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 «Автоматический полив растений»

Студент ИУ6-73Б (Группа) _____ (Подпись, дата) И.В. Ротанков (И.О. Фамилия)

Руководитель _____ (Подпись, дата) И.Б.Трамов (И.О. Фамилия)

2024 г.

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)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ6

_____ А.В. Пролетарский

« 2 » сентября 2024 г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Микропроцессорные системы

Студент группы ИУ6-7 Б

Тема курсовой работы: Автоматический полив растений

Направленность курсовой работы: учебная

Источник тематики (кафедра, предприятие, НИР): кафедра

График выполнения работы: 25% – 4 нед., 50% – 8 нед., 75% – 12 нед., 100% – 16 нед.

Техническое задание:

Разработать МК-систему на базе семейства AVR для автоматического полива растений. Обеспечить пользователю возможность задавать интервал полива с пульта и телефона. Предусмотреть возможность подключения к системе нескольких помп. Система также должна определять утечки воды.

Всю необходимую информацию передавать на телефон.

Разработать схему, алгоритмы и программу. Отладить проект в симуляторе или на макете. Оценить потребляемую мощность. Описать принципы и технологию программирования используемого микроконтроллера.

Оформление курсовой работы:

1. Расчетно-пояснительная записка на 30-35 листах формата А4.
2. Перечень графического материала:
   а) схема электрическая функциональная;
   б) схема электрическая принципиальная.

Дата выдачи задания: «2» сентября 2024 г.

Руководитель курсовой работы	02.09.2024 (Подпись, дата)	И.Б. Трамов (И.О.Фамилия)
Студент	02.09.2024 (Подпись, дата)	_____ (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах; один выдается студенту, второй хранится на кафедре.

РЕФЕРАТ

Расчетно-пояснительная записка состоит из n страниц, включающих в себя 32 рисунка, 11 таблиц, 12 литературных источников и 3 приложения.

Ключевые слова: АВТОМАТИЧЕСКИЙ ПОЛИВ РАСТЕНИЙ, МИКРОКОНТРОЛЛЕР AVR, ОБНАРУЖЕНИЕ УТЕЧЕК ВОДЫ, БЕСПРОВОДНОЕ УПРАВЛЕНИЕ.

Данная работа посвящена разработке и созданию функционального устройства — системы автоматического полива растений на базе микроконтроллера семейства AVR. Система позволяет пользователю задавать интервал и длительность полива с помощью пульта дистанционного управления и мобильного телефона, обеспечивая гибкость и удобство использования.

Разработанная система поддерживает подключение нескольких помп для одновременного полива различных зон или растений. В целях безопасности и предотвращения аварийных ситуаций в устройство интегрированы датчики обнаружения утечек воды, которые при срабатывании немедленно останавливают полив и уведомляют пользователя.

В процессе разработки были созданы алгоритмы функционирования системы, написаны соответствующие программные коды на языке C для микроконтроллера AVR. Также были разработаны функциональная и принципиальная схемы устройства с использованием программного обеспечения Proteus, в котором проведена симуляция и отладка работы системы.

В результате работы была создана система автоматического полива растений, обеспечивающая удаленное управление и мониторинг через мобильный телефон или пульт. Проведена оценка потребляемой мощности системы, а также подробно описаны принципы и технология программирования использованного микроконтроллера.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

CISC – Complex Instruction Set Computer, тип процессорной архитектуры.

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

Flash – разновидность программируемой памяти.

LCD / ЖКД – Liquid Crystal Display / жидкокристаллический дисплей.

MIPS – производительность, миллион инструкций в секунду.

ШИМ – широтно-импульсный модулятор.

RFID – Radio Frequency Identification, радиочастотная идентификация.

RISC – Reduced Instruction Set Computer, тип процессорной архитектуры.

SPI – Serial Peripheral Interface, последовательный периферийный интерфейс.

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

UART – Universal Asynchronous Receiver/Transmitter, универсальный асинхронный приемопередатчик.

USART – Universal Synchronous/Asynchronous Receiver/Transmitter, универсальный синхронный/асинхронный приемопередатчик.

ПО – программное обеспечение.

ПК – персональный компьютер.

ПЭВМ – персональная электровычислительная машина.

ВВЕДЕНИЕ

В данной курсовой работе осуществляется разработка системы автоматического полива растений на базе микроконтроллера ATmega8515. Цель проекта – создать автоматизированную систему полива с возможностью удаленного управления, применимую как в домашних условиях, так и в тепличных хозяйствах.

Современное растениеводство движется в сторону автоматизации процессов ухода за растениями, что позволяет оптимизировать расход воды и человеческих ресурсов. В этом контексте, использование микроконтроллера ATmega8515 для создания системы автоматического полива представляет собой технически обоснованное решение.

Управление системой полива осуществляется по заданному пользователем расписанию, с возможностью ручного управления через пульт или телефон. Система включает возможность подключения нескольких помп для организации независимых зон полива, а также функцию определения утечек воды с помощью расходометров.

Особое внимание в работе уделено разработке пользовательского интерфейса. Реализовано отображение данных на ЖК-дисплее устройства и передача информации на компьютер или смартфон через последовательный интерфейс. Система включает функцию оповещения о нештатных ситуациях, таких как утечки воды. Таким образом, данная разработка представляет собой решение для автоматизации процесса полива растений, обеспечивающее выполнение всех поставленных задач.

1 КОНСТРУКТОРСКАЯ ЧАСТЬ

В конструкторской части изложены этапы создания аппаратной части курсовой работы.

1.1 Анализ требований и принцип работы системы

Исходя из требований, изложенных в техническом задании, необходимо разработать устройство – автоматической системы полива на базе микроконтроллеров семейства AVR. В качестве целевого устройства было принято решение использовать Atmega8515. Более подробно процесс выбора будет описан в пункте 1.1.1.

С помощью входящего в состав микроконтроллера таймера общего назначения реализована работа отсчета времени, которая отображается на ЖКД. Система поддерживает настройку глобального времени, которое пользователь может установить через кнопки управления. Настройка времени выполняется кнопкой Time, а изменение часов и минут производится с шагом в 5 минут кнопками V+ и V-. Время синхронизировано с управляющим модулем и используется для определения времени запуска полива на заданных зонах.

Система обеспечивает управление параметрами полива для каждой зоны, включая:

- время старта полива;
- порог влажности;
- продолжительность полива.

С помощью кнопок Zone, Param, V+ и V- пользователь может изменять параметры каждой зоны. ЖКД отображает текущую зону в формате Zone N hh:mm/hum, где N – номер зоны, hh:mm – время старта полива, а hum – текущий уровень влажности.

В системе реализован контроль утечек с помощью расходомеров, подключенных к каждой зоне. В случае, если расход воды выходит за допустимые пределы, система активирует звуковую сигнализацию и зажигает

индикатор ошибки. Звуковой сигнал отключается автоматически при восстановлении нормальной работы.

В системе также предусмотрена возможность передачи информации через интерфейс UART.

Для контроля влажности используются подключенные к системе датчики DHT-22. Если влажность в зоне превышает заданный порог, полив автоматически прекращается.

В случае возникновения ошибок работы системы (например, отсутствие сигнала от датчика влажности или критические утечки воды), система подает звуковой сигнал, зажигает индикатор ошибки и выводит информацию об этом на ЖКД.

1.1.1 Выбор элементной базы

Для выполнения поставленной задачи было решено использовать микроконтроллер ATmega8515 от компании Atmel (теперь часть компании – Microchip Technology).

Данный микроконтроллер обладает широкими возможностями по вводу и выводу данных. Для этих целей ATmega8515 имеет четыре параллельных восьмиразрядных порта и один трехразрядный порт. 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 35…120 кОм [4].

Данное устройство имеет контроллер прерываний, который позволяет быстро и удобно обрабатывать как внешние (например, нажатие кнопки), так и внутренние прерывания (например, от внутреннего таймера). Также ATmega8515 поддерживает специальный канал UART, который необходим для связи с компьютером и со смартфоном.

Таким образом, делаем вывод, что выбранный микроконтроллер обладает всем необходимым функционалом для реализации проекта. Более подробное описание и характеристики используемого микроконтроллера представлены в разделе 1.2.1.

1.1.2 Внешние компоненты

С помощью входящего в состав микроконтроллера таймера общего назначения реализована работа отсчета времени, которая отображается на используемом ЖК-дисплее. Для установки и настройки глобального времени пользователь использует пульт оператора, состоящий из шести кнопок: отображение и настройка текущего времени, отображение полной информации о конкретной зоне, переключение между параметрами конкретной зоны, увеличение или уменьшение значения выбранного параметра. Таким образом, пульт оператора представляет собой набор из шестии кнопок, позволяющих гибко настраивать время работы системы.

ЖК-дисплей используется для отображения необходимой информации: текущее время, информация о времени запуска полива и уровне влажности в каждой зоне, уведомления об успешной настройке параметров полива и предупреждения о критических ситуациях, таких как утечки воды. В случае обнаружения утечки воды система воспроизводит звуковой сигнал через подключенный звуковой сигнализатор и зажигает соответствующий индикаторный светодиод, сигнализируя о наличии проблемы.

Для связи с внешними устройствами используется UART-интерфейс, реализованный через последовательный порт RS-232. Для преобразования сигналов TTL (используемых микроконтроллером) в сигналы RS-232 применяется драйвер MAX232, обеспечивающий надежную передачу данных между системой полива и внешними устройствами, такими как компьютер или смартфон. Дополнительно для беспроводного подключения смартфона к устройству используется Bluetooth-модуль HC-05, позволяющий принимать команды от пользователя и передавать данные о состоянии системы без проводов.

В системе реализован контроль уровня влажности почвы с помощью датчиков влажности DHT-22. Если уровень влажности в зоне превышает заданный порог, система автоматически прекращает полив, предотвращая переувлажнение.

1.1.3 Проектирование структурной схемы

Учитывая вышесказанное, имеем следующие устройства/внешние компоненты, необходимые для реализации целевого устройства – регистратора времени:

- микроконтроллер ATmega8515;
- пульт оператора, состоящий из шести кнопок;
- пьезоизлучатель, для воспроизведения звукового сигнала в случае обнаружения утечки в одной из зон;
- жидкокристаллический дисплей LM016L для отображения необходимой информации;
- датчики влажности DHT-22;
- расходометры YF-S201;
- разъем RS-232 и драйвер MAX232, необходимые для связи с внешними ПЭВМ;
- модуль HC-05, необходимый для управления по Bluetooth.

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 необходимые компоненты, которые необходимы для работы итогового устройства, однако, этого достаточно для проектирования структурной схемы. Обобщенная структурная схема представлена на рисунке 1.

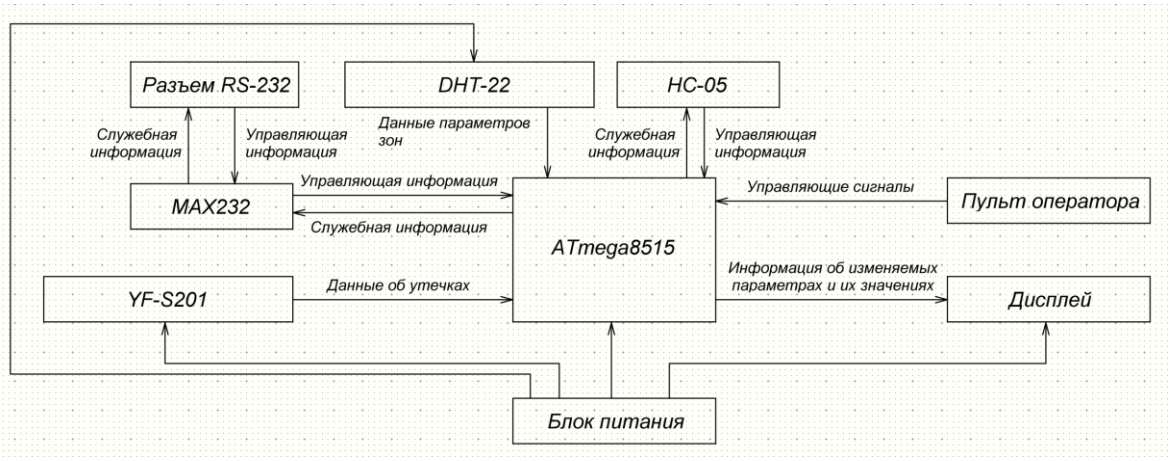

Рисунок 1 – Структурная схема разрабатываемого устройства

Представленная структурная схема, показывает общий принцип работы итогового устройства.

1.2 Проектирование функциональной схемы

В данном разделе приведено функциональное описание работы разрабатываемой системы и проектирование функциональной схемы устройства.

1.2.1 Микроконтроллер ATmega8515

В данном разделе приведено подробное рассмотрение характеристик и архитектуры выбранного микроконтроллера, а также его компонентов, которые будут использованы при создании целевого устройства.

1.2.1.1 Общие сведения

Общие характеристики ATmega8515 представлены в таблице 1 [5].

Таблица 1 – Общие характеристики ATmega8515

Название характеристики		Значение
Ядро		8-разрядное RISC-ядро со встроенной поддержкой AVR архитектуры
Тактовая частота		До 16 МГц
Средняя производительность		До 16 MIPS
Размер памяти	Flash	8 Кбайт
	SRAM	512 Байт (с возможностью расширения до 64 Кбайт)
	EEPROM	512 Байт
Линии ввода/вывода		35

Ядро микроконтроллера ATmega8515 имеет RISC-архитектуру, которая использует набор простых инструкций, способных выполняться в течение одного тактового цикла. Это контрастирует с CISC-архитектурой, где используется более сложный набор инструкций. Основные особенности RISC:

- более быстрая и эффективная обработка данных за счет уменьшения количества и сложности инструкций;
- большинство инструкций выполняются за один такт, что ускоряет общую обработку;
- более простые инструкции позволяют эффективнее использовать ограниченное пространство памяти.

Данное ядро имеет RISC-архитектуру в сочетании с гарвардской архитектурой, которая отделяет физические пути и хранилища для инструкций и данных. Это позволяет одновременно читать инструкцию и обращаться к данным, увеличивая производительность. На рисунке 2 представлена обобщенная структурная схема такой архитектуры [2].

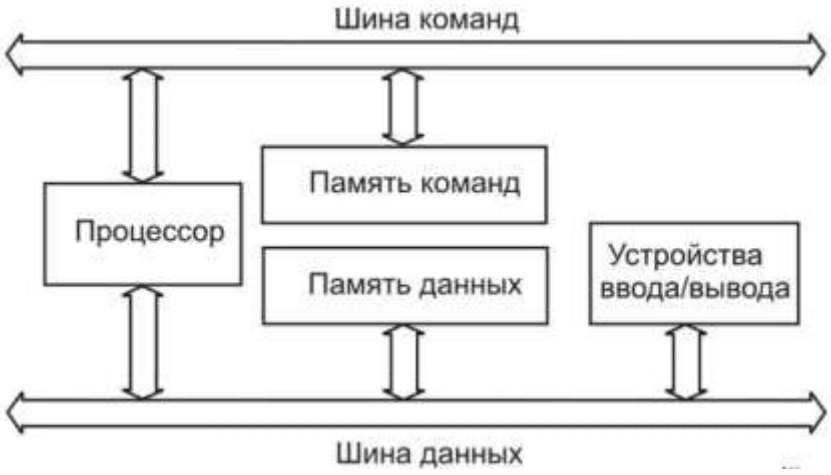

Рисунок 2 – Гарвардская архитектура

Функциональные возможности микроконтроллера ATmega8515:

- поддержка интерфейса USART;
- поддержка интерфейса SPI;
- 3 внешних источника прерываний;
- 32 регистра общего назначения;
- 8-битный таймер/счетчик;
- 16-битный таймер/счетчик;
- ШИМ на основе 16-битного таймера;
- наличие аналогового компаратора.

Архитектура ATmega8515 представлена на рисунке 3.

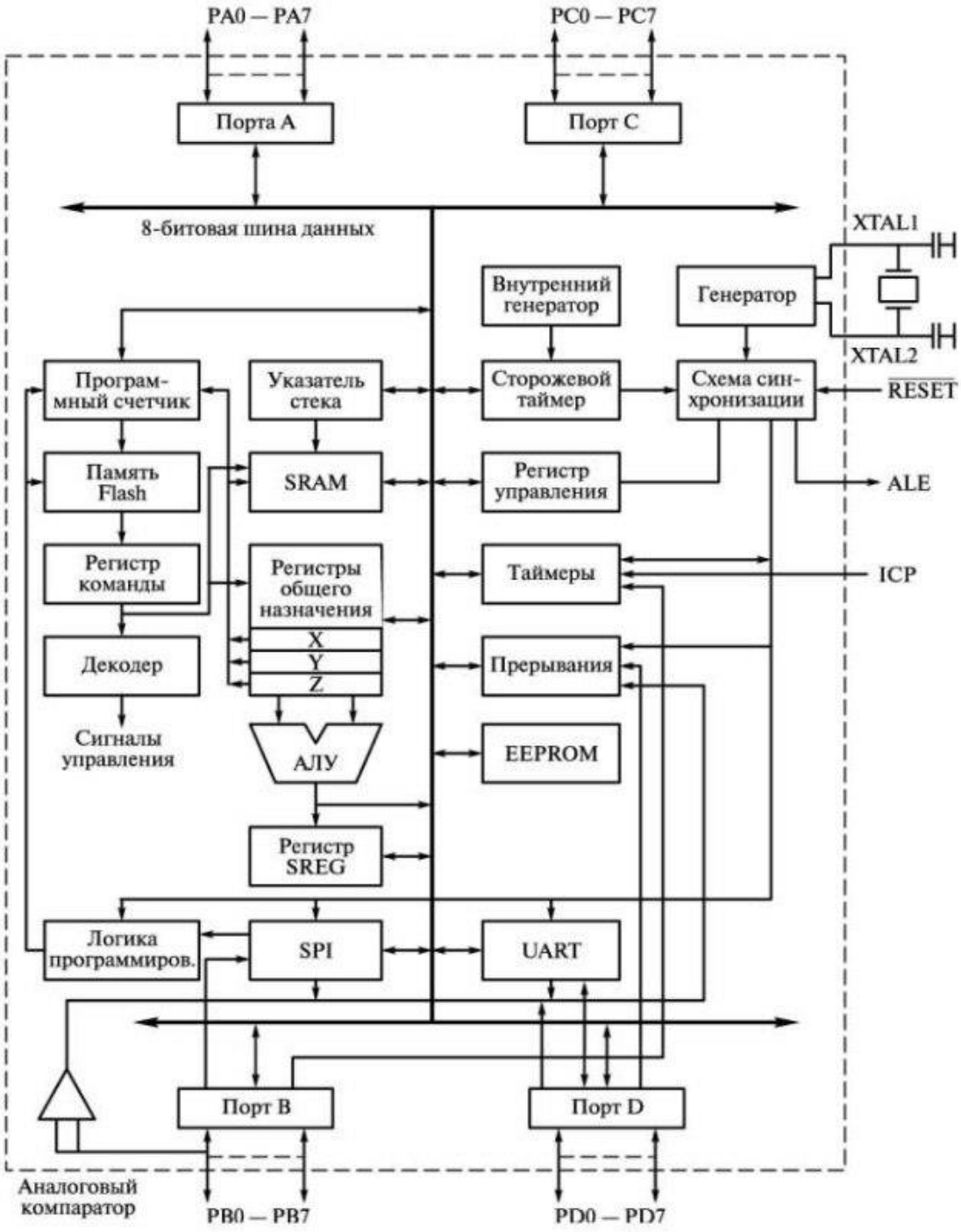

Рисунок 3 – Архитектура ATmega8515

Возможности данного микроконтроллера полностью покрывают требования для реализации итогового устройства:

- большое количество линий ввода-вывода позволяют подключить все необходимые компоненты;
- возможность использования внешних источников прерываний для быстрой и удобной обработки нажатия кнопок;

– 16-битный таймер для реализации отсчета времени;
– достаточный объем Flash-памяти для хранения программ и констант;
– UART для связи с внешними ПЭВМ.

1.2.1.2 Распределение адресного пространства

Карта распределения памяти ATmega8515 представлена на рисунке 4.

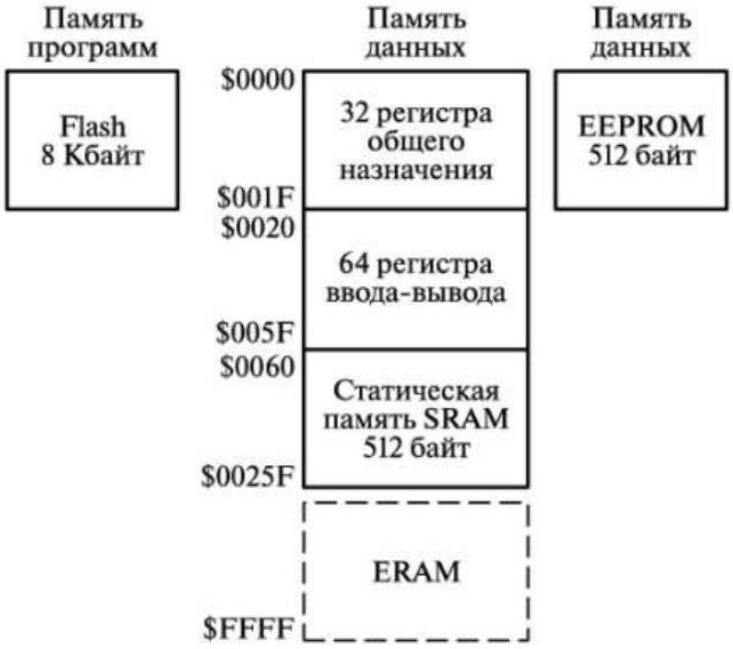

Рисунок 4 – Организация памяти ATmega8515

Память программ – Flash-память используется для хранения исполняемого кода программы. Данная память имеет 16-битную организацию и применяется одноуровневая конвейеризация – во время выполнения инструкции следующая за ней инструкция выгружаются заранее, благодаря чему инструкции могут выполняться каждый машинный цикл. Важной особенностью является то, что она сохраняет содержимое даже при отсутствии питания.

Регистры общего назначения и ввода-вывода используются для управления операциями и периферийными устройствами. Регистры общего назначения важны для выполнения программ и обработки данных, в то время как регистры ввода-вывода используются для контроля и управления внешними устройствами и интерфейсами.

Статическая память SRAM предназначена для временного хранения переменных и промежуточных данных во время выполнения программы. Она быстродействующая и доступна для чтения и записи во время работы микроконтроллера.

Энергонезависимая память EEPROM используется для хранения данных, которые должны сохраняться при выключении питания, например, настройки устройства или значения, которые не должны теряться при отключении.

Диапазон адресов под внешнюю память ERAM предусмотрен для возможности подключения внешней памяти, которая может быть использована для расширения объема доступной памяти или для хранения больших данных, которые не умещаются во внутренней памяти микроконтроллера. В стандартной комплектации ATmega8515 внешняя память не используется, но может быть добавлена при необходимости.

Для реализации целевого устройства будет достаточно встроенной памяти микроконтроллера, поэтому внешняя память не понадобится. Также, основываясь на техническом задании, можно сделать вывод, что не требуется сохранять данные о времени регистрации сотрудников таким образом, чтобы при перезагрузке устройства они оставались, поэтому память EEPROM так же не понадобится.

1.2.2 Внешняя периферия

Для реализации системы автоматического полива недостаточно одного микроконтроллера. Для взаимодействия с ним необходим ряд внешней периферии: кнопки, дисплей, датчики влажности DHT-22, расходометры, разъем RS-232, Bluetooth-модуль HC-05. В этом разделе будут подробно рассмотрены внешние компоненты, используемые для работы устройства.

1.2.2.1 LCD-дисплей

В техническом задании указана конкретная модель необходимого LCD-дисплея – LM016L. Это устройство – стандартный 16x2 символьный ЖКД, который может отображать до 32 символов (16 символов в двух строках). На

рисунке 5 представлена обобщенная структурная схема данного модуля, а на рисунке 6 – внешний вид LM016L и его распиновка.

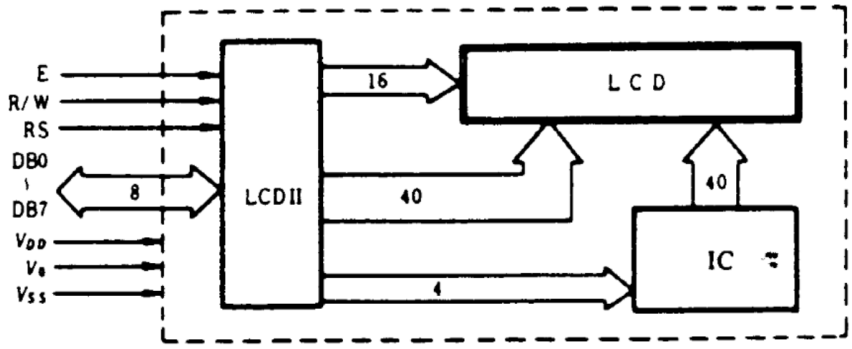

Рисунок 5 – Структурная схема LM016L

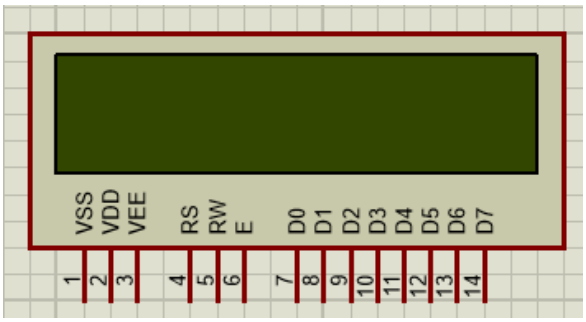

Рисунок 6 – Внешний вид и распиновка LM016L

Рекомендованное напряжение питание данного модуля – 5 В. При таком напряжении устройство сможет использовать свою максимальную яркость. 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 равен 0, то выбирается регистр инструкций (для записи команд в модуль), если RS равен 1, то выбирается регистр данных (для записи или чтения данных, которые будут отображаться);
– RW – Read/Write, определяет режим чтения или записи: если RW равен 0, то производится запись в модуль, если RW равен 1, то 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 этот вывод не используется, так как мы используем максимальную яркость, которая и устанавливается по умолчанию.

1.2.2.2 Датчик влажности и температуры DHT-22

Датчик DHT-22 (также известный как AM2302) представляет собой компактный и эффективный модуль для измерения относительной влажности и температуры. Основой его работы является емкостной датчик влажности и термистор, которые связаны с высокоточными электронными схемами. DHT-22 обеспечивает цифровой выход, что минимизирует сложность интеграции и повышает надежность передачи данных. Внешний вид и описание пинов DHT-22 представлено на рисунке 7.

Одной из причин выбора DHT-22 для моего проекта является его высокая точность измерения влажности, которая составляет ±2 %RH, и устойчивость к долгосрочным изменениям, что делает его подходящим для задач мониторинга микроклиматических условий. Несмотря на наличие возможности измерения температуры, эти данные в данном проекте не используются. Это объясняется спецификой разрабатываемой системы, где мониторинг температуры на текущем этапе не является критическим. Однако выбор DHT-22 обусловлен его универсальностью, поскольку возможность получения данных о температуре позволяет в будущем модернизировать

систему, добавив управление на основе температурных показателей, если это станет необходимо.

Дополнительное преимущество DHT-22 заключается в его малых габаритах, что облегчает его интеграцию в компактные устройства, и низком энергопотреблении, что важно для автономных систем. Цифровая передача данных с использованием одноразрядного протокола (single-wire communication) также обеспечивает надежность связи на больших расстояниях, до 20 метров, при использовании качественного экранированного кабеля.

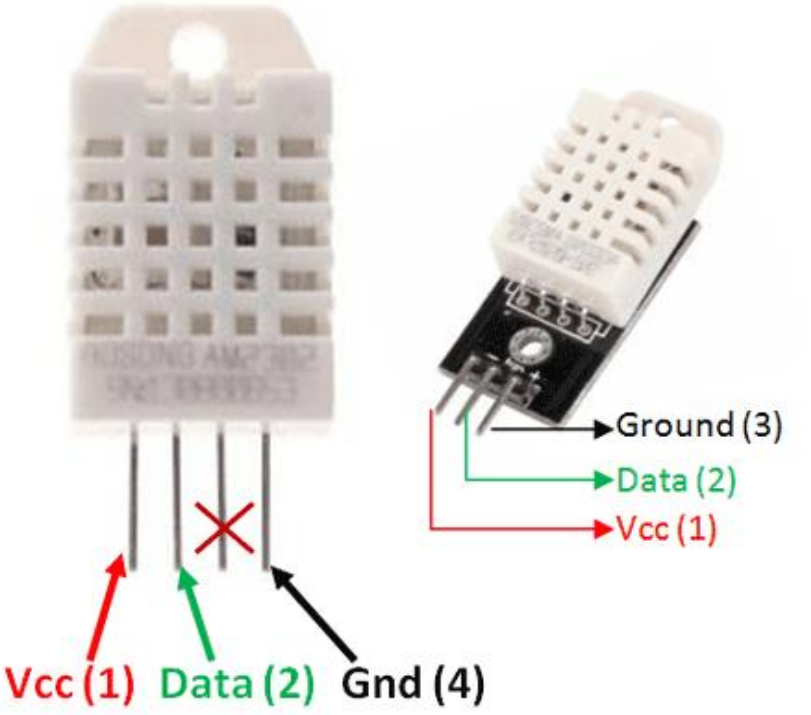

Рисунок 7 — Датчик температуры и влажности DHT-22

Обмен данными с DHT-22 осуществляется через цифровую шину с использованием простого одноразрядного протокола (single-wire communication). Протокол взаимодействия включает три основных этапа: отправку стартового сигнала, получение сигнала-отклика и передачу данных от датчика к микроконтроллеру. Рассмотрим назначение каждого пина данного модуля:

– Отправка стартового сигнала — микроконтроллер переводит линию данных в низкий уровень на период не менее 1 мс, это служит сигналом для датчика о начале передачи, после этого линия переводится в состояние высокого уровня на 20–40 мкс, чтобы датчик подготовился к отправке данных;
– Ответный сигнал от датчика — DHT-22 отправляет ответный сигнал, который начинается с низкого уровня длительностью 80 мкс, за которым следует сигнал высокого уровня той же длительности, это подтверждает готовность датчика к передаче данных;
– Передача данных — данные передаются в виде 40-битной последовательности, включающей относительную влажность, температуру и контрольную сумму.

Каждый бит данных начинается с низкого уровня длительностью 50 мкс, за которым следует высокий уровень. Длительность высокого уровня определяет значение бита: 26–28 мкс обозначают "0", а 70 мкс — "1". Микроконтроллер считывает данные, анализируя длительность высокого уровня.

Для проверки корректности передачи вычисляется контрольная сумма: суммируются все байты данных о влажности и температуре, а результат сравнивается с контрольным байтом.

1.2.2.3 Расходометр YF-S201

При рассмотрении данного компонента важно учитывать, что YF-S201 представляет собой цифровой расходомер, работающий на основе эффекта Холла. Основной принцип работы этого устройства заключается в регистрации количества жидкости, проходящей через встроенный турбинный механизм. Турбина, расположенная внутри датчика, вращается под воздействием потока воды. При каждом обороте турбины срабатывает встроенный датчик Холла, создавая электрический импульс. Эта конструкция обеспечивает изоляцию сенсора от воды, что предотвращает его повреждение в процессе эксплуатации.

Абстрактная схема YF-S201 отображена на рисунке 7 [9]. Датчик имеет три выхода: питание (+5 В или +15 В), заземление (GND) и цифровой сигнал, который поступает на микроконтроллер. Импульсный выход с датчика Холла характеризуется частотой, прямо пропорциональной скорости потока. Для YF-S201 используется коэффициент преобразования: частота (Гц) = 7.5 × расход (л/мин). Таким образом, один импульс соответствует объёму в 1/450 литра, что позволяет достичь высокой точности измерений.

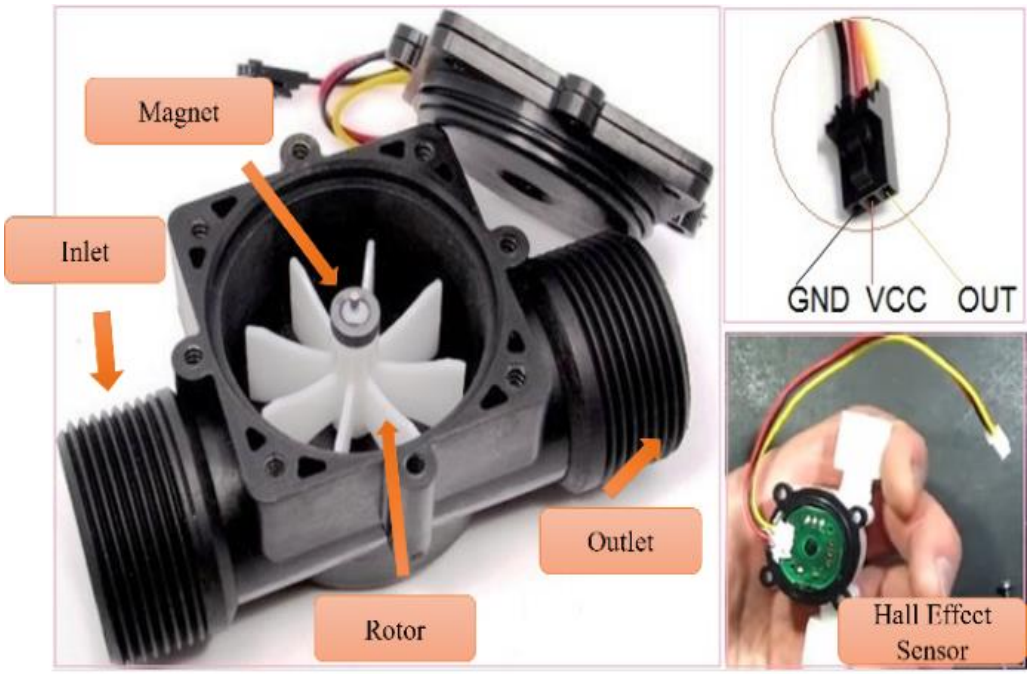

Рисунок 8 – Внешний вид и компоненты YF-S201.

Установка расходомера проста благодаря стандартному соединению с трубопроводом диаметром 1/2 дюйма. Для подключения используются три провода: красный для подачи напряжения, чёрный для заземления и жёлтый для передачи сигнала. Рабочий диапазон датчика охватывает расход от 1 до 30 литров в минуту, что делает его универсальным решением для систем полива, контроля расхода воды и других задач. Сама система будет ограничена значением в 20 литров в минуту для каждой зоны.

Одной из ключевых особенностей YF-S201 является низкое энергопотребление, что позволяет питать его непосредственно от микроконтроллера. В условиях отсутствия воды в системе выход датчика находится в состоянии логической единицы, что исключает ложные

срабатывания. Этот механизм делает устройство устойчивым к внешним факторам, таким как вибрация или переменный поток.

1.2.2.4 Кнопки

Данный компонент прост и понятен в использовании, поэтому нет смысла рассматривать работу самих кнопок, ведь по сути их основная задача – замыкание цепи. Однако стоит рассмотреть то, каким образом эти кнопки будут подключаться к микроконтроллеру.

В первую очередь стоит вспомнить, что микроконтроллер обладает встроенными подтягивающими резисторами. Таким образом, для того чтобы микроконтроллер обрабатывал нажатие кнопок, необходимо один выход кнопки подключить к земле, а второй – к выходу микроконтроллера, который «назначен» для конкретной кнопки. Таким образом, схема подключения, представленная на рисунке 9 максимально проста и ясна.

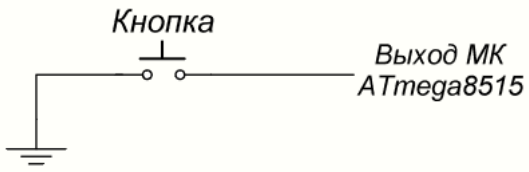

Рисунок 9 – Обобщенная схема подключения кнопки к микроконтроллеру с подтягивающим резистором

Для быстрой обработки нажатия на кнопку ее можно подключить к выходу, отвечающему за внешнее прерывание. Однако, у нас таких кнопок шесть, а выходов, отвечающих за внешнее прерывание, только три. В связи с этим схема подключения кнопок к микроконтроллеру необходимо немного усложнить.

Для того, чтобы иметь возможность получать сигнал на прерывание от всех шести кнопок, необходимо подключить кнопки к назначенным для них выходам микроконтроллера, а также каждую кнопку подключить к одному и тому же выходу, отвечающему за внешнее прерывание (например, выход, отвечающий за прерывание INT0) с помощью диодной сборки. Таким образом, нажатие любой из всех шести кнопок приведет к прерыванию, в обработчике

которого можно будет определить, какая именно кнопка была нажата путем «опроса» состояний назначенных на эти кнопки выходов. Обобщенная схема подключения одной кнопки к микроконтроллеру представлена на рисунке 10.

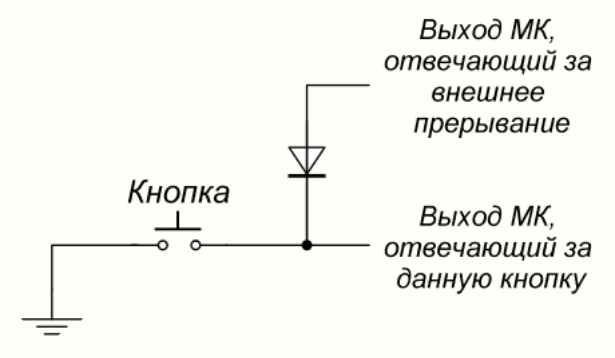

Рисунок 10 – Диодная сборка, обеспечивающая запрос прерывания от большого количества кнопок

1.2.2.5 Интерфейс RS-232 и модуль MAX232

Передача данных с микроконтроллера на внешние устройства происходят через разъем (интерфейс) RS-232.

RS-232 – стандарт физического уровня для синхронного и асинхронного интерфейса (USART и UART). Он обеспечивает передачу данных и некоторых специальных сигналов между терминалом и устройством приема. Информация передается по проводам с уровнями сигналов, отличающимися от стандартных 5 В, для обеспечения большей устойчивости к помехам. Асинхронная передача данных осуществляется с установленной скоростью при синхронизации уровнем сигнала стартового импульса. [9]

Интерфейс RS-232 предназначен для подключения аппаратуры, передающей или принимающей данные (АПД), к аппаратуре каналов данных (АКД). В роли АПД может выступать компьютер, принтер и другое периферийное оборудование. На рисунке 11 приведена стандартная схема подключения устройств. [10]

Рисунок 11 – Схема подключения по RS-232

В основном выделяют два вида разъемов RS-232: DB-9 и DB-25, с 9 и 25 контактами соответственно. В данной работе используется 9-контактный разъем. На рисунке 12 представлен внешний вид порта RS-232 и назначение каждого контакта разъема, которое также описано в таблице 2.

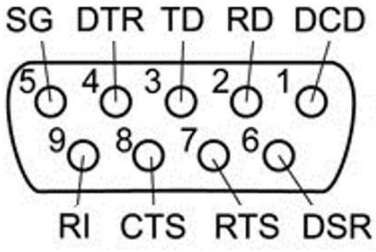

Рисунок 12 – Внешний вид и назначение контактов разъема RS-232

Таблица 2 – Назначение контактов разъема RS-232

Номер	Обозначение	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

Продолжение таблицы 2

8	CTS	Clear To Send – готовность приема
9	RI	Ring Indicator – «входящий вызов» на модеме

При разработке целевого устройства устанавливается подключение только к контактам RXD и TXD, чего достаточно для выполнения поставленной задачи.

Стандартная посылка одного сообщения по каналу UART занимает 10 бит. В режиме простоя, когда по линии ничего не передается, она находится в состоянии логической единицы. Начало передачи обозначают передачей стартового бита, который всегда равен нулю. Затем идет передача восьми бит

данных. Завершает посылку бит четности и стоповый бит. Бит четности осуществляет проверку переданных данных. Стоповый бит говорит нам, что пересылка данных завершена. Стоповый бит, как и стартовый, равен нулю. На рисунке 13 представлен формат асинхронной передачи данных по UART.

Рисунок 13 – Формат асинхронной передачи сообщения по UART

В большинстве случаев именно в АКД установлен модуль MAX232, который необходим для обеспечения совместимости между логическими уровнями RS-232 и низковольтными уровнями, используемыми в цифровых цепях. Этот модуль инвертирует сигнал, получаемый по UART. Так, например, АПД «А» хочет отправить сообщение на АПД «Б», АКД «А» получает сигнал, инвертирует его и отправляет по линии связи между АКД, АКД «Б» получает этот сигнал, инвертирует его обратно и АПД «Б» получает правильное сообщение, которое он может прочитать.

На рисунке 14 представлена распиновка модуля MAX232, а также обозначение каждого выхода.

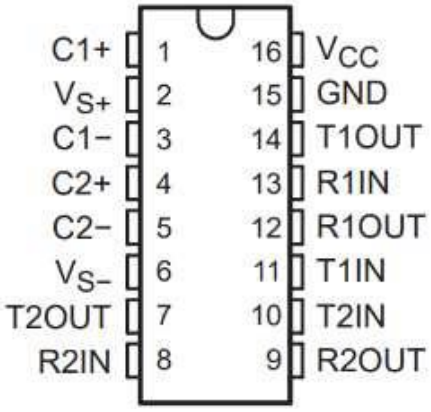

Рисунок 14 – Распиновка MAX232

В таблице 3 рассмотрено назначение всех выходов рассматриваемого модуля.

Таблица 3 – Назначение пинов MAX232

Номер	Обозначение	Описание
1	C1+	Положительный вывод C1 для подключения конденсатора
2	V_S+	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	V_S–	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

Продолжение таблицы 3

10, 11	T2IN, T1IN	Ввод логических данных
15	GND	Заземление
16	V_{CC}	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.

1.2.2.6 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 15 представлен внешний вид HC-05, а также назначение контактов этого модуля.

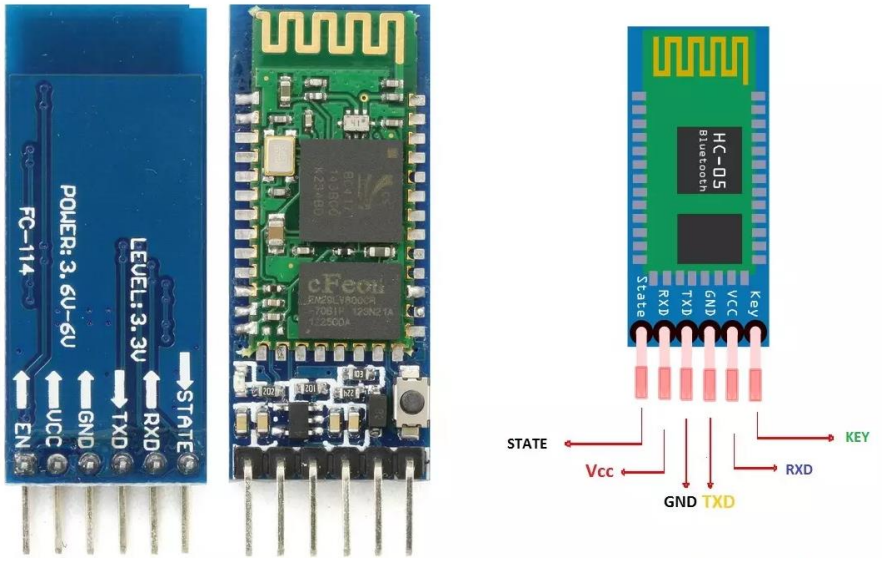

Рисунок 15 – Внешний вид и назначение контактов HC-05

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 16.

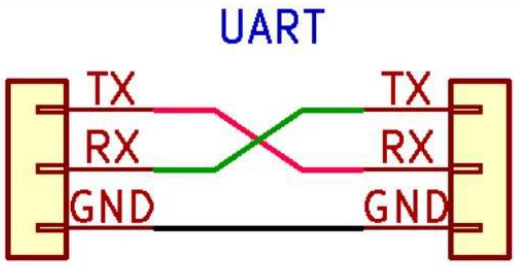

Рисунок 16 – Общая схема подключения устройств, обменивающихся сообщениями по каналу UART

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. Так как шина UART состоит из двух линий, было решено использовать переключатель типа DPDT – два полюса, два направления, что означает, что существуют два полюса, каждый из которых может быть подключен то к одному контакту, то к другому. На рисунке 17 представлено схематическое обозначение данного переключателя.

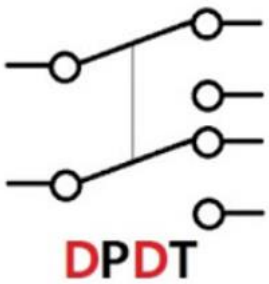

Рисунок 17 – DPDT-переключатель

1.2.4 Настройка частоты микроконтроллера ATmega8515

Для работы микроконтроллера необходим генератор тактовых импульсов. ATmega8515 имеет встроенный генератор, частоту которого можно настроить программно. Для высокой точности измерения времени было решено использовать внешний резонатор, который подключается к разъемам микроконтроллера XTAL1 и XTAL2.

Для работы микроконтроллера было решено использовать частоту равную 3,6864 МГц, так как именно такая частота обеспечивает надежную работу канала UART.

1.2.5 Настройка передачи данных по каналу UART

Для передачи данных по каналу UART необходимо настроить ряд регистров управления, предназначенных для UART (USART). Для начала рассмотрим используемые биты регистра UCSRB, который представлен в таблице 4.

Таблица 4 – Регистр UCSRB

Номер	7	6	5	4	3	2	1	0
Бит	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ	RXB8	TXB8

В регистре UCSRB необходимо выставить в единицу следующие биты:

– TXEN – включение передатчика, то есть разрешение отправки данных;

– RXEN – включение приемника, то есть разрешение получения данных;

– RXCIE – разрешение прерывания при приеме очередного байта сообщения.

Далее рассмотрим регистр UCSRC, представленный в таблице 5.

Таблица 5 – Регистр UCSRC

Номер	7	6	5	4	3	2	1	0
Бит	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ 1	UCSZ0	UCPOL

В регистре UCSRC необходимо выставить в единицу следующие биты:

– URSEL – последующие биты записываются в регистр UCSRC, а не в UBRRH;

– UCSZ1 и UCSZ0 – установка формата кадра данных, 8-битный формат.

Для настройки скорости передачи данных необходимо настроить регистр контроллера скорости передачи UBRR, который состоит из двух 8-битных регистров UBRRH (старший байт) и UBRRL (младший байт). Для передачи данных была выбрана скорость 38400 бод, которая является лучшей скоростью для передачи данных по Bluetooth. Для вычисления значения регистра UBRR воспользуемся формулой:

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

где $BAUD$ – скорость передачи данных (в бодах), f_{CLK} – тактовая частота микроконтроллера. Используя эту формулу, получим формулу для вычисления значения регистра $UBRR$:

$$UBRR = \frac{f_{CLK}}{16 \cdot BAUD} - 1.$$

Данное значение вычисляется программно, но для проверки можно вычислить вручную, подставив известные значения:

$$UBRR = \frac{3686400}{16 \cdot 38400} - 1 = 5.$$

Таким образом, для установки нужной скорости передачи данных достаточно установить значение, равное 5, в регистре UBRRL.

1.2.6 Настройка таймера микроконтроллера для отсчета времени

В качестве таймера для отсчета времени используется восьмиразрядный таймер/счётчик T0 в режиме CTC (Clear Timer on Compare Match). В этом режиме происходит очистка таймера при совпадении его значения с регистром сравнения OCR0.

Для настройки таймера T0 используются следующие регистры:

- TCCR0 (Timer/Counter Control Register) - регистр управления таймером;
- OCR0 (Output Compare Register) - регистр сравнения;
- TIMSK (Timer/Counter Interrupt Mask Register) - регистр маски прерываний таймера.

Структурная схема таймера T0 представлена на рисунке 18.

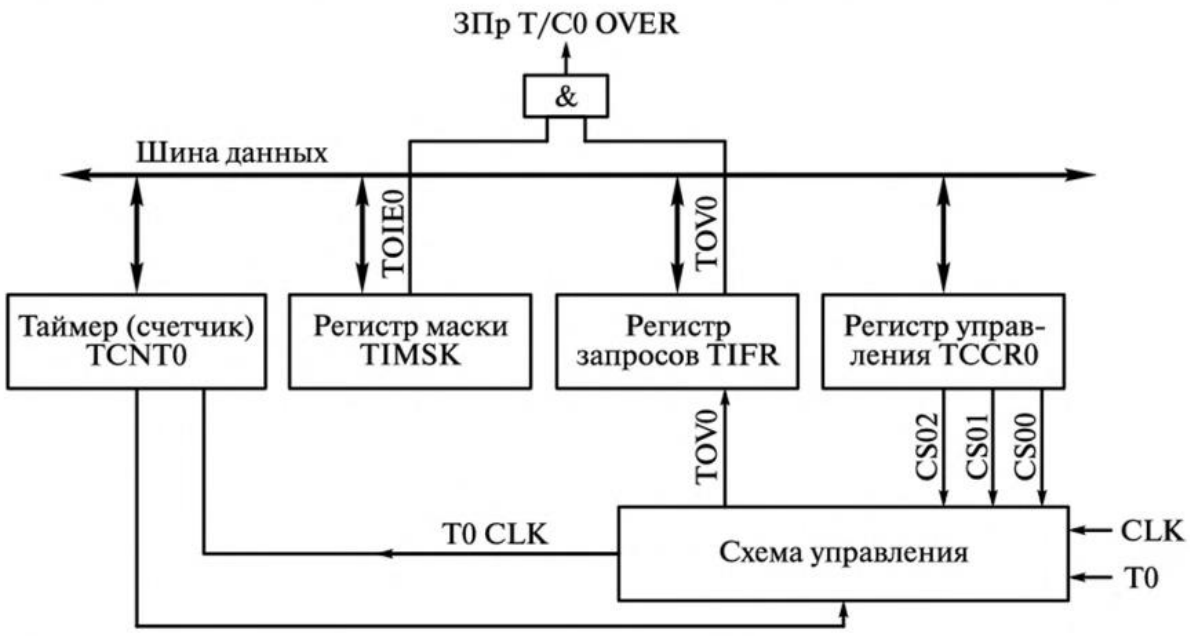

Рисунок 18 – Структурная схема таймера T0

Для включения режима CTC необходимо установить бит WGM01 в регистре TCCR0. В этом режиме счетчик сбрасывается в 0, когда значение в регистре таймера/счетчика (TCNT0) достигает значения OCR0.

Значение регистра сравнения OCR0 вычисляется по формуле:

$$OCR0 = \frac{f_{CLK}}{\text{Предделитель} \times f} - 1,$$

где f_{CLK} – тактовая частота микроконтроллера, f – желаемая частота (1000 Гц для миллисекундного таймера), предделитель – устанавливается в регистре TCCR0, равен 64.

$$OCR0 = \frac{3{,}6864 \times 10^6}{64 \times 1000} - 1 = 57$$

Таким образом, значение регистра сравнения должно быть равно 57.

Для генерации прерывания по совпадению необходимо установить бит OCIE0 в регистре TIMSK.

Для установки предделителя и запуска работы таймера необходимо выставить разряды CS01 и CS00 в регистре управления TCCR0 в единицу.

1.2.7 Построение функциональной схемы

На основе всех вышеописанных сведений и принятых при проектировании решений построена функциональная схема, представленная в графической части расчетно-пояснительной записки.

На схеме представлена кнопка RESET, которая подключена к соответствующему контакту микроконтроллера. Эта кнопка используется для ручного сброса и перезапуска работы микроконтроллера. Она имеет рекомендованную схему подключения, описанную в официальной документации к микроконтроллерам AVR. При включении схемы конденсатор разряжен и напряжение на контакте RESET близко к нулю, следовательно микроконтроллер не работает. Но спустя очень короткий промежуток времени, конденсатор зарядится и напряжение на RESET достигнет логической единицы и МК запустится. Основываясь на информации из документации, для элементов, необходимых для подключения данной кнопки, были использованы следующие номиналы: резистор – 10 кОм, конденсатор – 100 пФ.

1.3 Проектирование принципиальной схемы

На основе функциональной схемы разрабатывается принципиальная схема. Принципиальная схема должна содержать всю информацию, необходимую для конструирования проектируемого модуля [3].

1.3.1 Подключение цепи питания

Цепь питания для разрабатываемого устройства состоит из гнезда питания DS-201 и стабилизатора напряжения. Как было определено ранее все компоненты работают при напряжении, равном 5 В. В связи с этим выбран стабилизатор напряжения КР142ЕН5А, который преобразовывает входное напряжение 12 В от внешнего блока питания в 5 В для питания используемых компонентов. Схема выбранного стабилизатора представлена на рисунке 19 [11].

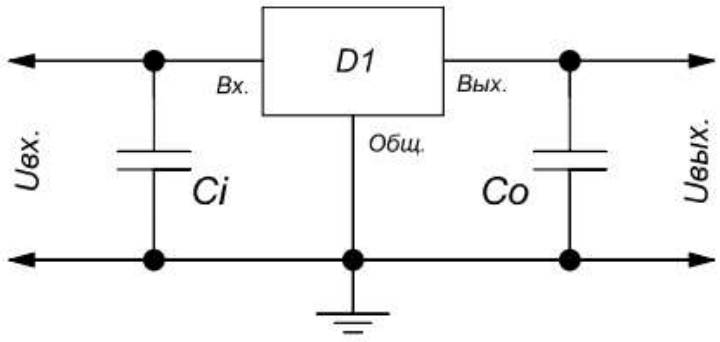

Рисунок 19 – Схема стабилизатора напряжения КР142ЕН5А

У стабилизатора три контакта:

– первый – входное напряжение 12 В;
– второй – заземление;
– третий – выходное напряжение 5 В.

При подключении стабилизатора напряжения к микроконтроллеру также необходимо использовать сглаживающие конденсаторы емкостью 2,2 мкФ и 1 мкФ.

Датчики влажности DHT-22 и расходометры YF-S201 в данной системе подключаются только по линиям передачи данных (Data pin). Их питание может осуществляться от отдельного источника питания, что позволяет снизить

нагрузку на основной источник питания микроконтроллера и повысить помехоустойчивость системы.

Но на принципиальной схеме, для того, чтобы упростить конечную реализацию датчики и расходометры запитаны от выходного напряжения со стабилизатора. Также реализован блок питания помп. В данном случае питание подается напрямую с 12 вольтового входного напряжения, а затем управляется через управляющие сигналы на транзисторах.

1.3.2 Расчет потребляемой мощности

Потребляемая мощность – это мощность, потребляемая интегральной схемой, которая работает в заданном режиме соответствующего источника питания. Чтобы рассчитать потребляемую мощность всего разрабатываемого устройства, необходимо вычислить суммарную мощность всех используемых элементов.

На все устройства подается напряжение 5 В. Мощность, потребляемая один устройством, в статическом режиме, рассчитывается следующей формулой:

$$P = I \cdot U,$$

а для резисторов используется следующая формула:

$$P = \frac{U^2}{R},$$

где P – мощность, I – ток потребления микросхемы, R – сопротивление резистора.

В схеме используются: один резистор номиналом 4,7 кОм, один резистор номиналом 10 кОм и два резистора номиналом 150 Ом. Вычислим потребляемую мощность для этих резисторов: $P_R = \frac{(5\text{В})^2}{4700\text{Ом}} + \frac{(5\text{В})^2}{10000\text{Ом}} + \frac{2\cdot(5\text{В})^2}{150\text{Ом}} \approx 341\text{мВт}$.

Для определения тока потребления ATmega8515 следует воспользоваться графиком тока через вывод микроконтроллера в зависимости от частоты и питающего напряжения, который изображен на рисунке 20. Микроконтроллер работает при напряжении питания 5 В на частоте 3,6864 МГц.

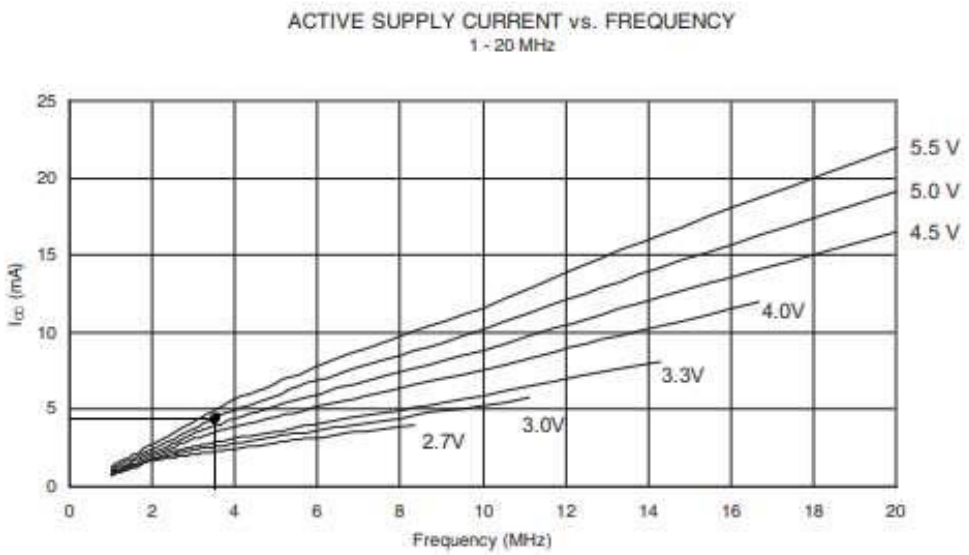

Рисунок 20 – Зависимость тока от частоты микроконтроллера

Из представленного рисунка видно, что при заданных параметрах ток потребления одного выхода микроконтроллера равен 4,93 мА. Всего используемых линий ввода/вывода настроенных на работу – 24. Следовательно, можно считать, что общий ток потребления $I_{ATmega8515} = 4{,}93\text{мА} \cdot 24 = 118{,}32\text{мА}$.

Проанализировав официальную документацию по ЖКД LM016L было получено значение тока потребления данного дисплея – 3 мА.

Получив все нужные значения тока потребления, составим таблицу потребляемых мощностей, представленных в таблице 6.

Таблица 6 – Потребляемые мощности используемых модулей

Модуль	Ток потребления, мА	Напряжение питания, В	Потребляемая мощность, мВт
ATmega8515	118,32	5	591,6
LM016L	3	5	15
DHT-22(2 шт.)	3	5	15
YF-S201(2 шт.)	3	5	150

Общая потребляемая мощность устройства равна сумме мощностей его элементов. Для разрабатываемого устройства формула суммарной потребляемой мощности:

$$P_{\text{сумм}} = P_{ATmega8515} + P_{LM016L} + P_R + 2P_{DHT-22} + 2P_{YF-S201}.$$

Подставим значения, полученные в таблице 6, в формулу и получим нужное значение:

$$P_{\text{сумм}} = 591{,}6\text{мВт} + 15\text{мВт} + 15\text{мВт} + 150\text{мВт} + 341\text{мВт} = 1112{,}6\text{мВт}.$$

Таким образом, суммарная потребляемая мощность разрабатываемого устройства равна 1112,6 мВт.

1.3.3 Построение принципиальной схемы

На основе всех вышеописанных сведений и принятых при проектировании решений построена принципиальная схема, представленная в графической части расчетно-пояснительной записки. Принципиальная схема содержит всю необходимую информацию для конструирования платы проектируемого модуля.

Особое внимание в схеме уделено разъему питания DS-201, который позволяет пользователю самостоятельно определять источник питания для устройства. Благодаря этому, важным является не сам источник, а возможность его подключения к указанному разъему, что обеспечивает универсальность и удобство эксплуатации.

Питание помп решено осуществить с использованием клеммной колодки ТВ-02А, которая позволяет упростить подачу питания на помпы, а также является универсальным решением, не привязанным к конкретному разъему.

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.

При запуске устройства на дисплее отображается информация о текущей зоне полива в формате "Zone N HH:MM/HH%", где N - номер зоны (1-2), HH:MM - время старта полива, HH% - текущий уровень влажности. На второй строке отображается выбранный параметр зоны и его значение.

Микроконтроллер принимает следующие команды по каналу UART:

– “data;” – запрос состояния всех зон;
– “time:HH:MM:SS;” – команда для установки заданного времени, указанного в виде целых чисел;
– “zoneN:on;” – включение полива на выбранной зоне;
– “zoneN:off;” – выключение полива выбранной зоны.

Микроконтроллер отправляет следующие сообщения:

— "ZoneN: H=XX% F=YL/min A=Z" - состояние зоны, где XX - влажность, Y - расход, Z - активность (0/1)
— "OK" - подтверждение выполнения команды;
— "ERROR" - ошибка в формате команды;
— "System started" - сообщение при запуске системы.

Управление устройством осуществляется кнопками:

— Zone - переключение между зонами
— Param - выбор параметра (расписание/влажность/время/расход)
— V+/V- - изменение значения параметра
— Time - переключение в режим установки времени
— Manual - ручное включение/выключение текущей зоны

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

На рисунке 21 представлена схема алгоритма основной программы, запускаемой при включении устройства.

Рисунок 21 – 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

В технологической части изложена информация, связанная с программированием микроконтроллера и отладкой исходного кода программы. Особое внимание уделено реализации спроектированного мобильного приложения, а также загрузке этого приложения в смартфон. В этой же части проводится тестирование разработанных программ.

2.1 Программирование микроконтроллера

Существует множество способов программирования AVR-микроконтроллеров:

- последовательное программирование при высоком напряжении питания (+12В);
- параллельное высоковольтное программирование;
- последовательное программирование при низком напряжении по интерфейсу SPI;
- через интерфейс JTAG;
- через Bootloader (самопрограммирование).

В данной работе используется внутрисхемное программирование через канал SPI, так как этот вариант самый популярный и простой на практике. Для программирования микроконтроллера указанным способом используются порты PB5 – PB7.

2.1.1 Интерфейс SPI

Интерфейс SPI используется для организации высокоскоростного канала связи между микроконтроллером и различными периферийными устройствами, а также для обмена данными между микроконтроллерами. В состав модуля SPI входят:

- 8-разрядный сдвиговый регистр SPDR, который принимает байт данных с шины данных микроконтроллера;
- 8-разрядные регистры управления SPCR и состояния SPSR;
- предварительный делитель частоты и схемы управления.

Регистр управления SPCR представлен в таблице 7.

Таблица 7 – Регистр управления SPCR

Номер	7	6	5	4	3	2	1	0
Бит	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

В регистре SPCR представленные биты имеют следующие назначения:

- SPIE (SPI Interrupt Enable) – бит, который разрешает прерывания;
- SPE (SPI Enable) – бит, включающий шину SPI;
- DORD (Data Order) – бит, устанавливающий порядок отправки бит. Если он установлен в 1, то первым отправляется младший бит, если в 0 – старший;
- MSTR (Master/Slave Select) – бит, который назначает устройство ведущим (бит должен быть установлен в единицу) либо ведомым (бит должен быть установлен в ноль);
- CPOL (Clock Polarity) – полярность синхронизации, определяет, при каком фронте синхронизирующего импульса будет инициироваться режим ожидания;
- CPHA (Clock Phase) – бит, отвечающий за фазу тактирования, то есть по какому именно фронту будет осуществляться передача бита;
- SPR1, SPR0 (SPI Clock Rate Select) – это биты, отвечающие за значение делителя частоты синхронизации, работают совместно с битом SPI2X, находящемся в регистре состояния.

Регистр состояния SPSR представлен в таблице 8.

Таблица 8 – Регистр состояния SPSR

Номер	7	6	5	4	3	2	1	0
Бит	SPIF	WCOL	–	–	–	–	–	SPI2X

В регистре SPSR представленные биты имеют следующие назначения:

– SPI2X (Double SPI Speed Bit) − бит, отвечающий за удваивание скорости, работает совместно с битами SPR1 и SPR0 управляющего регистра;
– SPIF (SPI Interrupt Flag) − флаг прерывания: как только байт от другого устройства появится полностью в буфере, то данный флаг установится, работает только в случае установки бита, разрешающего прерывания, а также при установленном разрешении глобальных прерываний;
– WCOL (Write Collision Flag) − флаг конфликта или коллизий, установится в том случае, если во время передачи данных будет конфликт битов, если во время передачи данных выполнится попытка записи в регистр данных.

2.1.2 Прошивка микроконтроллера

Для загрузки программы в память микроконтроллера используется программатор ISP500, который подключается через специальный разъем IDC-06MS, который позволяет использовать способ программирования по интерфейсу SPI. Выбранный разъем имеет следующие контакты:

– SCK – тактовый си гнал;
– MISO – для передачи данных от микроконтроллера в программатор;
– MOSI – для передачи данных от программатора в микроконтроллер;
– RESET – сигналом на RESET программатор вводит контроллер в режим программирования;
– GND – земля;
– VCC – питание.

Перевод микроконтроллера в режим программирования осуществляется подачей низкого логического уровня на линию RESET. Затем необходимо выждать не менее 20 мс, после чего послать инструкцию разрешения программирования на вывод MOSI интерфейса SPI. Программировать микроконтроллер следует при напряжении от 2,7 В до 6,0 В. [5]

Программирование по интерфейсу SPI осуществляется путем посылки 4-байтовых команд на вывод MOSI МК, в которых один или два байта определяют тип операции, а остальные – адрес, записываемый байт, установочные биты и биты защиты, холостой бит. При выполнении операции чтения считываемый байт снимается через вывод MISO. Передача команд и вывод результатов их выполнения осуществляется от старшего разряда к младшему. Необходимые команды для программирования ATmega8515 с помощью интерфейса SPI представлен в таблице 9. [5]

Таблица 9 – Команды для программирования ATmega8515 по каналу SPI

Команда	Формат команды				Операция
	Байт 1	Байт 2	Байт 3	Байт 4	
Разрешение программирования	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Разрешение последовательного программирования после подачи лог. 0 на RESET
Стирание кристалла	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Очистка Flash- и EEPROM- памяти
Чтение Flash-памяти	0010 H000	xxxx aaaa	bbbb bbbb	oooo oooo	Чтение старшего (или младшего) байта по адресу a:b
Запись в Flash-память	0100 H000	xxxx xxxa	bbbb bbbb	iiii iiii	Запись старшего (или младшего) байта по адресу a:b

Продолжение таблицы 9

Чтение EEPROM-памяти	1010 0000	xxxx xxxa	bbbb bbbb	oooo oooo	Чтение данных o из EEPROM по адресу a:b
Запись в EEPROM-память	1100 0000	xxxx xxxa	bbbb bbbb	iiii iiii	Запись данных i в EEPROM по адресу a:b
Запись битов защиты	1010 1100	111x xxxx	xxxx xxxx	11ii iiii	Запись бит защиты. Запись "0" приводит к программированию бита защиты
Чтение сигнатурного бита	0011 0000	xxxx xxxx	xxxx xxbb	oooo oooo	Чтение сигнатурного байта o по адресу b

В таблице 9 используются следующие обозначения:

- a – адрес старших разрядов;
- b – адрес младших разрядов;
- H = 0 – младший байт, H = 1 – старший байт;
- o – данные при чтении;
- i – данные при записи;
- x – произвольное значение.

Полный набор команд для программирования ATmega8515 представлен в официальной документации микроконтроллера.

2.1.3 Алгоритм прошивки микроконтроллера

Вся программа и данные загружаются во Flash-память микроконтроллера. В связи с этим целесообразно рассмотреть алгоритм входа в режим программирования и алгоритм записи во Flash-память.

В таблице 10 представлен алгоритм входа в режим программирования, необходимый для выполнения алгоритма записи во Flash-память.

Таблица 10 – Алгоритм входа в режим программирования

Номер	Действие
1	Подать напряжение 4.5 – 5.5 В между VCC и GND и подождать не менее 100 мкс.
2	Установить RESET в положение “0”, подождать не менее 100 нс и переключить XTAL1 не менее шести раз.
3	Установить биты PAGEL, XA1, XA0, BS1 регистра Prog_enable в ноль и подождать не менее 100 нс.
4	Подать напряжение 11.5 – 12.5В на RESET. Любая активность на пинах Prog_enable в течение 100 нс после того, как на RESET было подано напряжение +12В, приведет к сбою входа устройства в режим программирования.

Flash-память организована по страницам. При программировании Flash-памяти, данные программы захватываются в буфер страницы. Это позволяет одновременно программировать одну страницу данных программы. В таблице 11 представлен алгоритм записи во Flash-памяти.

Таблица 11 – Алгоритм записи во Flash-память

Номер	Действие
А. Загрузка команды «Запись во Flash-память»	
1	Установить XA1, XA0 в положение “10”. Это активирует загрузку команды.
2	Установить BS1 в “0”.
3	Установить DATA в “0001 0000”. Это команда для начала записи.

Продолжение таблицы 11

4	Подать на XTAL1 положительный импульс. Это загружает команду.
Б. Загрузка младшего байта адреса	
1	Установить XA1, XA0 в “00”. Это активирует загрузку адреса.
2	Установить BS1 в “0”. Это выбирает младший адрес.
3	Установите DATA = младший байт адреса ($00 - $FF).
4	Подать на XTAL1 положительный импульс. Это загружает младший байт адреса.
В. Загрузка младшего байта данных	
1	Установить XA1, XA0 в “01”. Это активирует загрузку данных.
2	Установить DATA = младший байт данных ($00 - $FF).
3	Подать на XTAL1 положительный импульс. Это загружает байт данных.
Г. Загрузка старшего байта данных	
1	Установить BS1 в “1”. Это выбирает старший байт данных.
2	Установить XA1, XA0 в “01”. Это активирует загрузку данных.
3	Установить DATA = старший байт данных ($00 - $FF).
4	Подать на XTAL1 положительный импульс. Это загружает байт данных.
Д. Защелкивание данных	
1	Установить BS1 в “1”. Это выбирает старший байт данных.
2	Подать на PAGEL положительный импульс. Это защелкивает байты данных.
Е. Повторить шаги Б – Д до тех пор, пока весь буфер не будет заполнен или пока все данные на странице не будут загружены.	
Ж. Загрузка старшего байта адреса	
1	Установить XA1, XA0 в “00”. Это активирует загрузку адреса.
2	Установить BS1 в “1”. Это выбирает старший адрес.
3	Установить DATA = старший байт адреса ($00 - $FF).

Продолжение таблицы 11

4	Подать на XTAL1 положительный импульс. Это загружает старший байт адреса.
З. Программирование страницы	
1	Установить BS1 = “0”.
2	Подать на $\overline{WR}$ отрицательный импульс. Это начинает программирование всей страницы данных. RDY/$\overline{BSY}$ переходит в низкое состояние.
3	Подождать, пока RDY/$\overline{BSY}$ не перейдет в высокое состояние.
И. Повторить шаги Б – З до тех пор, пока вся Флэш не будет запрограммирована или пока все данные не будут запрограммированы.	
К. Завершение программирования страницы	
1	Установить XA1, XA0 в “10”. Это активирует загрузку команды.
2	Установить DATA в “0000 0000”. Это команда для операции без действия.
3	Подать на XTAL1 положительный импульс. Это загружает команду, и внутренние сигналы записи сбрасываются.

2.2 Настройка и сборка проекта

В данном разделе изложена информация, необходимая для подготовки спроектированного устройства к его работе. Эта информация включает в себя выбор средств разработки, сборку программного кода, написанного по разработанным ранее схемам алгоритмов, в исполняемый файл и настройку всех компонентов устройства для последующей отладки

2.2.1 Выбор среды разработки

В качестве среды разработки для сборки исполняемого файла была выбрана програма Visual Studio Code. При настройке проекта был выбран компилятор AVR GCC, благодаря которому был скомпилирован необходимый исполняемый файл. Откладка проекта в среде Proteus показала, что программа ведет себя так, как и ожидается.

Как уже было упомянуто выше, для отладки проекта на собранном устройстве было решено использовать среду Proteus 8 Professional. Она позволяет проводить симуляцию собранного устройства с загруженной в микроконтроллер программой.

2.2.2 Сборка проекта

Структура разработанных программных модулей представлена на рисунке 32.

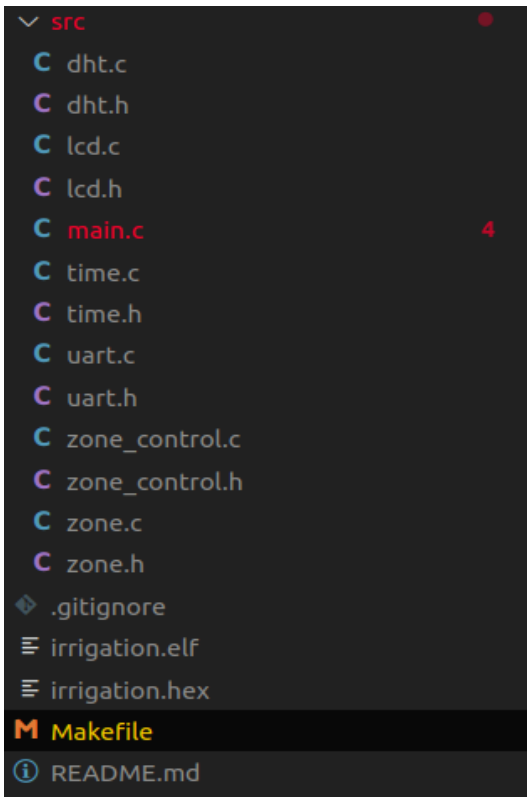

Рисунок 22 – Список программных модулей

После выполнения сборки разработанного проекта в окне сообщений отображается информация, представленная на рисунке 33.

```
AVR Memory Usage
----------------
Device: atmega8515

Program:    6320 bytes (77.1% Full)
(.text + .data + .bootloader)

Data:        374 bytes (73.0% Full)
(.data + .bss + .noinit)


Build succeeded with 0 Warnings...
```

Рисунок 23 – Сообщение, отображаемое после сборки

Это сообщение показывает объем используемой памяти микроконтроллера, а также сообщает об успешном окончании сборки проекта.

Сборка проекта создает два исполняемых файла с форматами .hex и .elf. Оба эти файла могут быть загружены в память микроконтроллера в среде Proteus. Основное отличие файла с форматом .elf заключается в том, что это объектный модуль с информацией для отладчика, в том числе и отладчика в среде Proteus. Это очень полезно на этапе разработки устройства: можно отладить работу программы, обнаружить какие-либо недочеты. Если программа уже отлажена и работает исправно, то можно использовать файл с форматом .hex – логика работы устройства не поменяется.

2.2.3 Настройка проекта в среде Proteus

В данном разделе рассматриваются этапы настройки компонентов, используемых в разрабатываемом устройстве.

2.2.3.1 Настройка микроконтроллера

На рисунке 34 представлено окно настройки параметров микроконтроллера в среде Proteus.

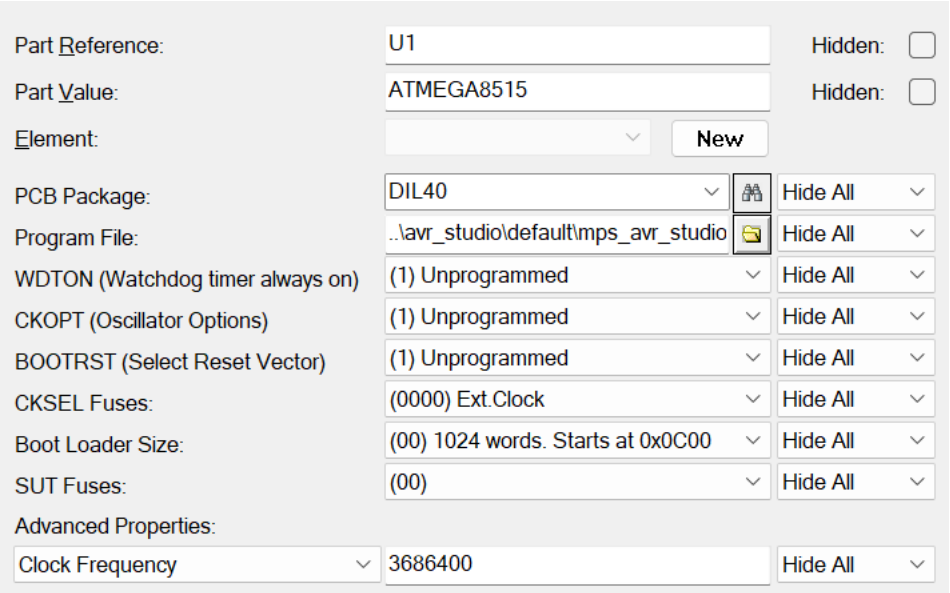

Рисунок 24 – Окно настройки ATmega8515 в среде Proteus

Здесь важно настроить следующие параметры:

– Program File – здесь нужно указать путь к исполняемому файлу в формате .hex или .elf;
– CKSEL Fuses – здесь указывается источник тактовых сигналов микроконтроллера, а именно «внешний источник», так как ранее мы определили, что для лучшей точности будет использован внешний источник тактовых сигналов;
– Clock Frequency – здесь указывается частота внешнего генератора тактовых сигналов (указываем выбранную частоту – 3,6864 МГц), что позволяет не подключать в среде Proteus кварцевый резонатор к XTAL1 и XTAL2.

2.2.3.2 Настройка компонентов для связи с внешними ПЭВМ

Как было определено ранее, устройство может обмениваться сообщениями с внешними ПЭВМ сообщениями по каналу UART с помощью разъема RS-232 или с помощью Bluetooth-модуля HC-05. В среде Proteus существует эмулятор разъема RS-232, настройки которого представлены на рисунке 35.

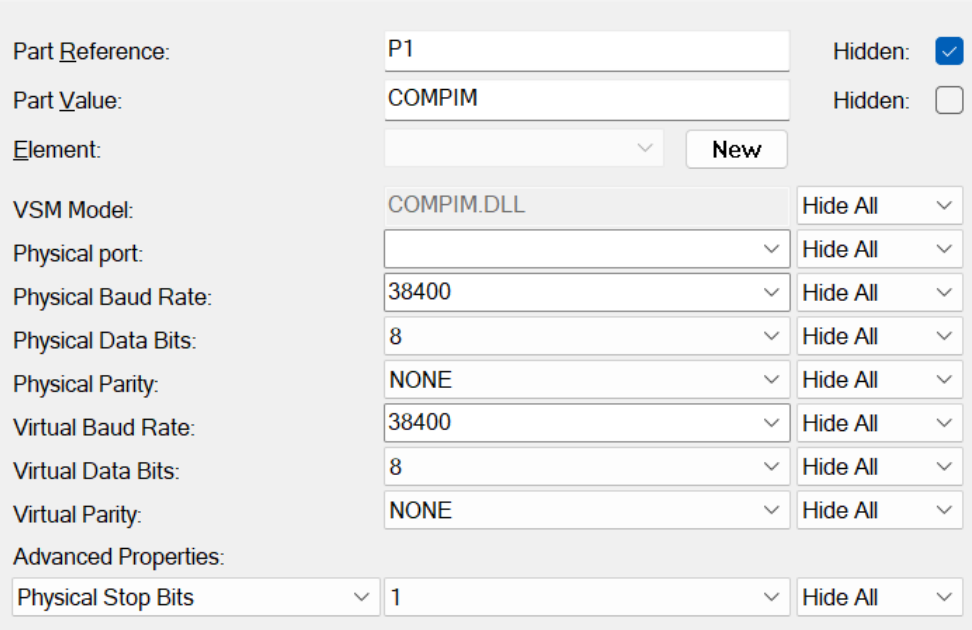

Рисунок 25 – Настройки эмулятора разъема RS-232

В первую очередь необходимо настроить скорость передачи данных. Это делается с помощью выставления выбранной скорости передачи данных (38400 бод) в соответствующие поля: Physical Baud Rate и Virtual Baud Rate.

Далее необходимо выбрать физический последовательный порт, к которому подключается разъем. И тут есть некоторые сложности. Дело в том, что так как на самом деле физически мы ничего не подключаем к ПК, на котором используется симуляция в среде Proteus, то и никаких физических портов у нас нет. Помимо этого, стоит еще раз отметить, что MAX232 инвертирует сигналы, передаваемые по каналу UART, а внешние программы-терминалы, с помощью которых мы могли бы обмениваться сообщениями с симуляцией в среде Proteus предварительно настроив виртуальные последовательные порты, не имеют функции обратного инвертирования сигналов, а следовательно, получаемые и передаваемые сообщения были бы нераспознаваемыми ни для пользователя, ни для микроконтроллера. В связи с этим было принято решение использовать виртуальный терминал, встроенный в среду Proteus, вместо RS-232 для симуляции обмена сообщениями с ПК. Для этого необходимо подключить контакты T1OUT и R1IN компонента MAX232 к контактам TXD и RXD виртуального терминала соответственно. В

настройках терминала необходимо установить используемую скорость – 38400.

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.

В операционной системе Windows предусмотрена возможность создания и автоматического назначение последовательного порта на Bluetooth-модуль. Для этого необходимо перейти в настройки системы и найти параметр «Дополнительные параметры Bluetooth», в открывшемся окне перейти во вкладку COM-порты и добавить новый COM-порт, с указанием направления «Входящий». Результат добавления нового COM-порта представлен на рисунке 36.

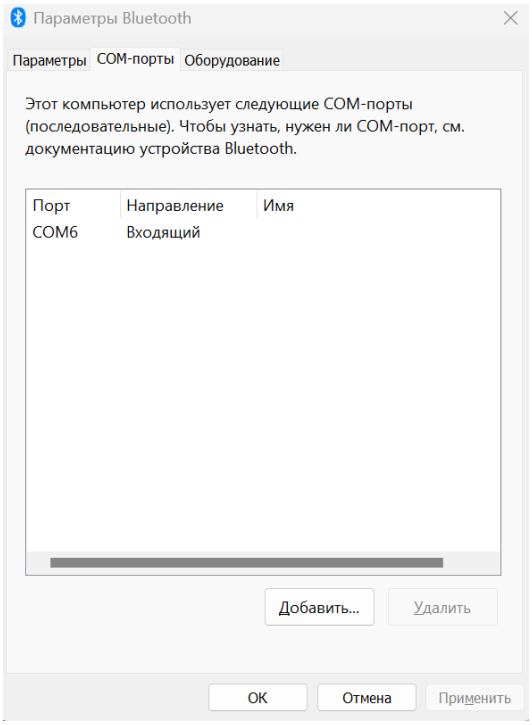

Рисунок 26 – Добавление COM-порта для Bluetooth-соединения

После добавления COM-порта становится известен его порядковый номер – COM6. Теперь мы можем вернуться к настройкам RS-232, представленным на рисунке 35, и указать в этих настройках созданный последовательный порт. Любое Bluetooth-подключение к используемому ПК будет перенаправляться на симуляцию в Proteus, и все данные, которые будут передаваться по Bluetooth будут идти на эмулятор разъема RS-232. Таким образом, в симуляции получилось реализовать функционал модуля HC-05 через эмулятор разъема RS-232 в среде Proteus.

Но в процессе реализации выяснилось, что у современных мобильных устройств на базе Android есть проблема с сопряжением по Bluettoth как с Windows 11, так и с Windows 10. 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.

При этом логика работы приложения и протокол обмена данными остались неизменными - изменился только способ доставки команд от мобильного приложения к устройству. Это демонстрирует гибкость разработанной архитектуры, где транспортный уровень может быть легко заменен без изменения основной функциональности.

2.2.3.3 Симуляция в среде Proteus

Остальные компоненты разрабатываемой схемы не требуют дополнительных настроек – их достаточно подключить согласно разработанной принципиальной схеме.

Собранная и запущенная схема итогового устройства представлена на рисунке 27.

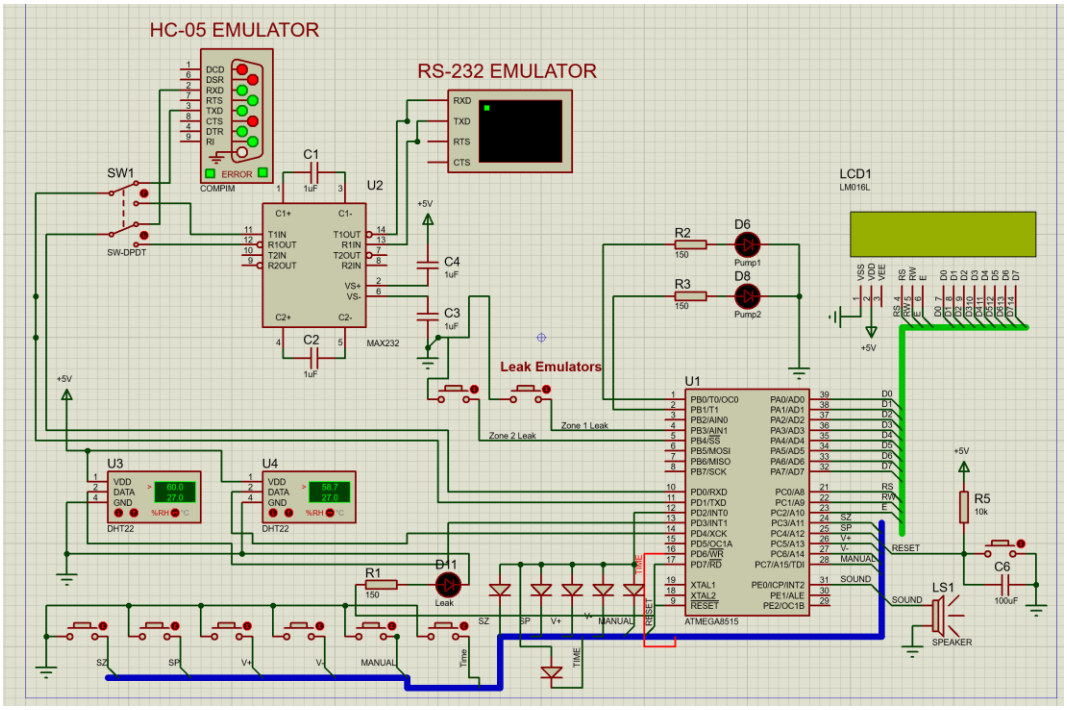

Рисунок 27 – Разработанная схема устройства в среде Proteus

2.3 Тестирование системы автоматического полива

В этом разделе проводится тестирование разработанного устройства. Тестирование делится на три этапа: тестирование работы самого устройства в среде Proteus, тестирование обмена сообщениями регистратора времени с внешним терминалом и тестирование управления через телефон.

2.3.1 Тестирование работы устройства в среде Proteus

На рисунке 28 представлен первоначальный запуск устройства и отображение информационной сообщения.

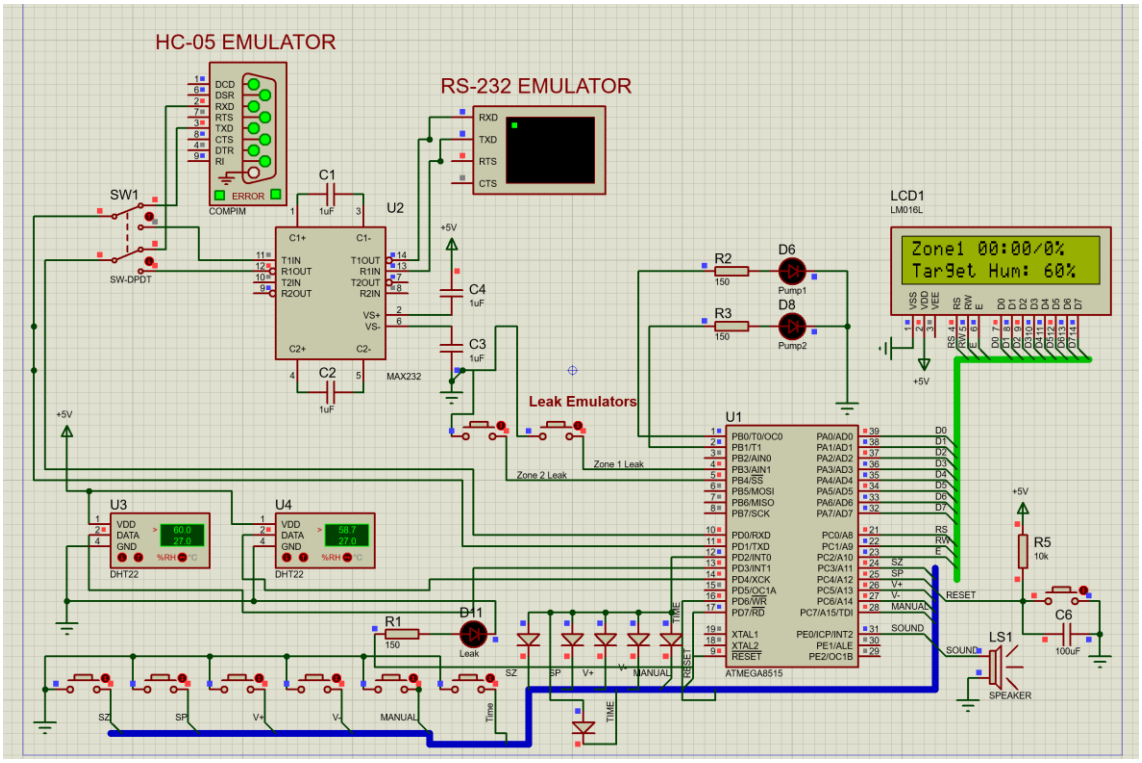

Рисунок 28 – Запуск устройства и отображение информационного сообщения

Пусть оператору нужно настроить параметры второй зоны. При запуске устройства на информационном экране отображаются параметры для первой зоны. Нажав кнопку “SZ” оператор получает доступ к настройке зоны 2. После чего использую “SP” переключается на нужный параметр и меняет его.

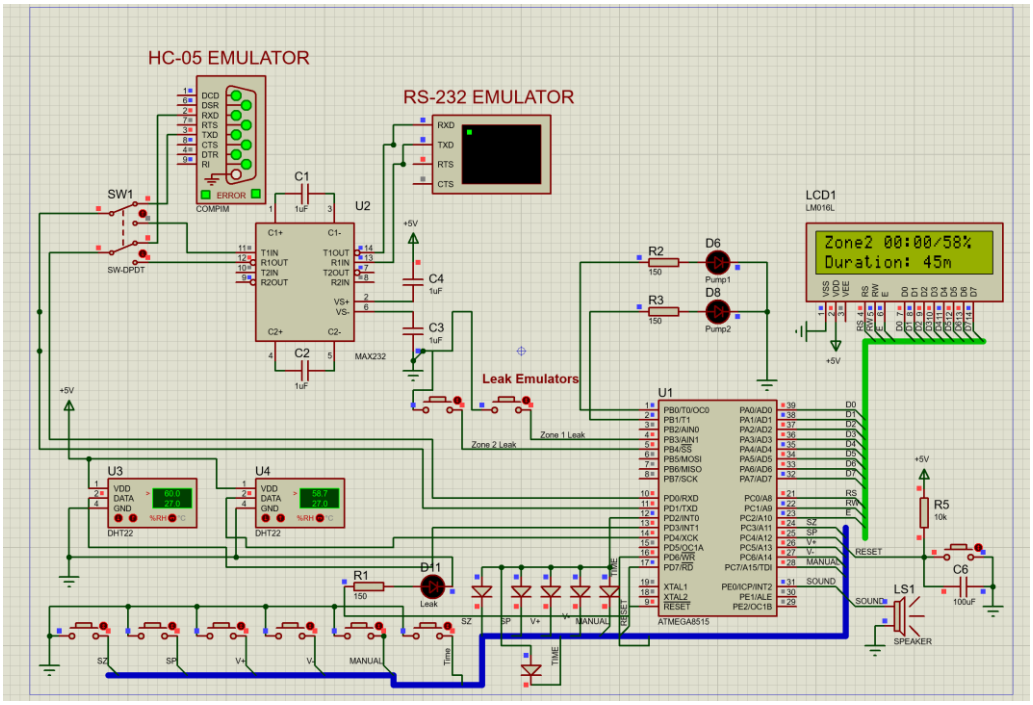

Рисунок 29 – Изменение параметра времени полива

Нажатием на кнопку Time получаем доступ к настройки системного времени.

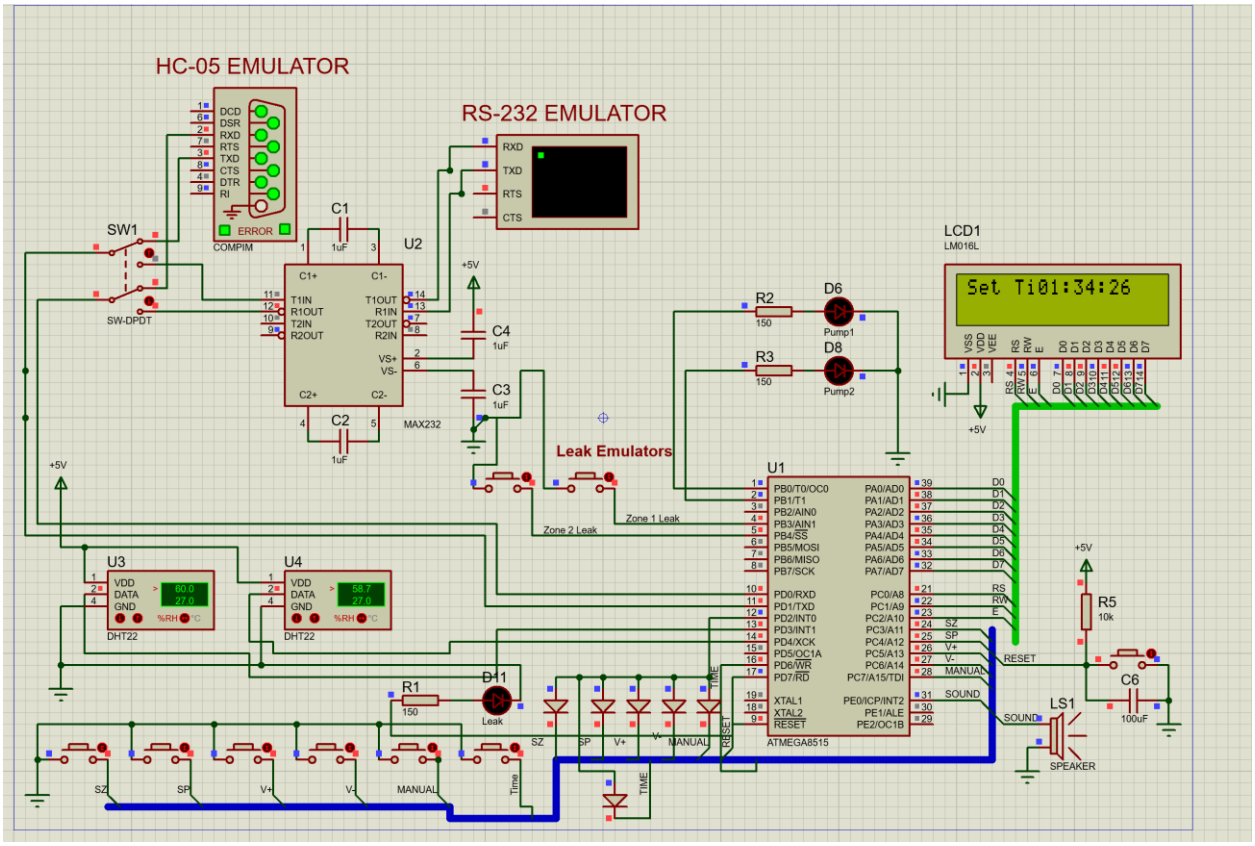

Рисунок 30 – Регистрация времени ухода сотрудника ID-5

При нажатии кнопки Manual происходит запуск полива в выбранной зоне.

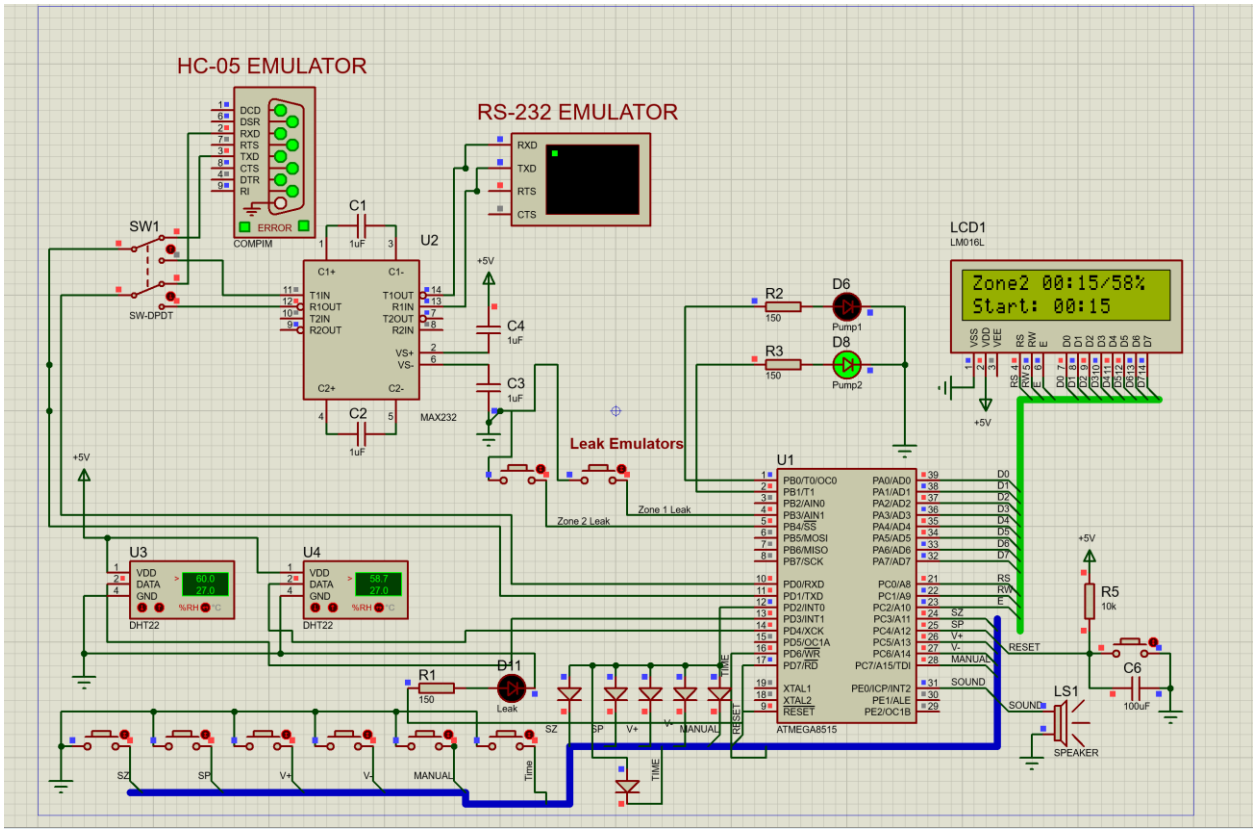

Рисунок 31 – Отображение сохраненной информации о времени прихода/ухода сотрудника с ID-5

Проверим работу системы при имитации утечки воды в первой зоне. Для этого замкнем контакт PB3, имитируя срабатывание датчика утечки. В результате на дисплее отобразится сообщение "Leak in Zone 1!", загорится красный светодиод аварийной индикации, подключенный к выводу PD7, и включится звуковой сигнал через пьезоизлучатель на выводе PE0. На рисунке 32 представлен результат обработки аварийной ситуации системой. Работа пьезоизлучателя отображается в симуляции сменой цвета вывода PE0 микроконтроллера с синего на серый при формировании звукового сигнала.

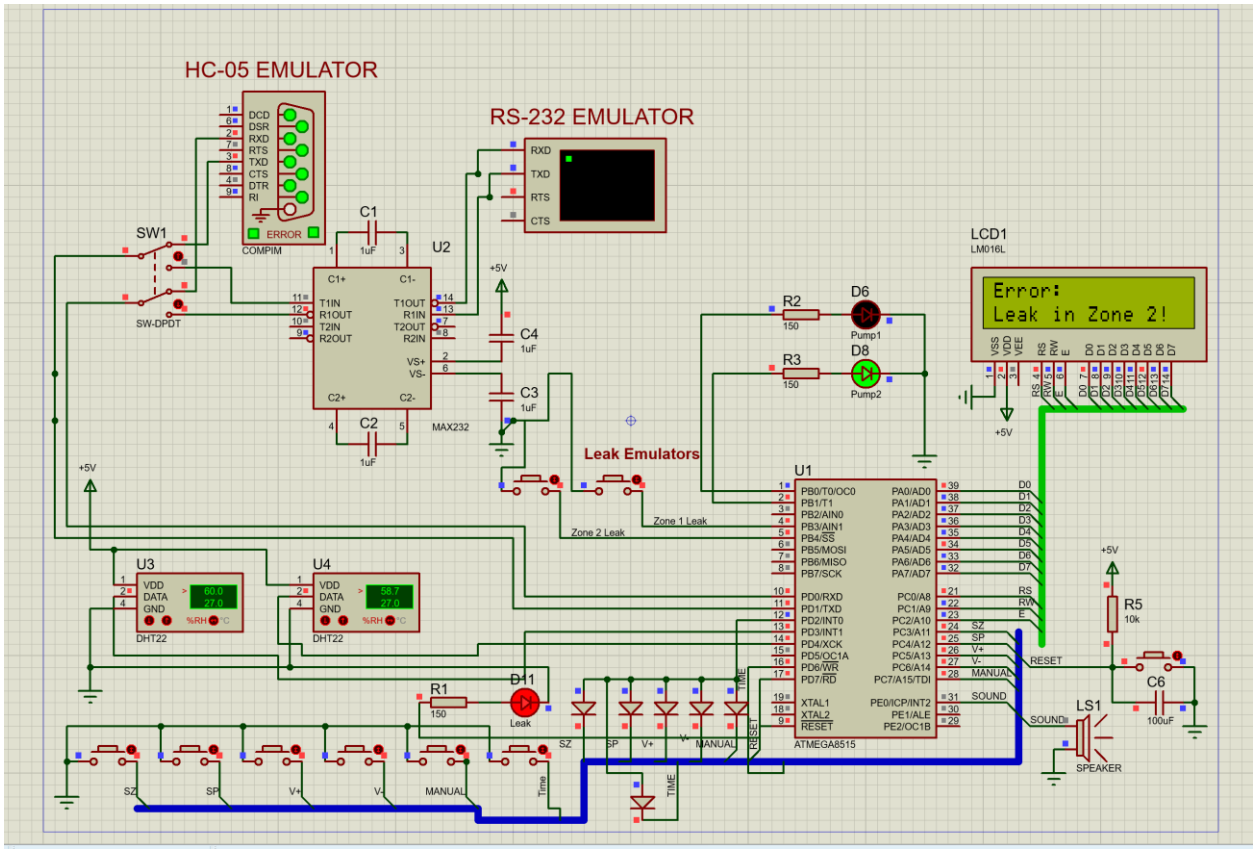

Рисунок 32 – Проверка обработки утечки

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы по дисциплине «Микропроцессорные системы» была спроектирована и реализована система автоматического полива растений на базе микроконтроллера ATmega8515. Система обеспечивает управление двумя независимыми зонами полива с функцией контроля влажности почвы и мониторинга расхода воды.

Для измерения уровня влажности были применены датчики DHT-22, а для учета расхода воды использовались расходометры YF-S201. Управление помпами осуществляется автоматически в соответствии с заданным расписанием, при этом предусмотрена функция обнаружения утечек, позволяющая в экстренных ситуациях автоматически останавливать процесс полива. Система также поддерживает ручное управление, которое осуществляется либо с помощью аппаратных кнопок, либо через мобильный телефон. Связь между мобильным устройством и системой обеспечивается через Bluetooth.

Все этапы разработки были выполнены в соответствии с техническим заданием. В результате работы был создан полный комплект документации, включающий расчетно-пояснительную записку, электрические схемы, программный код и спецификацию радиоэлементов, использованных в устройстве.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 ГОСТ 7.32-2017 СИБИД. Отчет о научно-исследовательской работе. Структура и правила оформления [Электронный ресурс]. – 2017. – URL: https://docs.cntd.ru/document/1200157208 (дата обращения: 15.12.2023).

2 Хартов В.Я. Конспект лекций по дисциплине «Микропроцессорные системы». – 2023.

3 Хартов В.Я. Методические указания к выполнению курсовой работы по дисциплине «Микропроцессорные системы» : учебно-методическое пособие. – Москва: Издательство МГТУ им. Н.Э. Баумана, 2021. – 31 с.

4 Хартов В.Я. Методические указания для выполнения лабораторных работ по дисциплине «Микропроцессорные системы». – 2023.

5 Документация микроконтроллера ATmega8515 [Электронный ресурс]. – URL: https://ww1.microchip.com/downloads/en/DeviceDoc/doc2512.pdf (дата обращения: 01.12.2023).

6 Документация жидкокристаллического дисплея LM016L [Электронный ресурс]. – URL: https://datasheetspdf.com/pdf/1462370/Hitachi/LM016L/1 (дата обращения 01.12.2023).

7 Хабр. Введение в 1-Wire [Электронный ресурс]. – 2010. – URL: https://habr.com/ru/articles/101954/ (дата обращения 03.12.2023).

8 Aveon. Что такое интерфейс RS-232? [Электронный ресурс]. – URL: https://aveon.ru/services/004/ (дата обращения 15.12.2023).

9 Документация драйвера MAX232 Datasheet [Электронный ресурс]. – URL: https://web.fe.up.pt/~ee94159/proj/old/max232.pdf (дата обращения 15.12.2023).

10 Документация стабилизатора напряжения КР142ЕН5А [Электронный ресурс]. – URL: https://www.chipdip.ru/product/kr142en5a (дата обращения 25.11.2023).

11 ГОСТ 2.743-91 ЕСКД. Обозначения условных графических элементов в схемах. Элементы цифровой техники [Электронный ресурс]. – URL: https://znaytovar.ru/gost/2/GOST_274391_ESKD_Oboznacheniya.html (дата обращения 21.12.2023).
12 ГОСТ 2.701-84 Правила выполнения схем.
13 ГОСТ 2.702-75 Правила выполнения электрических схем.
14 Википедия. Универсальный асинхронный приёмопередатчик [Электронный ресурс]. – 2023. – URL: https://ru.wikipedia.org/wiki/Универсальный_асинхронный_приёмопередатчик#Сеть (дата обращения 29.12.2023).

ПРИЛОЖЕНИЕ А

Исходный код программы микроконтроллера

На n листах

Листинг 1 – Главная программа main.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "zone.h"
#include "time.h"
#include "lcd.h"
#include "uart.h"
#include "dht.h"
#include "zone_control.h"

// Глобальные переменные, доступные для других модулей
Zone zones[NUM_ZONES];
SystemTime systemTime;

// Локальные переменные
static uint8_t currentZone = 0;
static uint8_t selectedParam = 0;
static uint8_t displayNeedsUpdate = 1;

static volatile uint8_t lastButtonState = 0;
static volatile uint8_t lastTimeButtonState = 0;
static volatile uint16_t debounceTime = 0;

extern volatile uint8_t commandReady;

ISR(TIMER0_COMP_vect) {
    static uint16_t msCounter = 0;

    if(++msCounter >= 1000) {
        msCounter = 0;
        incrementTime(&systemTime);

        for(uint8_t i = 0; i < NUM_ZONES; i++) {
            updateZone(&zones[i], i, &systemTime);
        }

        if(systemTime.isSettingTime) {
            displayNeedsUpdate = 1;
        }
    }

    if(debounceTime > 0) {
        debounceTime--;
    }
}
```

```
static void initPorts(void) {
    // Кнопки зон
    DDRC &= ~((1<<PC3)|(1<<PC4)|(1<<PC5)|(1<<PC6)|(1<<PC7));
    PORTC |= (1<<PC3)|(1<<PC4)|(1<<PC5)|(1<<PC6)|(1<<PC7);

    // Настройка времени PD6
    DDRD &= ~(1<<PD6);
    PORTD |= (1<<PD6);

    // Детекция утечек
    DDRB &= ~((1<<PB3)|(1<<PB4));
    PORTB |= (1<<PB3)|(1<<PB4);

    // Выводы
    DDRB |= (1<<PB0)|(1<<PB1);      // Светодиоды помп
    PORTB &= ~((1<<PB0)|(1<<PB1));

    DDRD |= (1<<PD7);               // Светодиод утечки
    PORTD &= ~(1<<PD7);

    DDRE |= (1<<PE0);               // Пьезоизлучатель
    PORTE &= ~(1<<PE0);
}

static void handleButtons(void) {
    if(debounceTime > 0) return;

    uint8_t buttons = ~PINC & 0xF8;
    // char buf[32];
    uint8_t timeButton = ~PIND & (1<<PD6);

    if(buttons    !=    lastButtonState    ||    timeButton    !=
lastTimeButtonState) {
        debounceTime = 20;
        lastButtonState = buttons;
        lastTimeButtonState = timeButton;

        if(timeButton) {
            systemTime.isSettingTime                                =
!systemTime.isSettingTime;
            displayNeedsUpdate = 1;
            if(systemTime.isSettingTime) {
                updateTimeDisplay(&systemTime);
            } else {
                updateDisplay(zones,                currentZone,
selectedParam);
            }
        }
        else if(buttons) {
            if(systemTime.isSettingTime) {
                if(buttons  &  (1<<PC5))  adjustTime(&systemTime,
5);  // V+
```

```
                if(buttons & (1<<PC6)) adjustTime(&systemTime, -
5); // V-
                displayNeedsUpdate = 1;
            }
            else {
                if(buttons & (1<<PC3)) {
                    currentZone = (currentZone + 1) % NUM_ZONES;
                    //  sprintf(buf,  "Zone  change:  %d->%d",
currentZone, (currentZone + 1) % NUM_ZONES);
                    // uartSendString(buf);
                    displayNeedsUpdate = 1;
                }
                if(buttons & (1<<PC4)) {
                    selectedParam = (selectedParam + 1) % 4;
                    displayNeedsUpdate = 1;
                }
                if(buttons & (1<<PC5) || buttons & (1<<PC6)) {
                    int8_t change = (buttons & (1<<PC5)) ? 1 : -
1;
                    adjustParameter(&zones[currentZone],
selectedParam, change);
                    displayNeedsUpdate = 1;
                }
                if(buttons & (1<<PC7)) {
                    toggleManual(&zones[currentZone],
currentZone);
                    displayNeedsUpdate = 1;
                    // Отправляем статус по UART
                    sendZoneData(currentZone);
                }
            }
        }
    }
}

int main(void) {
    // Инициализация
    initPorts();
    initZones(zones);
    initSystemTime(&systemTime);
    initLcd();
    initUart();
    initDht();

    // Настройка таймера
    OCR0 = 57;
    TCCR0 = (1<<WGM01)|(1<<CS01)|(1<<CS00);
    TIMSK |= (1<<OCIE0);

    sei();

    // Начальное отображение
    updateDisplay(zones, currentZone, selectedParam);
```

```
    // Отправка сообщения о старте системы
    uartSendString("System started");

    while(1) {
        handleButtons();
        processDhtData(&zones[0], 0);
        processDhtData(&zones[1], 1);
        checkLeaks(zones);

        if(commandReady) {
            handleCommand();         // Обрабатываем команду в
основном цикле
            commandReady = 0;    // Сбрасываем флаг
        }

        if(displayNeedsUpdate) {
            if(systemTime.isSettingTime) {
                updateTimeDisplay(&systemTime);
            } else {
                updateDisplay(zones,                currentZone,
selectedParam);
            }
            displayNeedsUpdate = 0;
        }
    }

    return 0;
}
```

Листинг 2 – Заголовочный файл uart.h

```
#pragma once
#include "zone.h"
#include "time.h"
#include <util/delay.h>

// Базовые функции UART
void initUart(void);
void uartSendChar(char data);
void uartSendString(const char* str);
void sendCurrentData(void);
void sendZoneData(const uint8_t index);
```

Листинг 3 – Модуль uart.c

```
#include "uart.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include <stdio.h>
```

```
#define BAUD_RATE 38400

static char message[30];
static int messageIdx = 0;
volatile uint8_t commandReady = 0;

extern Zone zones[NUM_ZONES];
extern SystemTime systemTime;

// Простая отправка символа
void uartSendChar(char data) {
    while (!(UCSRA & (1 << UDRE)));
    UDR = data;
}

// Простая отправка строки
void uartSendString(const char* str) {
    while(*str) {
        uartSendChar(*str++);
    }
    uartSendChar('\r');
    uartSendChar('\n');
}

// Отправка числа напрямую без sprintf
static void uartSendNumber(uint8_t num) {
    // Для чисел до 255
    char buf[4];  // максимум 3 цифры + \0
    uint8_t i = 0;

    // Преобразуем число в строку
    do {
        buf[i++] = '0' + (num % 10);
        num = num / 10;
    } while(num > 0);

    // Отправляем в обратном порядке
    while(i > 0) {
        uartSendChar(buf[--i]);
    }
}

void sendZoneData(const uint8_t index) {
    uint8_t sreg = SREG;
    cli();

    // Сначала копируем все данные локально
    uint8_t humidity = zones[index].humidity;
    uint8_t flowRate = zones[index].flowRate;
    uint8_t isActive = zones[index].isActive;

    SREG = sreg;
```

```
    // Отправляем данные напрямую, без формирования строки
    uartSendChar('Z');
    uartSendChar('o');
    uartSendChar('n');
    uartSendChar('e');
    uartSendNumber(index + 1);
    uartSendChar(':');
    uartSendChar(' ');
    uartSendChar('H');
    uartSendChar('=');
    uartSendNumber(humidity);
    uartSendChar("%");
    uartSendChar(' ');
    uartSendChar('F');
    uartSendChar('=');
    uartSendNumber(flowRate);
    uartSendChar(' ');
    uartSendChar('L');
    uartSendChar('/');
    uartSendChar("m");
    uartSendChar('i');
    uartSendChar('n');
    uartSendChar(' ');
    uartSendChar('A');
    uartSendChar('=');
    uartSendNumber(isActive);
    uartSendString("");
}

void handleCommand(void) {
    // Команда запроса данных
    if(strcmp(message, "data;") == 0) {
        uartSendString("---Data---");
        // Отправляем данные по каждой зоне
        for(uint8_t i = 0; i < NUM_ZONES; i++) {
            sendZoneData(i);
        }
        uartSendString("---End---");
        return;
    }

    // Обработка команды установки времени
    int hour, min, sec;
    if(sscanf(message, "time:%d:%d:%d;", &hour, &min, &sec) ==
3) {
        if(hour >= 0 && hour < 24 && min >= 0 && min < 60 && sec
>= 0 && sec < 60) {
            uint8_t sreg = SREG;
            cli();
            setTime(&systemTime, hour, min, sec);
            SREG = sreg;
            uartSendString("OK");
            return;
```

```
        }
    }

    // Обработка команд для зон
    int zone;
    char cmd[10];
    if(sscanf(message, "zone%d:%[^;];", &zone, cmd) == 2 &&
       zone >= 1 && zone <= NUM_ZONES) {

        zone--; // Преобразуем в индекс массива
        uint8_t sreg = SREG;

        if(strcmp(cmd, "on") == 0) {
            cli();
            zones[zone].isManual = 1;
            zones[zone].isActive = 1;
            PORTB |= (1 << zone);
            SREG = sreg;
            uartSendString("ON");
            return;
        }
        else if(strcmp(cmd, "off") == 0) {
            cli();
            zones[zone].isManual = 0;
            zones[zone].isActive = 0;
            zones[zone].timeRemaining = 0;
            PORTB &= ~(1 << zone);
            SREG = sreg;
            uartSendString("OFF");
            return;
        }else if(strncmp(cmd, "flow:", 5) == 0) {
            // Parse flow rate
            int flowRate;
            if(sscanf(cmd + 5, "%d", &flowRate) == 1 && flowRate
>=1 && flowRate <= 20) {
                uint8_t sreg = SREG;
                cli();
                zones[zone].flowRate = flowRate;
                SREG = sreg;
                uartSendString("OK");
                return;
            }
        }
        else if(strncmp(cmd, "start:", 6) == 0) {
            // Parse start time
            int startHour, startMin;
            if(sscanf(cmd + 6, "%d:%d", &startHour, &startMin)
== 2 &&
               startHour >= 0 && startHour < 24 && startMin >= 0
&& startMin < 60) {
                uint8_t sreg = SREG;
                cli();
                zones[zone].startHour = startHour;
```

```
                zones[zone].startMinute = startMin;
                SREG = sreg;
                uartSendString("OK");
                return;
            }
        }else if(strncmp(cmd, "time:", 5) == 0) {
            // Parse start time
            int watering_time;
            if(sscanf(cmd + 5, "%d", &watering_time) == 1 &&
               watering_time >= 0 && watering_time < 360) {
                uint8_t sreg = SREG;
                cli();
                zones[zone].wateringTime = watering_time;
                SREG = sreg;
                uartSendString("OK");
                return;
            }
        }
    }

    uartSendString("ERROR");
}

ISR(USART_RX_vect) {
    char received = UDR;

    if(received == '\n') {
        message[messageIdx] = '\0';
        commandReady = 1;   // Устанавливаем флаг готовности
команды
        messageIdx = 0;
    }
    else if(received != '\r' && messageIdx < sizeof(message) -
1) {
        message[messageIdx++] = received;
    }
}

void initUart(void) {
    uint16_t baudRateForUbbr = (F_CPU / (BAUD_RATE * 16UL)) - 1;

    UBRRH = (unsigned char)(baudRateForUbbr >> 8);
    UBRRL = (unsigned char)baudRateForUbbr;

    UCSRB = (1 << RXEN) | (1 << TXEN) | (1 << RXCIE);
    UCSRC = (1 << URSEL) | (1 << UCSZ1) | (1 << UCSZ0);

    uartSendString("UART OK");
}
```

Листинг 4 – Заголовочный файл dht.h

```
#pragma once
#include "zone.h"

void initDht(void);
void processDhtData(Zone* zone, uint8_t sensorNum);
```

Листинг 5 – Модуль dht.c

```
#include "dht.h"
#include <avr/io.h>
#include <util/delay.h>

static uint8_t dhtBits[5];
static uint8_t sensorError[2] = {0, 0};

void initDht(void) {
    // Настройка соотв пинов на вход
    DDRD &= ~((1 << PD3) | (1 << PD4));
    // Включаем подтягивающие резисторы
    PORTD |= (1 << PD3) | (1 << PD4);
}

static uint8_t readDht(uint8_t pin) {
    uint8_t retries = 0;
    uint8_t pinMask = (1 << pin);
    uint8_t i, j;

    // Ресетим флаги
    for(i = 0; i < 5; i++) {
        dhtBits[i] = 0;
    }

    DDRD |= pinMask;            // Выход
    PORTD &= ~pinMask;          // Low
    _delay_ms(20);                // DHT22 требуется как минимум
10мс

    PORTD |= pinMask;           // High
    DDRD &= ~pinMask;           // Input
    _delay_us(40);

    // Проверяем начальные условия
    if(PIND & pinMask) return 0;
    _delay_us(80);
    if(!(PIND & pinMask)) return 0;
    _delay_us(80);

    // Считываем 40 бит
    for(i = 0; i < 40; i++) {
        // Ждем отрицательного фронта
```

```
        retries = 0;
        while(PIND & pinMask) {
            _delay_us(1);
            if(++retries > 100) return 0;
        }

        // Измеряем время до положительного фронта
        retries = 0;
        while(!(PIND & pinMask)) {
            _delay_us(1);
            if(++retries > 100) return 0;
        }

        // Wait 40us и проверяем значение бита
        _delay_us(30);
        j = i/8;
        dhtBits[j] <<= 1;
        if(PIND & pinMask) {
            dhtBits[j] |= 1;
        }
    }

    // Проверка checksum
    uint8_t checksum = dhtBits[0] + dhtBits[1] + dhtBits[2] +
dhtBits[3];
    if(checksum != dhtBits[4]) return 0;

    return 1;
}

void processDhtData(Zone* zone, uint8_t sensorNum) {
    static uint16_t readDelay = 0;

    if(readDelay < 2000) { // Ждем по 2 секунды между чтением
        readDelay++;
        return;
    }
    readDelay = 0;

    uint8_t pin = sensorNum ? PD4 : PD3;

    if(readDht(pin)) {
        // DHT22 возвращает влажность в формате: integral*10 +
decimal
        uint16_t humidity = (dhtBits[0] << 8) + dhtBits[1];
        zone->humidity = humidity / 10;
        sensorError[sensorNum] = 0;
    } else {
        if(++sensorError[sensorNum] >= 3) {
            zone->humidity = 0;
        }
    }
}
```

Листинг 6 – Заголовочный файл lcd.h

```
#pragma once
#include "zone.h"

void initLcd(void);
void lcd_string(const char *str);
void lcd_goto(uint8_t row, uint8_t col);
void displayError(const char* message);
void updateTimeDisplay(SystemTime* time);

// Функция отображения
void updateDisplay(Zone* zones, uint8_t currentZone, uint8_t
selectedParam);
```

Листинг 7 — модуль lcd.c

```
#include "lcd.h"
#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>

#define LCD_DATA_PORT PORTA
#define LCD_DATA_DDR  DDRA
#define LCD_CTRL_PORT PORTC
#define LCD_CTRL_DDR  DDRC

#define LCD_RS 0
#define LCD_RW 1
#define LCD_EN 2

static void lcd_cmd(uint8_t cmd) {
    LCD_CTRL_PORT &= ~((1<<LCD_RS)|(1<<LCD_RW));
    LCD_DATA_PORT = cmd;
    LCD_CTRL_PORT |= (1<<LCD_EN);
    _delay_us(1);
    LCD_CTRL_PORT &= ~(1<<LCD_EN);
    _delay_ms(2);
}

static void lcd_data(uint8_t data) {
    LCD_CTRL_PORT |= (1<<LCD_RS);
    LCD_CTRL_PORT &= ~(1<<LCD_RW);
    LCD_DATA_PORT = data;
    LCD_CTRL_PORT |= (1<<LCD_EN);
    _delay_us(1);
    LCD_CTRL_PORT &= ~(1<<LCD_EN);
    _delay_us(50);
}

void initLcd(void) {
    LCD_DATA_DDR = 0xFF;
```

```
    LCD_CTRL_DDR |= (1<<LCD_RS)|(1<<LCD_RW)|(1<<LCD_EN);

    _delay_ms(20);
    lcd_cmd(0x38);
    _delay_ms(5);
    lcd_cmd(0x0C);
    lcd_cmd(0x06);
    lcd_cmd(0x01);
    _delay_ms(2);
}

void lcd_string(const char *str) {
    while(*str) lcd_data(*str++);
}

void lcd_goto(uint8_t row, uint8_t col) {
    lcd_cmd(0x80 | (row * 0x40 + col));
}

void updateTimeDisplay(SystemTime* time) {
    char buf[16];
    lcd_cmd(0x01);  // Очищаем дисплей

    if(time->isSettingTime) {
        lcd_string("Set Time:");
    } else {
        lcd_string("Time:");
    }

    sprintf(buf, "%02d:%02d:%02d", time->hours, time->minutes,
time->seconds);
    lcd_goto(0, 6);
    lcd_string(buf);
}

void updateDisplay(Zone* zones, uint8_t currentZone, uint8_t
selectedParam) {
    char buf[16];
    lcd_cmd(0x01);

    sprintf(buf, "Zone%d %02d:%02d/%d%%",
            currentZone + 1,
            zones[currentZone].startHour,
            zones[currentZone].startMinute,
            zones[currentZone].humidity);
    lcd_string(buf);

    lcd_goto(1, 0);
    switch(selectedParam) {
        case PARAM_SCHEDULE:
            sprintf(buf, "Start: %02d:%02d",
                    zones[currentZone].startHour,
                    zones[currentZone].startMinute);
```

```
            break;

        case PARAM_HUMIDITY:
            sprintf(buf, "Target Hum: %d%%",
                    zones[currentZone].targetHumidity);
            break;

        case PARAM_TIME:
            sprintf(buf, "Duration: %dm",
                    zones[currentZone].wateringTime);
            break;

        case PARAM_FLOW:
            sprintf(buf, "Flow: %d L/min",
                    zones[currentZone].flowRate);
            break;
    }
    lcd_string(buf);
}

void displayError(const char* message) {
    lcd_cmd(0x01);
    lcd_string("Error:");
    lcd_goto(1, 0);
    lcd_string(message);
}
```

Листинг 11 – Заголовочный файл workers.h

```
#pragma once
#include "zone.h"

void checkLeaks(Zone* zones);
void playErrorTone();
```

Листинг 12 – Модуль workers.c

```
#include "zone_control.h"
#include "lcd.h"
#include "uart.h"
#include <avr/io.h>
#include <util/delay.h>

#define FLOW_CHECK_MS 10000
#define LEAK_THRESHOLD 20    // 20% отклонение
#define SOUND_PORT PORTE
// Пин вывода звукового сигнала
#define SOUND_PIN PE0
#define DELAY_MS_VAL 1500

static uint8_t leakStates = 0;
```

```
void playErrorTone() {
    for (int i = 0; i < DELAY_MS_VAL; i++) {
        SOUND_PORT |= (1 << SOUND_PIN);
        _delay_us(DELAY_MS_VAL / 2);
        SOUND_PORT &= ~(1 << SOUND_PIN);
        _delay_us(DELAY_MS_VAL / 2);
    }
}

void checkLeaks(Zone* zones) {
    // Читаем состояния кнопок утечки (активный уровень - низкий)
    uint8_t currentPB3 = (PINB & (1 << PB3)) == 0;
    uint8_t currentPB4 = (PINB & (1 << PB4)) == 0;

    // Проверяем каждую зону
    for(uint8_t i = 0; i < NUM_ZONES; i++) {
        uint8_t hasLeak = (i == 0) ? currentPB3 : currentPB4;

        // Обрабатываем утечку только если зона активна
        if(zones[i].isActive) {
            if(hasLeak) {
                if(!(leakStates & (1 << i))) {
                    leakStates |= (1 << i);
                    PORTD |= (1 << PD7);    // LED
                    playErrorTone();
                    char buf[16];
                    sprintf(buf, "Leak in Zone %d!", i + 1);
                    uartSendString(buf);
                    displayError(buf);
                }
            } else {
                leakStates &= ~(1 << i);
                if(!leakStates) {
                    PORTD &= ~(1 << PD7);
                    PORTE &= ~(1 << PE0);
                }
            }
        } else {
            // Если зона неактивна, сбрасываем её состояние
утечки
            leakStates &= ~(1 << i);
            if(!leakStates) {
                PORTD &= ~(1 << PD7);
                PORTE &= ~(1 << PE0);
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

Графическая часть

На 2 листах

Электрическая схема функциональная

Электрическая схема принципиальная

ПРИЛОЖЕНИЕ В

Спецификация радиоэлементов схемы

На 2 листах