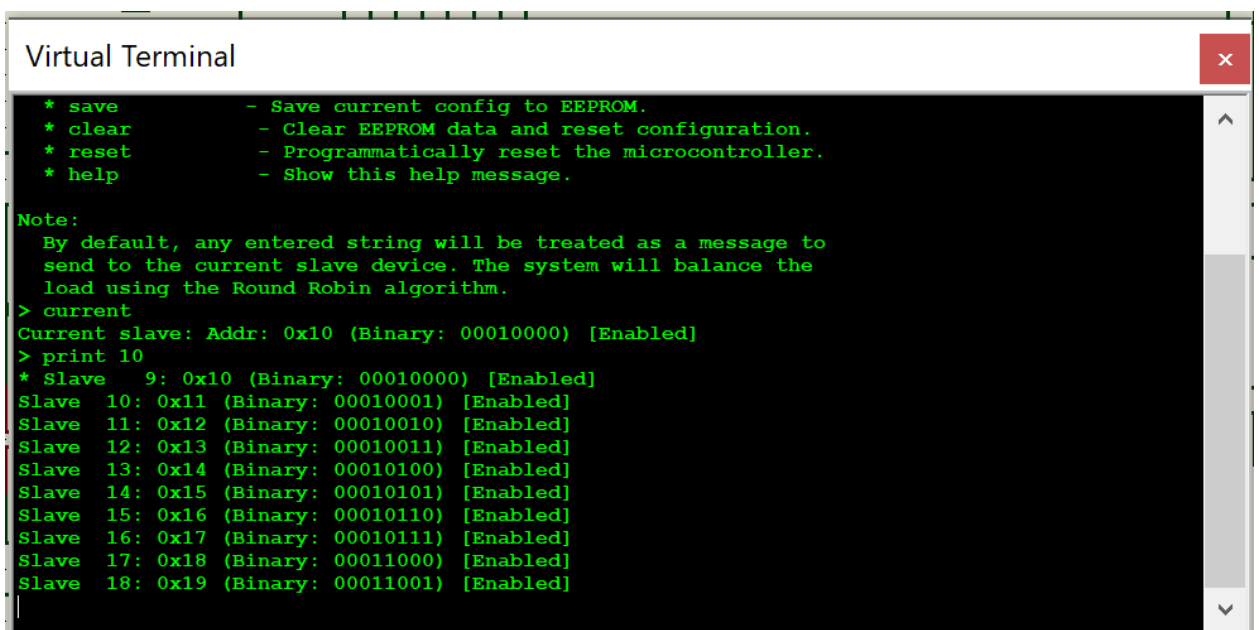


```
Virtual Terminal
This is AVR ATmega16 with Round Robin
> help
Available commands:
* print [N]      - Display all active slave addresses or the next N slaves.
                  Without N, displays all active addresses.
* enable N       - Enable slave with number N.
* disable N      - Disable slave with number N.
* current        - Show the current active slave address.
* save           - Save current config to EEPROM.
* clear          - Clear EEPROM data and reset configuration.
* reset          - Programmatically reset the microcontroller.
* help           - Show this help message.

Note:
By default, any entered string will be treated as a message to
send to the current slave device. The system will balance the
load using the Round Robin algorithm.
> current
Current slave: Addr: 0x10 (Binary: 00010000) [Enabled]
```

Рисунок 17 - Ввод команды «current»

Теперь пользователь ввёл команду «print N», она выводит следующие N
слейвов. Результат ввода показана на рисунке 18.

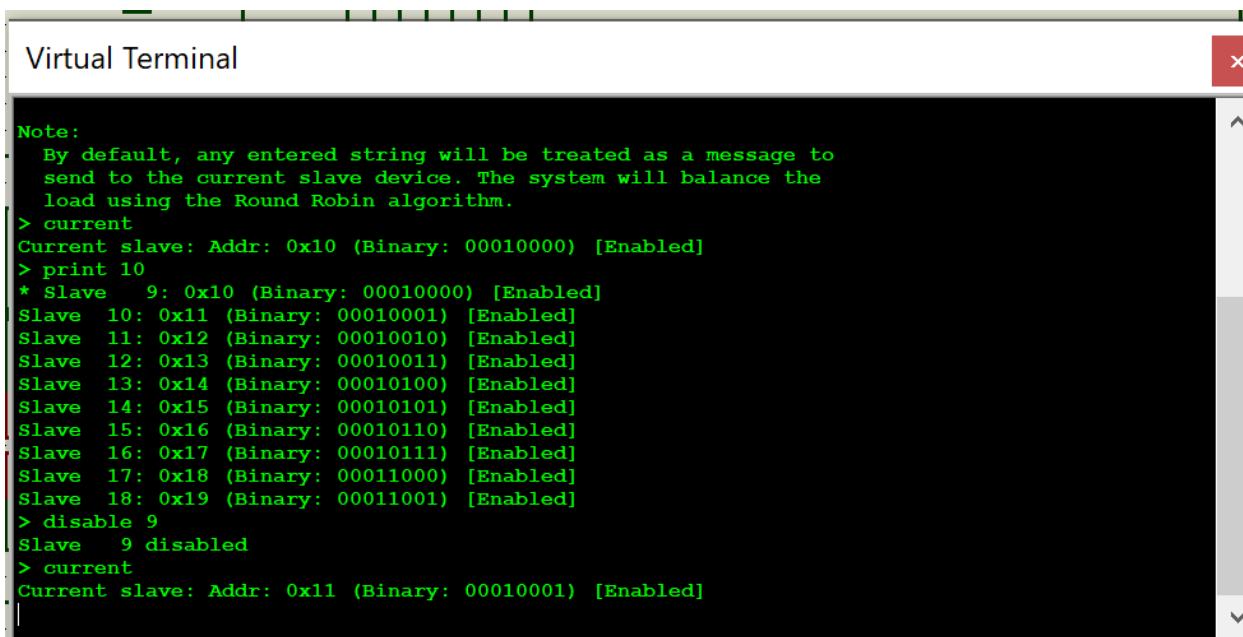


```
Virtual Terminal
* save          - Save current config to EEPROM.
* clear          - Clear EEPROM data and reset configuration.
* reset          - Programmatically reset the microcontroller.
* help           - Show this help message.

Note:
By default, any entered string will be treated as a message to
send to the current slave device. The system will balance the
load using the Round Robin algorithm.
> current
Current slave: Addr: 0x10 (Binary: 00010000) [Enabled]
> print 10
* Slave 9: 0x10 (Binary: 00010000) [Enabled]
Slave 10: 0x11 (Binary: 00010001) [Enabled]
Slave 11: 0x12 (Binary: 00010010) [Enabled]
Slave 12: 0x13 (Binary: 00010011) [Enabled]
Slave 13: 0x14 (Binary: 00010100) [Enabled]
Slave 14: 0x15 (Binary: 00010101) [Enabled]
Slave 15: 0x16 (Binary: 00010110) [Enabled]
Slave 16: 0x17 (Binary: 00010111) [Enabled]
Slave 17: 0x18 (Binary: 00011000) [Enabled]
Slave 18: 0x19 (Binary: 00011001) [Enabled]
```

Рисунок 18 - Ввод команды «print N»

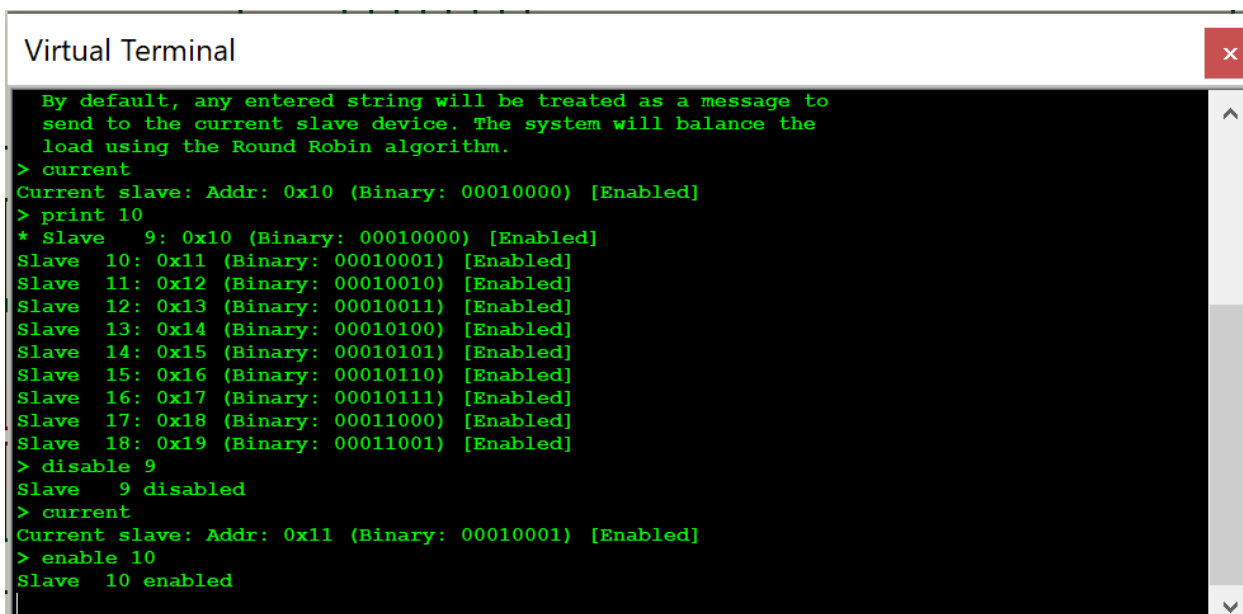
Теперь пользователь ввёл команду «disable N», она отключает слейв с
номером N. Результат ввода показана на рисунке 19.



```
Virtual Terminal
Note:
  By default, any entered string will be treated as a message to
  send to the current slave device. The system will balance the
  load using the Round Robin algorithm.
> current
Current slave: Addr: 0x10 (Binary: 00010000) [Enabled]
> print 10
* Slave 9: 0x10 (Binary: 00010000) [Enabled]
Slave 10: 0x11 (Binary: 00010001) [Enabled]
Slave 11: 0x12 (Binary: 00010010) [Enabled]
Slave 12: 0x13 (Binary: 00010011) [Enabled]
Slave 13: 0x14 (Binary: 00010100) [Enabled]
Slave 14: 0x15 (Binary: 00010101) [Enabled]
Slave 15: 0x16 (Binary: 00010110) [Enabled]
Slave 16: 0x17 (Binary: 00010111) [Enabled]
Slave 17: 0x18 (Binary: 00011000) [Enabled]
Slave 18: 0x19 (Binary: 00011001) [Enabled]
> disable 9
Slave 9 disabled
> current
Current slave: Addr: 0x11 (Binary: 00010001) [Enabled]
```

Рисунок 19 - Ввод команды «disable N»

Теперь пользователь ввёл команду «enable N», она отключает слейв с номером N. Результат ввода показана на рисунке 20.



```
Virtual Terminal
  By default, any entered string will be treated as a message to
  send to the current slave device. The system will balance the
  load using the Round Robin algorithm.
> current
Current slave: Addr: 0x10 (Binary: 00010000) [Enabled]
> print 10
* Slave 9: 0x10 (Binary: 00010000) [Enabled]
Slave 10: 0x11 (Binary: 00010001) [Enabled]
Slave 11: 0x12 (Binary: 00010010) [Enabled]
Slave 12: 0x13 (Binary: 00010011) [Enabled]
Slave 13: 0x14 (Binary: 00010100) [Enabled]
Slave 14: 0x15 (Binary: 00010101) [Enabled]
Slave 15: 0x16 (Binary: 00010110) [Enabled]
Slave 16: 0x17 (Binary: 00010111) [Enabled]
Slave 17: 0x18 (Binary: 00011000) [Enabled]
Slave 18: 0x19 (Binary: 00011001) [Enabled]
> disable 9
Slave 9 disabled
> current
Current slave: Addr: 0x11 (Binary: 00010001) [Enabled]
> enable 10
Slave 10 enabled
```

Рисунок 20 - Ввод команды «enable N»

Теперь пользователь ввёл команду «save», она сохраняет текущую конфигурацию в EEPROM. Результат ввода показана на рисунке 21.

Virtual Terminal

```
load using the Round Robin algorithm.
> current
Current slave: Addr: 0x10 (Binary: 00010000) [Enabled]
> print 10
* Slave 9: 0x10 (Binary: 00010000) [Enabled]
Slave 10: 0x11 (Binary: 00010001) [Enabled]
Slave 11: 0x12 (Binary: 00010010) [Enabled]
Slave 12: 0x13 (Binary: 00010011) [Enabled]
Slave 13: 0x14 (Binary: 00010100) [Enabled]
Slave 14: 0x15 (Binary: 00010101) [Enabled]
Slave 15: 0x16 (Binary: 00010110) [Enabled]
Slave 16: 0x17 (Binary: 00010111) [Enabled]
Slave 17: 0x18 (Binary: 00011000) [Enabled]
Slave 18: 0x19 (Binary: 00011001) [Enabled]
> disable 9
Slave 9 disabled
> current
Current slave: Addr: 0x11 (Binary: 00010001) [Enabled]
> enable 10
Slave 10 enabled
> save
Data saved to EEPROM.
```

Рисунок 21 - Ввод команды «save»

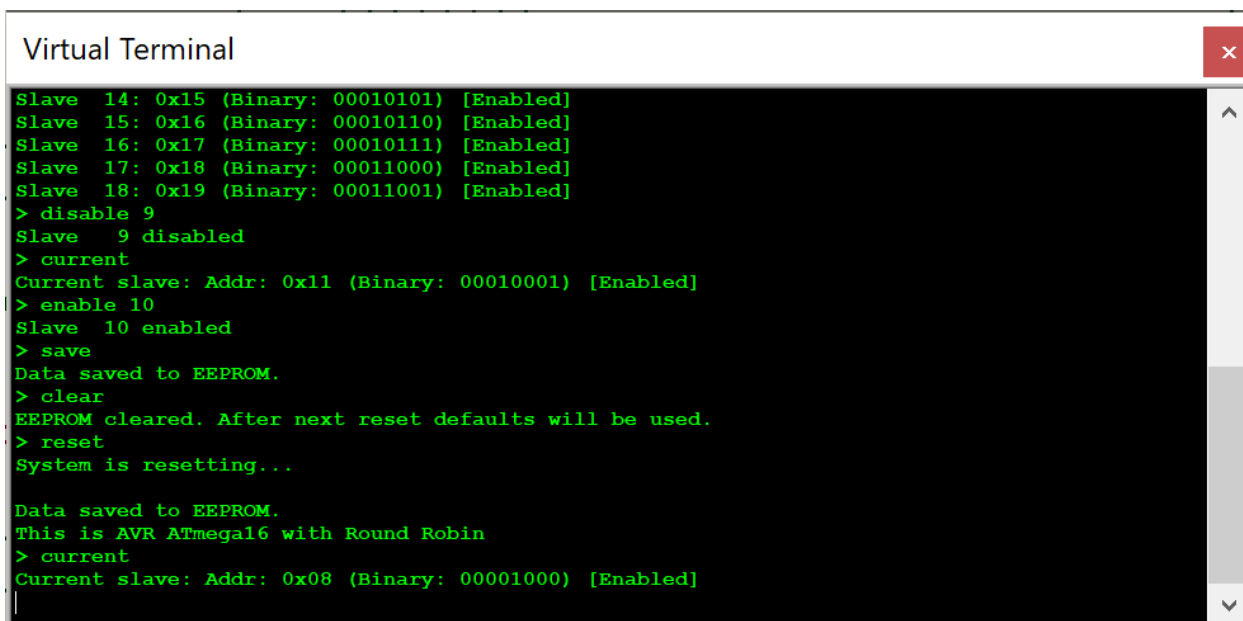
Теперь пользователь ввёл команду «clear», она очищает данные в EEPROM. Результат ввода показана на рисунке 22.

Virtual Terminal

```
Current slave: Addr: 0x10 (Binary: 00010000) [Enabled]
> print 10
* Slave 9: 0x10 (Binary: 00010000) [Enabled]
Slave 10: 0x11 (Binary: 00010001) [Enabled]
Slave 11: 0x12 (Binary: 00010010) [Enabled]
Slave 12: 0x13 (Binary: 00010011) [Enabled]
Slave 13: 0x14 (Binary: 00010100) [Enabled]
Slave 14: 0x15 (Binary: 00010101) [Enabled]
Slave 15: 0x16 (Binary: 00010110) [Enabled]
Slave 16: 0x17 (Binary: 00010111) [Enabled]
Slave 17: 0x18 (Binary: 00011000) [Enabled]
Slave 18: 0x19 (Binary: 00011001) [Enabled]
> disable 9
Slave 9 disabled
> current
Current slave: Addr: 0x11 (Binary: 00010001) [Enabled]
> enable 10
Slave 10 enabled
> save
Data saved to EEPROM.
> clear
EEPROM cleared. After next reset defaults will be used.
```

Рисунок 22 - Ввод команды «clear»

Теперь пользователь ввёл команду «reset», она программно сбрасывает микроконтроллер. Результат ввода показана на рисунке 23.

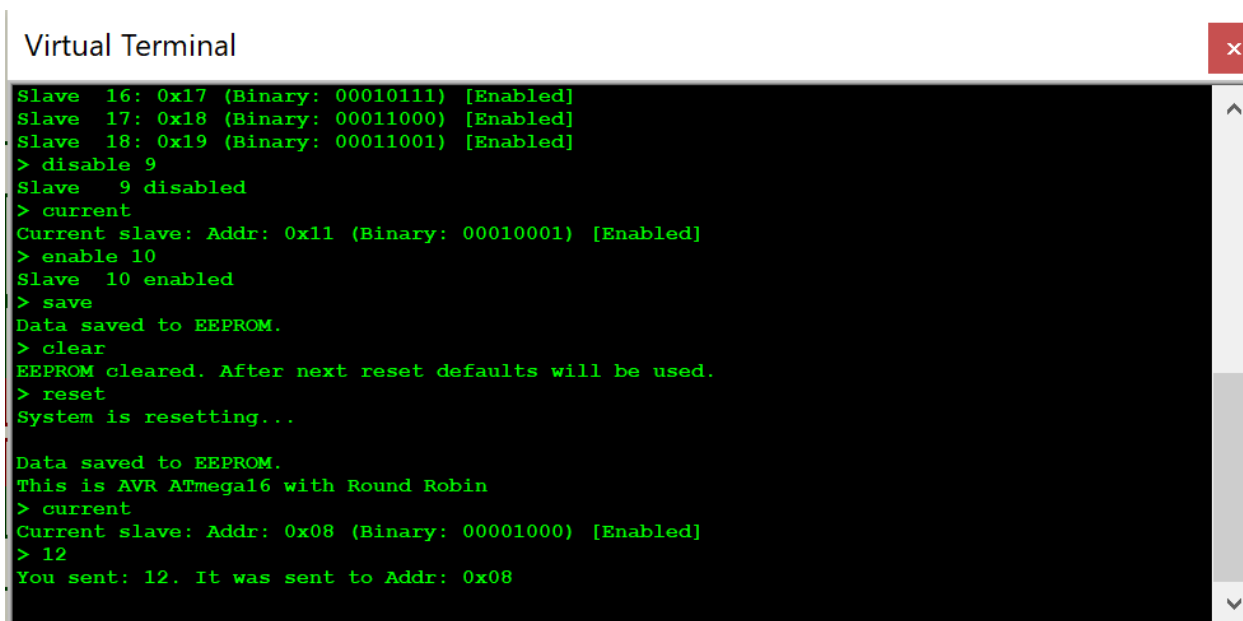


```
Virtual Terminal
Slave 14: 0x15 (Binary: 00010101) [Enabled]
Slave 15: 0x16 (Binary: 00010110) [Enabled]
Slave 16: 0x17 (Binary: 00010111) [Enabled]
Slave 17: 0x18 (Binary: 00011000) [Enabled]
Slave 18: 0x19 (Binary: 00011001) [Enabled]
> disable 9
Slave 9 disabled
> current
Current slave: Addr: 0x11 (Binary: 00010001) [Enabled]
> enable 10
Slave 10 enabled
> save
Data saved to EEPROM.
> clear
EEPROM cleared. After next reset defaults will be used.
> reset
System is resetting...

Data saved to EEPROM.
This is AVR ATmega16 with Round Robin
> current
Current slave: Addr: 0x08 (Binary: 00001000) [Enabled]
```

Рисунок 23 - Ввод команды «reset»

Теперь пользователь ввёл любой другой текст, она будет трактоваться как сообщение для отправки текущему слейву, система будет балансировать нагрузку. Результат ввода показана на рисунке 24. Также, после отправки сообщения слейву, на дисплее показывается отладочная информация – адрес и сообщение слейва, а также эта же информация, но кодированная под I2C. Дисплей показан на рисунке 25, а на рисунке 26 показано отладочная информация из I2C, которую можно сравнить с данными с дисплея.



```
Virtual Terminal
Slave 16: 0x17 (Binary: 00010111) [Enabled]
Slave 17: 0x18 (Binary: 00011000) [Enabled]
Slave 18: 0x19 (Binary: 00011001) [Enabled]
> disable 9
Slave 9 disabled
> current
Current slave: Addr: 0x11 (Binary: 00010001) [Enabled]
> enable 10
Slave 10 enabled
> save
Data saved to EEPROM.
> clear
EEPROM cleared. After next reset defaults will be used.
> reset
System is resetting...

Data saved to EEPROM.
This is AVR ATmega16 with Round Robin
> current
Current slave: Addr: 0x08 (Binary: 00001000) [Enabled]
> 12
You sent: 12. It was sent to Addr: 0x08
```

Рисунок 24 - Ввод команды любой другой текст

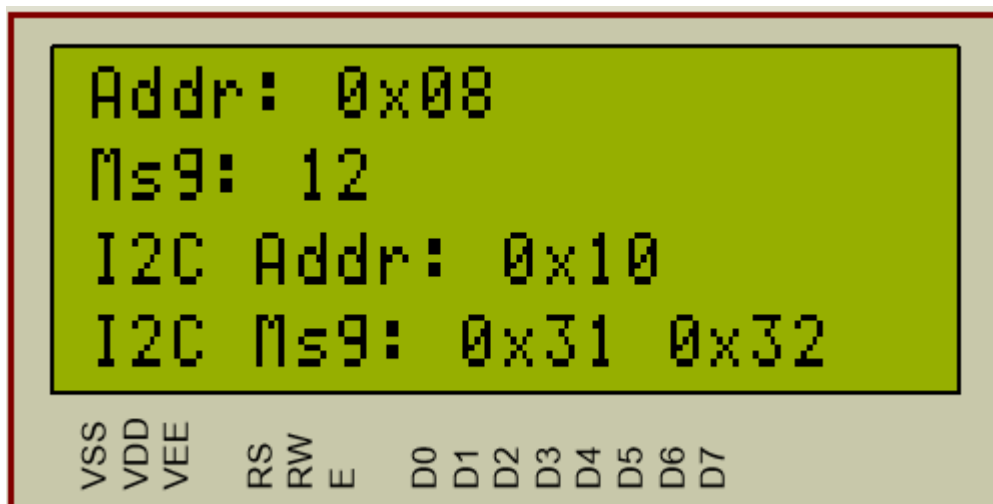


Рисунок 25 - Данные с дисплея

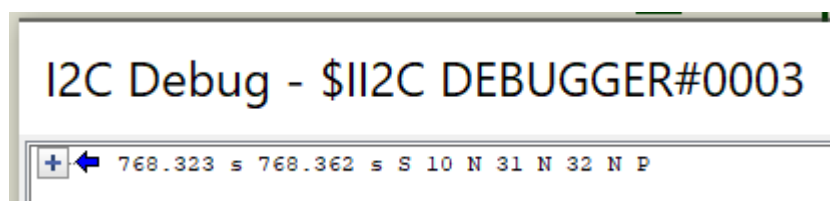


Рисунок 26 - Отладочная информация I2C

Как мы видим, на рисунке 26 есть комбинация трёх чисел – 10, 31, 32, где 10 – адрес слейва, а 31, 32 – передаваемые данные. Эти значения совпадают со значениями с дисплея на рисунке 25, значит, система работает корректно.

2.3 Способы программирования МК

После написания и тестирования программного кода, начинается этап загрузки шестнадцатеричного объектного файла с расширением hex на микроконтроллер. Это может быть выполнено следующими способами [9]:

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

В моем проекте используется метод внутрисхемного программирования (ISP). Для этого задействован интерфейс SPI, обеспечивающий полнодуплексный обмен данными между программатором и

микроконтроллером. Представление процесса программирования микроконтроллера показано на рисунке 27.

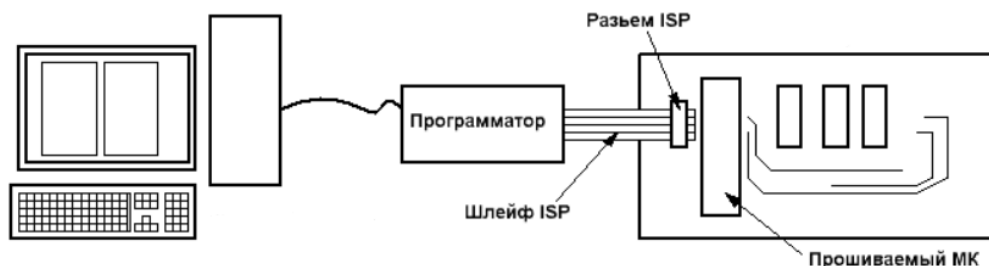


Рисунок 27 - Программирование МК

Прошивка проходит по интерфейсу SPI, в работе программатора задействованы следующие линии:

MISO (Master In Slave Out): Для передачи данных от микроконтроллера к программатору.

- MOSI (Master Out Slave In): Для передачи данных от программатора к микроконтроллеру;

- SCK (Serial Clock): Для синхронизации обмена данными;

- RESET: Сигнал сброса, активирующий режим программирования;

- GND: Общий провод;

- VCC: Питание микроконтроллера, необходимое для работы SPI.

Принципы работы интерфейса SPI:

- интерфейс SPI работает в полнодуплексном режиме, что позволяет одновременно передавать и принимать данные;

- передача данных осуществляется по фронтам тактового сигнала SCK;

- данные передаются через линии MOSI и MISO. Их синхронизация с тактовым сигналом гарантирует точность работы программатора и микроконтроллера.

Описанным образом на МК передается бинарный файл с расширением .hex, полученный в результате компиляции программы.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был создан проект, представляющий собой системы для балансировки нагрузки в распределенной системе. Основой устройства является микроконтроллер серии AVR – ATmega16. Разработка выполнена в соответствии с техническим заданием.

В ходе выполнения работы были созданы электрические функциональная и принципиальная схемы устройства, спецификация на его компоненты и документация. Программное обеспечение, написанное на языке C, успешно протестировано и отлажено с использованием симулятора Proteus 8 Professional. Система обеспечивает прием данных через интерфейс UART, их обработку микроконтроллером, передачу ответа обратно через интерфейс UART, обмен сообщениями с PC по I2C и вывод отладочной информации на дисплей.

Проект продемонстрировал работоспособность разработанного устройства и возможность его применения в образовательных и исследовательских целях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Микроконтроллеры 8051, PIC, AVR и ARM: отличия и особенности [Электронный ресурс]. - URL: http://digitrode.ru/computing-devices/mcu_cpu/1253-mikrokontrollery-8051-pic-avr-i-arm-otlichiya-i-osobennosti.html (дата обращения: 17.10.2024).
2. ATmega16 Datasheet [Электронный ресурс]. - URL: <https://www.rhydolabz.com/documents/Atmega16.pdf> (дата обращения: 17.10.2024)
3. MAX232x Dual EIA-232 Drivers/Receivers Datasheet [Электронный ресурс]. - URL: <https://elsg.ru/pdf/max232.pdf> (дата обращения: 31.10.2024)
4. USART/UART на ATmega16 – Обмен данными по последовательному каналу [Электронный ресурс]. - URL: <https://micro-pi.ru/usart-uart-na-atmega16-обмен-данными/?ysclid=m4ksys612e891165212> (дата обращения: 31.10.2024)
5. Подключение микроконтроллера. Ликбез. | Электроника для всех [Электронный ресурс]. - URL: <http://easyelectronics.ru/podklyuchenie-mikrokontrollera-likbez.html> (дата обращения: 31.10.2024)
6. ГОСТ 2.743-91 ЕСКД ОБОЗНАЧЕНИЯ БУКВЕННО-ЦИФРОВЫЕ В ЭЛЕКТРИЧЕСКИХ СХЕМАХ [Электронный ресурс]. - URL: <https://docs.cntd.ru/document/1200001985> (дата обращения: 07.11.2024)
7. ГОСТ 2.721-74 ЕСКД ОБОЗНАЧЕНИЯ УСЛОВНЫЕ ГРАФИЧЕСКИЕ В СХЕМАХ [Электронный ресурс]. - URL: <https://docs.cntd.ru/document/1200007058> (дата обращения: 07.11.2024)
8. Учебный курс AVR. Использование внешних прерываний в AVR [Электронный ресурс]. - URL: <https://chipenable.ru/index.php/item/105> (дата обращения: 07.11.2024)
9. AVR. Учебный курс. Трактат о программаторах | Электроника для всех [Электронный ресурс]. - URL: <http://easyelectronics.ru/avr-uchebnyj-kurs-traktat-o-programmatorax.html> (дата обращения: 21.11.2024)
10. Хартов В.Я. Микроконтроллеры AVR. Микропроцессорные системы. - 2-е изд., испр. и доп. – М.: Издательство: «Академия», 2014. - 368с.

Приложение А
Листинг программы
На 8 листах

Листинг 1 – Программный код файла main.c

```
#define F_CPU 8000000UL
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1

#include <avr/io.h>
#include <util/delay.h>
#include <util/twi.h>
#include <avr/eeprom.h>    // Для работы с EEPROM
#include <avr/pgmspace.h>   // Для работы с PROGRAM
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/wdt.h>        // Для Watchdog Timer

#define TW_WRITE 0 // Запись по I2C
#define NUM_SLAVES 112

// ===== Глобальные переменные =====
uint8_t slaveAddresses[NUM_SLAVES];
uint8_t slaveEnabled[NUM_SLAVES];
uint8_t currentSlave = 0;

// Сигнатура для проверки данных в EEPROM
const uint8_t eepromSignature[4] = {0xDE, 0xAD, 0xBE, 0xEF};

// Адреса в EEPROM
uint8_t EEMEM eeSignature[4];
uint8_t EEMEM eeSlaveAddresses[NUM_SLAVES];
uint8_t EEMEM eeSlaveEnabled[NUM_SLAVES];
uint8_t EEMEM eeCurrentSlave;

// Строки в PROGRAM для экономии SRAM
const char str_welcome[] PROGRAM = "This is AVR ATmega16 with  
Round Robin";
const char str_data_save[] PROGRAM = "Data saved to EEPROM.";
const char str_data_clear[] PROGRAM = "EEPROM cleared. After next  
reset defaults will be used.";
const char str_reset[] PROGRAM = "System is resetting...";
const char str_err_inv_num[] PROGRAM = "Error: Invalid slave number";
const char str_err_no_active[] PROGRAM = "Error: No active slaves  
available";
const char str_err_current_disable[] PROGRAM = "Error: Current slave is  
disabled";
const char str_help_header[] PROGRAM = "Available commands:";
const char str_help_print1[] PROGRAM = " * print [N] - Display all  
active slave addresses or the next N slaves.";
const char str_help_print2[] PROGRAM = " Without N, displays  
all active addresses.";
const char str_help_enable[] PROGRAM = " * enable N - Enable slave with  
number N.";
const char str_help_disable[] PROGRAM = " * disable N - Disable slave with  
number N.";
const char str_help_current[] PROGRAM = " * current - Show the current  
active slave address.";
const char str_help_save[] PROGRAM = " * save - Save current  
config to EEPROM.";
const char str_help_clear[] PROGRAM = " * clear - Clear EEPROM  
data and reset configuration.";
const char str_help_reset[] PROGRAM = " * reset -  
Programmatically reset the microcontroller.";
```

```

const char str_help_help[] PROGMEM = " * help          - Show this help
message.";
const char str_help_note[] PROGMEM = "Note:";
const char str_help_note2[] PROGMEM = " By default, any entered string
will be treated as a message to";
const char str_help_note3[] PROGMEM = " send to the current slave device.
The system will balance the";
const char str_help_note4[] PROGMEM = " load using the Round Robin
algorithm.";

// ===== UART Функции =====
void USARTInit(unsigned int ubrr) {
    UBRRH = (unsigned char)(ubrr >> 8);
    UBRRL = (unsigned char)(ubrr);
    UCSRB = (1 << RXEN) | (1 << TXEN);
    UCSRC = (1 << URSEL) | (1 << UCSZ0) | (1 << UCSZ1);
}

void USARTTransmitChar(char c) {
    while (!(UCSRA & (1 << UDRE)));
    UDR = c;
}

void USARTTransmitString(const char *str) {
    while (*str) USARTTransmitChar(*str++);
}

void USARTTransmitStringLn(const char *str) {
    USARTTransmitString(str);
    USARTTransmitChar('\r');
    USARTTransmitChar('\n');
}

// Вывод строки из PROGMEM
void USARTTransmitStringP(const char *pstr) {
    char c;
    while ((c = pgm_read_byte(pstr++))) USARTTransmitChar(c);
}

void USARTTransmitStringLnP(const char *pstr) {
    USARTTransmitStringP(pstr);
    USARTTransmitChar('\r');
    USARTTransmitChar('\n');
}

void USARTReceiveString(char *buf, uint8_t n) {
    uint8_t idx = 0;
    char c;
    do {
        while (!(UCSRA & (1 << RXC)));
        c = UDR;
        buf[idx++] = c;
    } while ((idx < n - 1) && (c != '\r'));
    buf[idx - 1] = '\0';
}

// ===== I2C (TWI) =====
#define SCL_CLOCK 100000L
void TWIInit(void) {
    TWSR = 0x00;
    TWBR = ((F_CPU / SCL_CLOCK) - 16) / 2;
    TWCR = (1 << TWEN);
}

```

```

void TWIStart(void) {
    TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));
}

void TWIStop(void) {
    TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
    while (TWCR & (1 << TWSTO));
}

void TWIWrite(uint8_t data) {
    TWDR = data;
    TWCR = (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));
}

// ===== LCD =====
#define LCD_RS PC2
#define LCD_RW PC3
#define LCD_E PC4

#define LCD_DATA_PORT PORTA
#define LCD_DATA_DDR DDRA

void LCD_Command(unsigned char cmd) {
    LCD_DATA_PORT = cmd;
    PORTC &= ~(1 << LCD_RS);
    PORTC |= (1 << LCD_E);
    _delay_us(1);
    PORTC &= ~(1 << LCD_E);
    _delay_ms(2);
}

void LCD_Char(unsigned char data) {
    LCD_DATA_PORT = data;
    PORTC |= (1 << LCD_RS);
    PORTC |= (1 << LCD_E);
    _delay_us(1);
    PORTC &= ~(1 << LCD_E);
    _delay_ms(2);
}

void LCD_Init() {
    LCD_DATA_DDR = 0xFF;
    DDRC |= (1 << LCD_RS) | (1 << LCD_RW) | (1 << LCD_E);
    _delay_ms(20);

    LCD_Command(0x38);
    LCD_Command(0x0C);
    LCD_Command(0x06);
    LCD_Command(0x01);
    _delay_ms(2);
}

void LCD_Clear() {
    LCD_Command(0x01);
    _delay_ms(2);
}

void LCD_SetCursor(unsigned char row, unsigned char col) {
    static const uint8_t pos[4] PROGMEM = {0x80, 0xC0, 0x94, 0xD4};
    uint8_t base = pgm_read_byte(&pos[row - 1]);

```

```

        LCD_Command(base + col - 1);
    }

void LCD_String(char *str) {
    uint8_t i = 0;
    while (str[i]) LCD_Char(str[i++]);
}

// ===== Функции инициализации и EEPROM =====

void initSlaveAddresses() {
    for (uint8_t i = 0; i < NUM_SLAVES; i++) {
        slaveAddresses[i] = 0x08 + i;
        slaveEnabled[i] = 1;
    }

    currentSlave = 0;
}

void toBinary(uint8_t value, char *binaryStr) {
    for (int i = 7; i >= 0; i--) binaryStr[7 - i] = (value & (1 << i)) ? '1'
: '0';
    binaryStr[8] = '\0';
}

// Сохранение данных в EEPROM
void saveDataToEEPROM() {
    // Сохранение сигнатуры
    for (uint8_t i = 0; i < 4; i++)
        eeprom_update_byte(&eeSignature[i], eepromSignature[i]);

    // Сохранение адресов
    for (uint8_t i = 0; i < NUM_SLAVES; i++)
        eeprom_update_byte(&eeSlaveAddresses[i], slaveAddresses[i]);

    // Сохранение статуса
    for (uint8_t i = 0; i < NUM_SLAVES; i++)
        eeprom_update_byte(&eeSlaveEnabled[i], slaveEnabled[i]);

    // Сохранение текущего слейва
    eeprom_update_byte(&eeCurrentSlave, currentSlave);

    USARTTransmitStringLnP(str_data_save);
}

// Загрузка данных из EEPROM
uint8_t loadDataFromEEPROM() {
    // Проверка сигнатуры
    for (uint8_t i = 0; i < 4; i++) {
        uint8_t b = eeprom_read_byte(&eeSignature[i]);
        if (b != eepromSignature[i]) {
            return 0; // Сигнатура не совпала
        }
    }

    // Загрузка адресов
    for (uint8_t i = 0; i < NUM_SLAVES; i++)
        slaveAddresses[i] = eeprom_read_byte(&eeSlaveAddresses[i]);

    // Загрузка статуса
    for (uint8_t i = 0; i < NUM_SLAVES; i++)
        slaveEnabled[i] = eeprom_read_byte(&eeSlaveEnabled[i]);
}

```



```

        // Загрузка текущего слейва
        currentSlave = eeprom_read_byte(&eeCurrentSlave);

        return 1;
    }

    // Инициализация при старте
    void initialLoad() {
        if (!loadDataFromEEPROM()) {
            // Если данные ещё не сохранены, создаём по умолчанию
            initSlaveAddresses();
            // Сохраняем их
            saveDataToEEPROM();
        }
    }

    // ===== Управление слейвами =====

    void enableSlave(uint8_t slaveNumber) {
        if (slaveNumber < 1 || slaveNumber > NUM_SLAVES) {
            USARTTransmitStringLnP(str_err_inv_num);
            return;
        }
        slaveEnabled[slaveNumber - 1] = 1;
        char outBuffer[30];
        sprintf(outBuffer, "Slave %3d enabled", slaveNumber);
        USARTTransmitStringLn(outBuffer);
    }

    void disableSlave(uint8_t slaveNumber) {
        if (slaveNumber < 1 || slaveNumber > NUM_SLAVES) {
            USARTTransmitStringLnP(str_err_inv_num);
            return;
        }
        slaveEnabled[slaveNumber - 1] = 0;
        char outBuffer[30];
        sprintf(outBuffer, "Slave %3d disabled", slaveNumber);
        USARTTransmitStringLn(outBuffer);
    }

    // ===== Функция для программного сброса =====
    void softwareReset() {
        // Включаем Watchdog Timer с минимальным тайм-аутом
        wdt_enable(WDTO_15MS);
        // Бесконечный цикл ожидания сброса
        while(1) {}
    }

    // ===== Функция для отключения Watchdog Timer =====
    void disableWatchdog() {
        // Отключаем Watchdog Timer согласно последовательности из даташита
        WDTCSR |= (1 << WDE) | (1 << WDTOE); // Установить WDE и WDTOE
        WDTCSR = 0x00;                       // Отключить WDT
    }

    int main(void) {
        char buffer[20];
        char outBuffer[50];
        char binaryAddress[9];
        char displayBuffer[21];

        // Отключаем Watchdog Timer при старте, чтобы избежать непреднамеренных
        сбросов

```

```

MCUCSR |= (1 << WDRF); // Очищаем флаг сброса от WDT
disableWatchdog(); // Отключаем Watchdog Timer

USARTInit(MYUBRR);
TWIInit();
LCD_Init();

initialLoad(); // Загружаем данные из EEPROM или создаём новые

USARTTransmitStringLnP(str_welcome);

while (1) {
    USARTReceiveString(buffer, sizeof(buffer));
    sprintf(outBuffer, "> %s", buffer);
    USARTTransmitStringLn(outBuffer);

    if (strncmp(buffer, "print", 5) == 0) {
        uint8_t n = 0;
        if (strlen(buffer) > 6) {
            n = atoi(&buffer[6]);
        }

        uint8_t index = 0;
        if (n > 0 && n < NUM_SLAVES) {
            index = currentSlave;
        }

        uint8_t printed = 0;
        for (uint8_t i = 0; i < NUM_SLAVES && (n == 0 || printed < n);
i++) {
            toBinary(slaveAddresses[index], binaryAddress);
            const char *status = (slaveEnabled[index]) ? "Enabled" :
"Disabled";
            if (index == currentSlave) {
                sprintf(outBuffer, "* Slave %3d: 0x%02X (Binary: %s)
[%s]",
                    index + 1, slaveAddresses[index], binaryAddress,
status);
            }
            else {
                sprintf(outBuffer, "Slave %3d: 0x%02X (Binary: %s) [%s]",
                    index + 1, slaveAddresses[index], binaryAddress,
status);
            }
            USARTTransmitStringLn(outBuffer);
            printed++;
            index = (index + 1) % NUM_SLAVES;
        }
    } else if (strncmp(buffer, "enable ", 7) == 0) {
        uint8_t slaveNumber = atoi(&buffer[7]);
        enableSlave(slaveNumber);
        // Если текущий был отключен, но теперь включен, ничего не
        делать, до след. сообщения.
    } else if (strncmp(buffer, "disable ", 8) == 0) {
        uint8_t slaveNumber = atoi(&buffer[8]);
        disableSlave(slaveNumber);

        if (slaveEnabled[currentSlave] == 0) {
            uint8_t attempts = 0;
            do {
                currentSlave = (currentSlave + 1) % NUM_SLAVES;
                attempts++;
                if (attempts >= NUM_SLAVES) {

```

```

        USARTTransmitStringLnP(str_err_no_active);
        break;
    }
    } while (slaveEnabled[currentSlave] == 0);
}
} else if (strncmp(buffer, "current", 7) == 0) {
    if (slaveEnabled[currentSlave] == 1) {
        toBinary(slaveAddresses[currentSlave], binaryAddress);
        sprintf(outBuffer, "Current slave: Addr: 0x%02X (Binary: %s)
[%s]",
                slaveAddresses[currentSlave], binaryAddress,
"Enabled");
        USARTTransmitStringLn(outBuffer);
    } else {
        USARTTransmitStringLnP(str_err_current_disable);
    }
} else if (strncmp(buffer, "save", 4) == 0) {
    saveDataToEEPROM();
} else if (strncmp(buffer, "clear", 5) == 0) {
    // Очищаем сигнатуру в EEPROM
    for (uint8_t i = 0; i < 4; i++) {
        eeprom_update_byte(&eeSignature[i], 0xFF);
    }

    USARTTransmitStringLnP(str_data_clear);
} else if (strncmp(buffer, "reset", 5) == 0) {
    USARTTransmitStringLnP(str_reset);
    USARTTransmitStringLn("");

    softwareReset();
} else if (strncmp(buffer, "help", 4) == 0) {
    USARTTransmitStringLnP(str_help_header);
    USARTTransmitStringLnP(str_help_print1);
    USARTTransmitStringLnP(str_help_print2);
    USARTTransmitStringLnP(str_help_enable);
    USARTTransmitStringLnP(str_help_disable);
    USARTTransmitStringLnP(str_help_current);
    USARTTransmitStringLnP(str_help_save);
    USARTTransmitStringLnP(str_help_clear);
    USARTTransmitStringLnP(str_help_reset);
    USARTTransmitStringLnP(str_help_help);
    USARTTransmitStringLn("");
    USARTTransmitStringLnP(str_help_note);
    USARTTransmitStringLnP(str_help_note2);
    USARTTransmitStringLnP(str_help_note3);
    USARTTransmitStringLnP(str_help_note4);
} else {
    sprintf(outBuffer, "You sent: %s. It was sent to Addr: 0x%02X",
buffer, slaveAddresses[currentSlave]);
    USARTTransmitStringLn(outBuffer);

    LCD_Clear();
    LCD_SetCursor(1, 1);
    sprintf(displayBuffer, "Addr: 0x%02X",
slaveAddresses[currentSlave]);
    LCD_String(displayBuffer);

    LCD_SetCursor(2, 1);
    sprintf(displayBuffer, "Msg: %s", buffer);
    LCD_String(displayBuffer);

    LCD_SetCursor(3, 1);

```

```

        uint8_t transformedAddress = (slaveAddresses[currentSlave] << 1)
| TW_WRITE;
        sprintf(displayBuffer, "I2C Addr: 0x%02X", transformedAddress);
        LCD_String(displayBuffer);

        TWIStart();
        TWIWrite(transformedAddress);

        LCD_SetCursor(4, 1);
        char i2cMessage[80] = "I2C Msg:";
        uint8_t i = 0;
        while (buffer[i]) {
            TWIWrite(buffer[i]);

            char temp[7];
            sprintf(temp, " 0x%02X", buffer[i]);
            if ((strlen(i2cMessage) + strlen(temp)) < sizeof(i2cMessage))
{
                strcat(i2cMessage, temp);
            } else {
                break;
            }
            i++;
        }

        LCD_String(i2cMessage);

        TWIStop();

        uint8_t attempts = 0;
        do {
            currentSlave = (currentSlave + 1) % NUM_SLAVES;
            attempts++;
            if (attempts >= NUM_SLAVES) {
                USARTTransmitStringLnP(str_err_no_active);
                break;
            }
        } while (slaveEnabled[currentSlave] == 0);

        _delay_ms(1000);
    }
}

return 0;
}

```

Приложение Б
Схема электрическая функциональная
Листов 1

Схема электрическая функциональная

Приложение В
Схема электрическая принципиальная
Листов 1

Схема электрическая принципиальная

Приложение Г
Перечень элементов
Листов 1