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 09.03.03 **Прикладная информатика**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по дисциплине «Микропроцессорные системы»
на тему:

Контроллер теплицы

Студент	ИУ6-73Б		Д.О. Андреев
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Руководитель			И.Б. Трамов
		(Подпись, дата)	(И.О. Фамилия)

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 45 с., 25 рис., 4 табл., 6 источников, 2 прил.

МИКРОКОНТРОЛЛЕР, UART, ТАЙМЕР, Atmega8, SPI, КОНТРОЛЛЕР ТЕПЛИЦЫ.

Объектом разработки данной курсовой работы является контроллер теплицы.

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

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

В результате было спроектирована требуемая система и получена сопутствующая документация, а именно: функциональная и принципиальная схемы, схемы алгоритмов управления и соответствующая программа микроконтроллера.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ТЗ – техническое задание

МК – микроконтроллер

УГО – условное графическое обозначение

Atmega8 – используемый микроконтроллер

SPI – (Serial Peripheral Interface) последовательный периферийный интерфейс

Proteus ISIS – среда моделирования

UART – (англ. Universal Asynchronous Receiver-Transmitter) Универсальный асинхронный приемопередатчик

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Курсовая работа «Контроллер теплицы» выполнялся на основании учебного плана кафедры ИУ6.

Цель данной курсовой работы – на основе микроконтроллера AVR Atmega8 разработать МК-систему для работы в качестве контроллера теплицы. Проектирование контроллера теплицы состоит из двух основных частей: конструкторская часть и технологическая часть.

Конструкторская часть включает в себя:

– описание архитектуры, используемого микроконтроллера и описание назначения функциональных элементов схемы;

– описание принципиальной электрической схемы системы с обоснованием выбора используемых радиоэлементов;

– описание алгоритмов функционирования системы;

– расчет потребляемой мощности устройства.

Технологическая часть включает в себя:

– характеристику использованных систем разработки и отладки программ;

– тестирование и отладку программы;

– описание и моделирование работы системы;

– описание способа программирования МК.

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

1 Конструкторская часть

1.1 Анализ требований

Согласно ТЗ, необходимо разработать МК-систему для работы в качестве контроллера теплицы. Система должна реагировать на повышение уровня влажности и температуры открытием форточек. Закрытие форточек происходит в ситуации, когда температура и влажность внутри теплицы придут в норму. Необходимо обеспечить возможность открытия форточек в ручном режиме с помощью управляющих кнопок или команд с телефона. Необходимо фиксировать время открытия/закрытия форточек и отправлять его на ПЭВМ. Необходимо обеспечить периодическую передачу ключевых метрик на ПЭВМ и телефон оператора.

Основанием для выполнения данной работы являются:

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

Для реализации данного функционала устройство должно содержать следующие структурные блоки:

- микроконтроллер;
- кнопки;
- датчик температуры и влажности;
- блок управления двигателем (для открытия форточки);
- блок связи с ПЭВМ и телефоном.

1.2 Проектирование схемы электрической функциональной

1.2.1 Описание микроконтроллера Atmega8

Atmega8 – 8-разрядный микроконтроллер, основанный на усиленной AVR RISC архитектуре. Atmega8 обеспечивает производительность 1 млн. оп. в сек на 1 МГц синхронизации за счет выполнения большинства инструкций

за один машинный цикл и позволяет оптимизировать потребление энергии за счет изменения частоты синхронизации.

AVR ядро объединяет богатый набор инструкций с 32 рабочими регистрами общего назначения. Все 32 регистра непосредственно подключены к АЛУ (арифметико-логическое устройство), что позволяет указывать два регистра в одной инструкции и выполнить ее за один цикл.

Atmega8 обладает следующими возможностями: 8 Кбайт внутрисхемно программируемой флэш-памяти с возможностью чтения во время записи, 512 байт ЭППЗУ, 1 Кбайт статического ОЗУ, 23 линий ввода-вывода, 32 рабочих регистров общего назначения, два универсальных таймера-счетчика с режимами компаратора, внутренние и внешние запросы на прерывание, последовательный программируемый USART, программируемый сторожевой таймер с внутренним генератором, последовательный порт SPI. Режим холостого хода (Idle) останавливает ЦПУ, но оставляет в работе статическое ОЗУ, таймеры-счетчики, порт SPI и систему прерываний. Режим пониженного потребления (Power-down) сохраняет содержимое регистров, но останавливает генератор, выключает все встроенные функции до появления следующего запроса на прерывание или аппаратного сброса. В дежурном режиме генератор на кварцевом резонаторе запущен, а остальная часть отключена. Данный режим позволяет реализовать быстрый запуск в комбинации с малым потреблением.

Устройство выпускается по разработанной Atmel технологии энергонезависимой памяти высокой емкости. Встроенная ISP флэш-память может внутрисхемно перепрограммироваться через последовательный интерфейс SPI, обычным программатором энергонезависимой памяти или запущенной программой в секторе начальной загрузки AVR ядра. Структурная схема микроконтроллера ATMega8 представлена на рисунке 1.

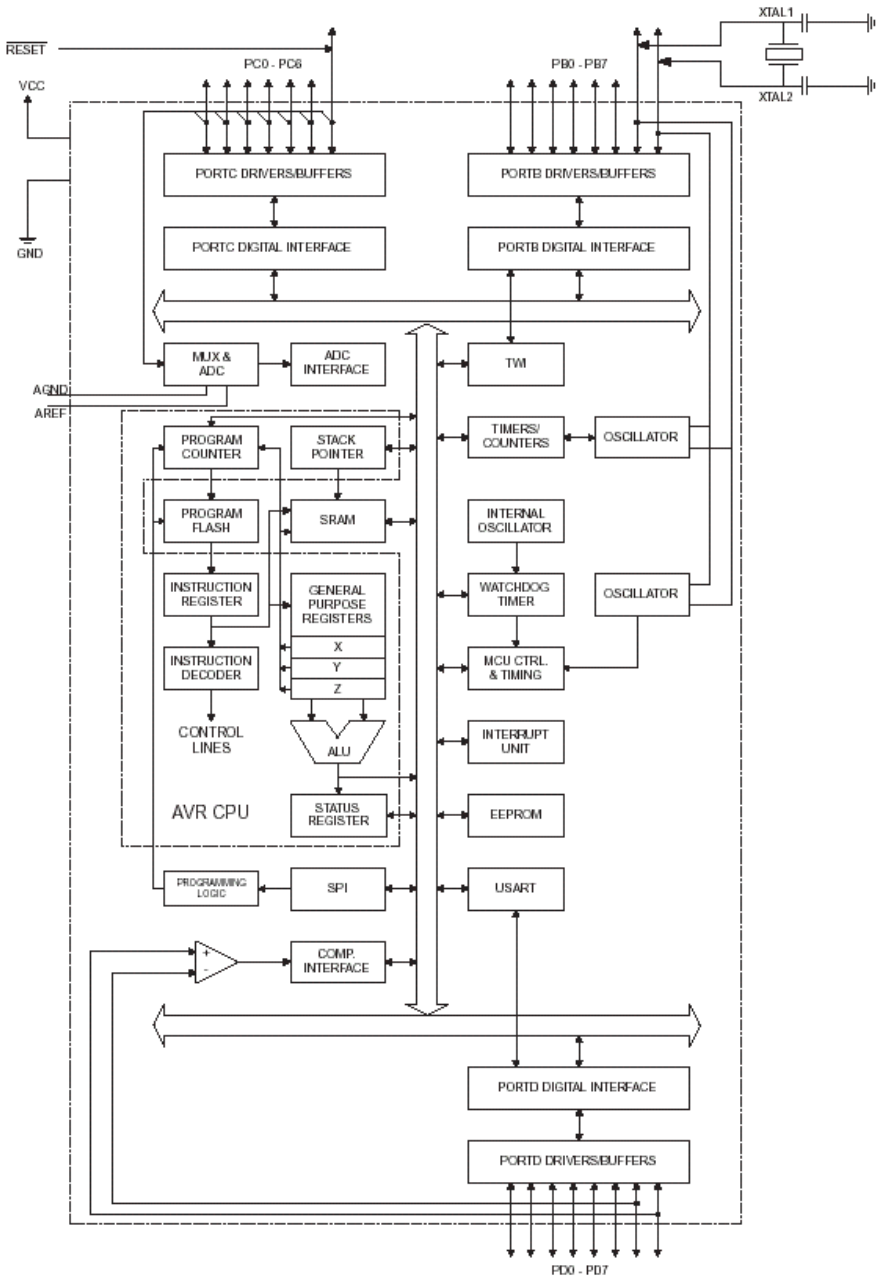

Рисунок 1 – Структурная схема микроконтроллера Atmega8

1.2.2 Организация памяти микроконтроллера Atmega8

Микроконтроллер Atmega8 имеет Гарвардскую архитектуру и содержит 3 вида памяти:

- память программ FLASH;
- оперативная память (ОЗУ) SRAM (Static RAM);
- энергонезависимая память данных EEPROM.

Карта адресного пространства представлена на рисунке 2.

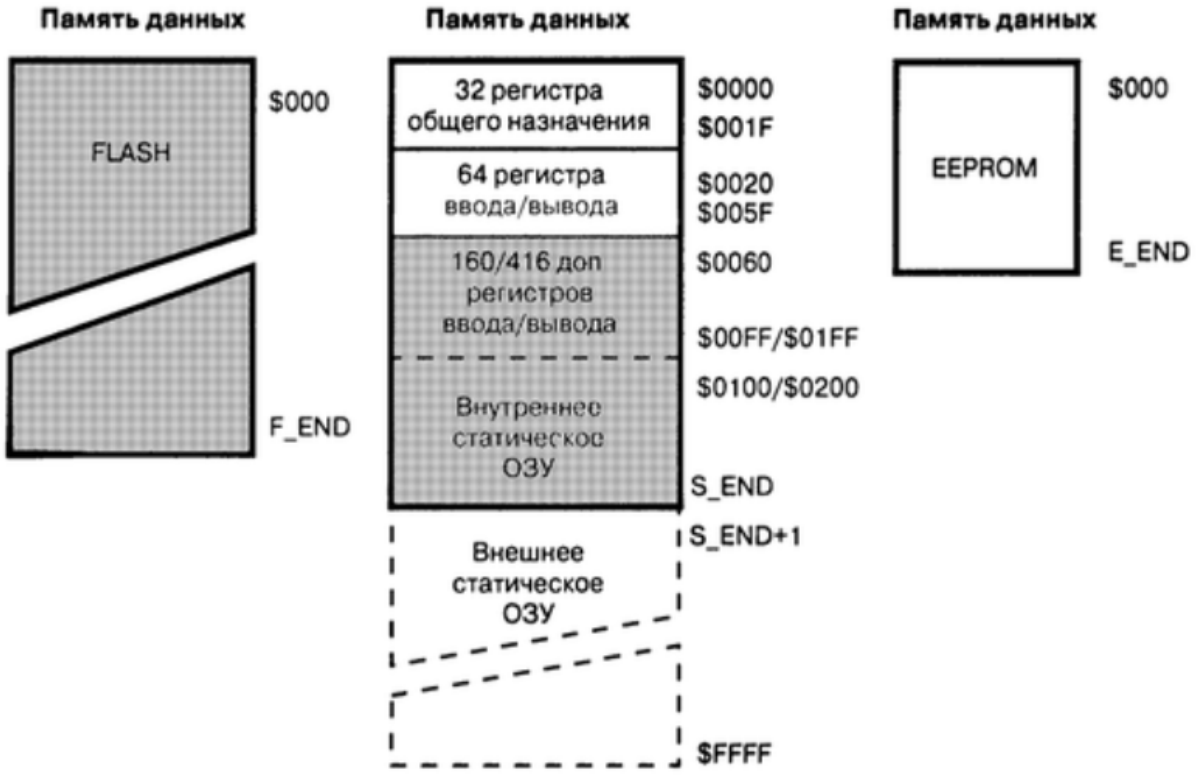

Рисунок 2 – Карта адресного пространства Atmega8

Адресные пространства указанных видов памяти, как правило, разделены. Способы адресации и доступа к этим областям памяти также различны. Такая структура позволяет центральному процессору работать одновременно как с памятью программ, так и с памятью данных, что существенно увеличивает производительность. Каждая из областей памяти данных (SRAM и EEPROM) также расположена в своем адресном пространстве.

Память программ представляет собой электрически стираемое ППЗУ (FLASH) и может поддерживать команды с разрядностью больше 8 бит. В Atmega8 память программ разделена на 2 секции:

– секцию загрузчика (Boot Program);

– секцию прикладных программ (Application Program).

Для адресации памяти программ используется счетчик команд (Program Counter – PC). В памяти программ также находится вектор сброса – в момент подачи питания микроконтроллер начинает выполнение программы с этого

адреса, и здесь размещается команда перехода к началу исполняемой программы. Кроме того, память программ содержит таблицу векторов прерываний. При возникновении прерывания после сохранения в стеке текущего значения счетчика команд происходит выполнение команды, расположенной по адресу соответствующего вектора. Поэтому по данным адресам располагаются команды перехода к подпрограммам обработки прерываний.

Положение вектора сброса и таблицы векторов прерываний может быть перенесено из секции прикладных программ в секцию загрузчика.

В некоторых случаях память программ может использоваться не только для хранения кода программы, но и для хранения различных констант.

Оперативная память содержит 3 области:

- регистры общего назначения;
- служебные регистры;
- память для хранения данных.

Регистры общего назначения (РОН) находятся в непосредственной близости к АЛУ. Применение набора регистров общего назначения в сочетании с конвейерной обработкой позволяет АЛУ выполнять одну операцию (извлечение операндов из набора регистров, выполнение команды и запись результата обратно в регистр) за один такт.

Служебные регистры имеют свои имя, адрес и назначение. Они предназначены для конфигурации и обслуживания периферийных узлов микроконтроллера. Остальная часть оперативной памяти предназначена для хранения пользовательских данных.

Энергонезависимая память данных (EEPROM) организована таким образом, что содержимое каждого байта отдельно может быть считано или записано. Количество циклов перезаписи энергонезависимой памяти превышает 100 тысяч. Энергонезависимая память предназначена для хранения настроек и конфигурации программы, то есть тех данных, которые должны сохраняться при пропадании питания.

1.3 Проектирование схемы электрической принципиальной

1.3.1 Конфигурация выводов микроконтроллера Atmega8

В разрабатываемом устройстве использован микроконтроллер фирмы Atmel – Atmega8 в прямоугольном пластиковом корпусе с 28 выводами. Конфигурация выводов микроконтроллера показана на рисунке 3.

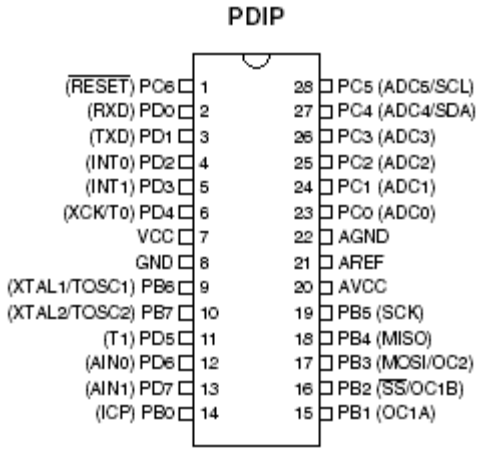

Рисунок 3 – Конфигурация выводов МК

1.3.2 Разработка схемы программирования МК

Для подключения программатора к МК-системе для прошивки МК необходима вилка на плату. Для этого была использована вилка IDC-06MS. Расположение выводов вилки представлено на рисунке 4.

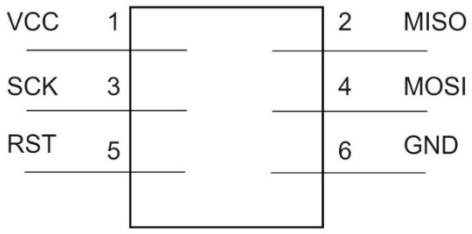

Рисунок 4 – Расположение выводов IDC-06MS

Вилка IDC-06MS используется для программирования микроконтроллера посредством интерфейса SPI.

SPI модуль микроконтроллера использует для своей работы 4 вывода - MOSI, MISO, SCK и SS. Когда модуль не задействован, эти выводы являются линиями портов ввода/вывода общего назначения. Назначение данных выводов описано ниже.

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

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

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

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

1.3.3 Датчик влажности и температуры

Для определения влажности и температуры необходимы датчики. Были рассмотрены три датчика: DS18B20, HIH-5030 и DHT11. Сравнительная характеристика датчиков представлена в таблице 1.

Таблица 1 – Сравнительная характеристика

Датчик	Измерение влажности	Измерение температуры	Интерфейс	Время отклика, с	Напряжение питания, В
DS18B20	-	+	1-Wire	0,75	3-5,5
HIH-5030	+	-	Аналоговый выход	5	2,7-5,5
DHT11	+	+	1-Wire	1	3,5-5,5

Из таблицы 1 видно, что датчик DHT11 является оптимальным выбором, потому что в отличии от двух других датчиков может изменять и влажность и

температуру. Также данный датчик обладает относительно небольшим временем отклика (1 секунда). Для определения влажности и температуры был выбран датчик DHT11. DHT11 – это цифровой датчик влажности и температуры, состоящий из термистора и емкостного датчика влажности.

Термистор – это термический резистор, сопротивление которого изменяется с температурой, т.е. увеличение температуры приводит к падению его сопротивления. Емкостной датчик влажности – это конденсатор с переменной емкостью, который содержит токопроводящие обкладки из медной фольги на текстолите. Этот конденсатор заключен в герметичный чехол, поверх которого расположен влагопоглощающий слой. При попадании частиц воды на этот слой, меняется его диэлектрическая проницаемость, что приводит к изменению емкости конденсатора. УГО датчика показано на рисунке 5.

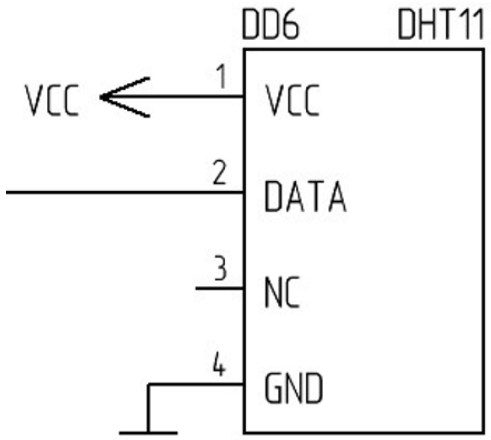

Рисунок 5 – УГО DHT11

Также датчик содержит в себе АЦП для преобразования аналоговых значений влажности и температуры. Датчик обладает следующими техническими характеристиками:

- питание: 3,5 – 5,5 В;
- ток питания: в режиме измерения 0,3 мА;
- определение влажности 20-80 % с точностью 5 %;
- определение температуры 0-80 °С с точностью 2 %.

Датчик имеет 4 вывода стандарта 2,54 мм (рисунок 6):

– 1 – VCC (питание 3–5 В);

– 2 – DATA (вывод данных);

– 3 – не используется;

– 4 – GND (земля).

Между выводами питания и вывода данных необходимо разместить резистор.

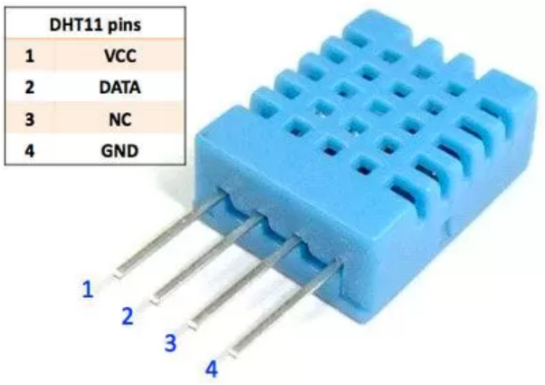

Рисунок 6 – Датчик DHT11

Для опроса датчика была написана процедура опроса датчика «dhtRead». Данная процедура показана в листинге 1.

Листинг 1 – Процедура опроса датчика

```
int dhtRead()
{
	for (uint8_t i = 0; i < 5; i++)
	{
		datadht[i] = 0;
	}

	//Шаг №1
	DHT_DDR    |=   (1<<DHT_BIT);
	DHT_PORT   &=~  (1<<DHT_BIT);
	_delay_ms (18);
	DHT_PORT   |=   (1<<DHT_BIT);
	_delay_us (40);

	//Шаг №2
	DHT_DDR &=~(1<<DHT_BIT);
	if (DHT_PIN&(1<<DHT_BIT))
	{
		return 0;
	}
	_delay_us (80);
	if (!(DHT_PIN&(1<<DHT_BIT)))
	{
```

```
            return 0;
      }

      //Шаг№3
      while (DHT_PIN&(1<<DHT_BIT));
      for (uint8_t j = 0; j < 5; j++)
      {
            datadht[j] = 0;

            for (uint8_t i = 0; i < 8; i++)
            {
                  cli();
                  while (!(DHT_PIN & (1<<DHT_BIT)));
                  _delay_us(30);
                  if (DHT_PIN & (1<<DHT_BIT))
                  datadht[j] |= 1 << (7 - i);
                  while (DHT_PIN & (1<<DHT_BIT));
                  sei();
                  }
            }
      return 1;
}
```

Опрос датчика осуществляется один раз в секунду (установка флага «askSensor» в прерывании таймера 1, который настроен на прерывание 1 раз в секунду) в процедуре «askDHT11». В данный процедуре происходит проверка флага «askSensor». Если флаг установлен, то вызывается процедура «dhtRead». После завершения опроса датчика флаг сбрасывается.

1.3.4 Разработка блока связи с ПЭВМ и телефоном

Связь с ПЭВМ была реализована посредством последовательного интерфейса UART через COM-порт компьютера с помощью драйвера MAX232.

UART (универсальный асинхронный приёмопередатчик) – технология передачи данных. Слово «асинхронный» означает, что интерфейс не использует линию для синхросигнала, приемник и передатчик заранее настраиваются на одну частоту.

Регистры USART:

– регистры данных UDR;

– UCSRA;

– UCSRB;

– UCSRC;

– UBRRL и UBRRH.

На рисунке 7 показана структура регистра UCSRA.

Регистр UCSRA

RXC	TxC	UDRE	FE	DOR	PE	U2X	MPCM
R	R/W	R	R	R	R	R	R

Рисунок 7 – Структура регистра UCSRA

Биты регистра UCSRA:

– RXC (этот бит (флаг) устанавливается, когда в приемном буфере есть непрочитанные данные, и очищается – когда буфер приема пуст);

– TXC (этот бит (флаг) устанавливается, когда все данные из сдвигового регистра были сдвинуты, а в буфере передачи (UDR) ещё нет новых данных);

– UDRE (указывает, готов ли буфер передачи (UDR) для записи в него новых данных);

– FE (бит устанавливается, если в очередном полученном пакете в приемном буфере обнаружена ошибка формата данных, то есть, когда первый стоп-бит равен нулю);

– DOR (бит устанавливается, если обнаружено переполнение приёмного буфера);

– PE (бит устанавливается, если в очередном полученном пакете в приемном буфере обнаружена ошибка чётности);

– U2X (при записи единицы в этот бит, делитель скорости передачи данных уменьшается с 16 до 8, таким образом, увеличивая вдвое скорость обмена в асинхронном режиме);

– MPCM (бит включает режим многопроцессорной связи).

На рисунке 8 показана структура регистра UCSRB.

Регистр UCSRB

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ	RXB8	TXB8
R	R/W	R/W	R/W	R/W	R/W	R	R/W

Рисунок 8 – Структура регистра UCSRB

Биты регистра UCSRB:

– RXCIE (включение прерывания по завершению приёма);

– TXCIE (включение прерывания по завершению отправки);

– UDRIE (включение прерывания по опустошению буфера данных);

– RXEN (запись единицы в этот бит включает приемник USART);

– TXEN (запись единицы в этот бит включает передатчик USART);

– UCSZ2 (устанавливает количество бит данных в пакете;

– RXB8 (в бит RXB8 записывается значение старшего бита (8) принятого пакета, при установленном размере посылки в 9 бит);

– TXB8 (в бит TXB8 записывается значение старшего бита (8) отправляемого пакета, при установленном размере посылки в 9 бит).

На рисунке 9 показана структура регистра UCSRC.

Регистр UCSRC

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	1	1	0

Рисунок 9 – Структура регистра UCSRC

Биты регистра UCSRC:

– URSEL (для записи в UCSRC – в бит URSEL должна быть записана единица, для записи в UBRRH – соответственно, ноль);

– UMSEL (установкой этого бита выбирается режим обмена: 0 – асинхронный, 1 – синхронный);

– UPM (установкой битов UPM1:0 выбирается режим проверки чётности);

– USBS (выбор количества стоп-бит);

– UCSZ (биты UCSZ1:0, совместно с битом UCSZ2 регистра UCSRB устанавливают количество бит данных в пакете (размер посылки) для приёмника и передатчика);

– UCPOL (этот бит используется только в синхронном режиме и определяет, по какому фронту тактирующего сигнала будет производиться отправка/приём данных, в асинхронном режиме – в бит UCPOL нужно записать ноль).

Регистры UBRRL и UBRRH отвечают за настройку скорости обмена. Передача данных в UART осуществляется по одному биту в равные промежутки времени. Этот временной промежуток определяется заданной скоростью UART и для конкретного соединения указывается в бодах, что соответствует количество бит в секунду. Существует общепринятый ряд стандартных скоростей: 4800, 9600, 19200 бод и т.д. Так как для UART используется для передачи небольшого количества данных на ПЭВМ (время открытия и закрытия форточек), то нет необходимости использовать большие скорости передачи данных, и при этом для удобства отладки программы лучше использовать не очень большую скорость передачи данных. Был принято решение использовать скорость 9600.

Настройка скорости обмена UART вычисляется по формуле 1:

$$UBRR = \frac{XTAL}{16 * baudrate} - 1 \quad (1)$$

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

$$UBRR = \frac{8\,000\,000}{16 * 9600} - 1 = 51$$

Таким образом в регистр UBBR записываем значение 51.

Для настройки UART была написана процедура инициализации «usartInit», которая показана в листинге 2.

Листинг 2 – Процедура инициализации UART

```
void usartInit()
{
	UBRRL=51;
	UCSRB=(1<<TXEN)|(1<<RXEN);
	UCSRC=(1<<URSEL)|(3<<UCSZ0);
	UCSRB |= (1<<RXCIE);
}
```

Для передачи символа через UART была реализована процедура «usartPutchar», которая показана в листинге 3.

Листинг 3 – Процедура передачи символа через UART

```
static int usartPutchar(char c, FILE *stream)
{
	if (c == '\n')
	usartPutchar('\r', stream);
	while(!(UCSRA & (1<<UDRE)));
	UDR = c;
	return 0;
}
```

Для передачи данных не телефон необходим Bluetooth модуль. Рассмотрим три модуля: HC-05, HC-06, HM-10. Сравнительная характеристика модулей представлена в таблице 2.

Таблица 2 – Сравнительная характеристика

Модуль	Версия Bluetooth	Переход в АТ-режим
HC-05	Bluetooth 2.0	Подача 1 на контакт
HC-06	Bluetooth 2.0	Включен по умолчанию
HM-10	4.0 BLE	Подача 0 на контакт

Из таблицы 2 видно, что модуль HC-06 находится в режиме АТ по умолчанию. Также данный модуль обладаем версией Bluetooth 2.0, что

позволяет взаимодействовать с этим модулем даже очень старыми мобильными устройствами. Поэтому связь с телефоном была реализована с помощью Bluetooth модуля HC-06. Модули HC-06 – это ведомые модули Bluetooth класса 2, создающие беспроводную передачу данных на коротком расстоянии между двумя системами или микроконтроллерами. HC-06 имеет четыре функциональных вывода, хотя некоторые модули HC-06 могут иметь до 6 выводов. УГО модуля HC-06 показано на рисунке 10.

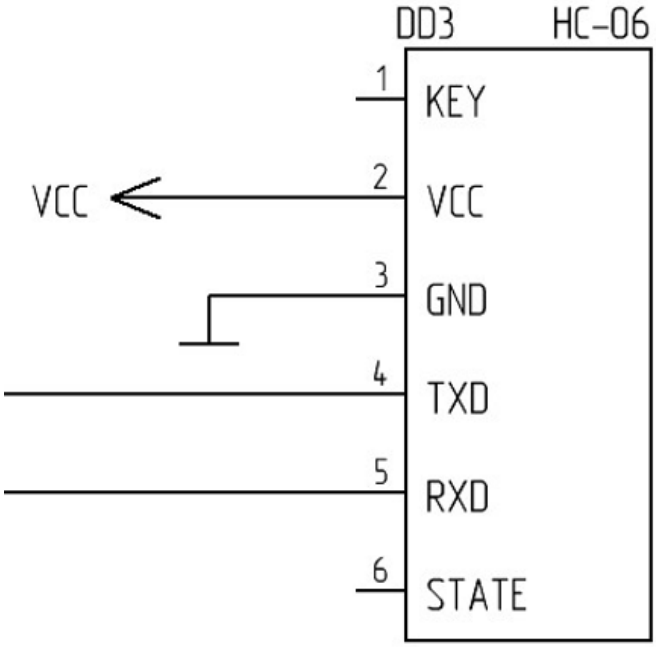

Рисунок 10 – УГО модуля HC-06

Связь с модулем HC-06 осуществляется через интерфейс UART. Технические характеристики:

- питание: 3,3В–6В;
- максимальное входное напряжение: 5В;
- максимальный ток: 45 мА;
- скорость передачи данных: 1200–1382400 бод;
- рабочие частоты: 2,40 ГГц – 2,48ГГц;
- поддержка спецификации bluetooth версии 2.1;
- дальность связи: 30 м.

Bluetooth модуль HC-06 показан на рисунке 11.

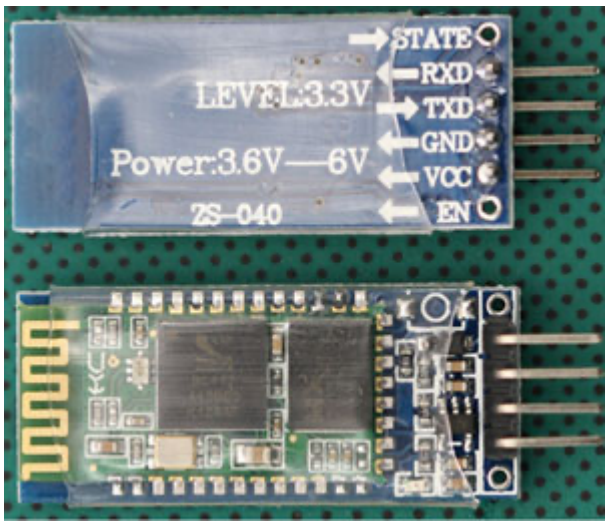

Рисунок 11 – Bluetooth модуль HC-06

Принципиальная схема модуля HC-06 показана на рисунке 12.

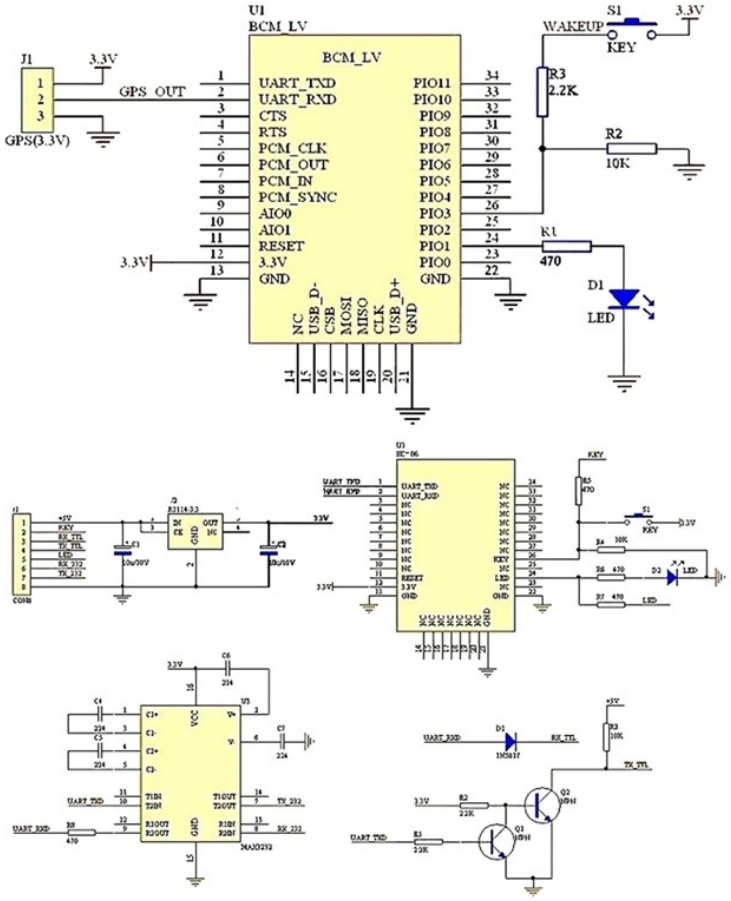

Рисунок 12 – Принципиальная схема модуля HC-06

Так как взаимодействие с модулем происходит с помощью UART, то используется те же процедуры и функции, которые используется для связи с ПЭВМ через UART. Для принятия команд с телефона была написана

процедура для принятия сообщения через UART, которая представлена в листинге 4.

Листинг 4 – Процедура принятия сообщения через UART

```
void receivingUsart()
{
	memset(data, 0, sizeof data);

	int i=0;
	do {
		while(!(UCSRA&(1<<RXC))) {};
		data[i]=UDR;
		i++;
	} while (data[i-1] != '\r');
}
```

1.3.5 Таймеры и прерывания

В данной программе используется 16-битный таймер Timer1 микроконтроллера ATMega8. Основная задача таймера - обеспечение периодических прерываний с частотой 1 Гц для отслеживания времени и регулярного выполнения задач.

Режим работы таймера

Таймер настроен на работу в режиме CTC (Clear Timer on Compare Match), что позволяет сбрасывать счетчик таймера при достижении заданного значения.

Настройка таймера

Настройка таймера включает в себя следующие шаги:

- Установка режима CTC: Регистр TCCR1B настраивается для работы в режиме CTC, что достигается установкой бита WGM12;
- Выбор предделителя: Для таймера устанавливается предделитель 256, что достигается установкой бита CS12 в регистре TCCR1B;

- Установка значения сравнения: В регистры OCR1AH и OCR1AL записывается значение 31250, что соответствует частоте прерываний 1 Гц при тактовой частоте 8 МГц;
- Разрешение прерываний: В регистре TIMSK устанавливается бит OCIE1A для разрешения прерываний по совпадению с регистром OCR1A;

Использование прерываний

Программа использует два типа прерываний: прерывания от USART и таймера Timer1.

- USART RX Complete Interrupt – Прерывание USART_RXC_vect активируется при завершении приема данных через USART;
- Timer1 Compare Match A Interrupt - Прерывание TIMER1_COMPA_vect срабатывает при достижении счетчиком таймера значения, установленного в регистре OCR1A. Это прерывание используется для отслеживания времени, регулярного опроса датчика и выполнения задач с заданной периодичностью.

Управление прерываниями

В программе используются команды sei() и cli() для глобального разрешения и запрещения прерываний соответственно. Это позволяет контролировать процесс выполнения критических участков кода, где необходимо избежать прерывания выполнения.

1.3.6 Схема управления двигателем

Для открытия и закрытия форточек будут использоваться шаговые двигатели. Шаговые двигатели – это такие двигатели, которые посредством

подачи напряжения на определённую обмотку переводят свой ротор в определённое место, тем самым достигается более точное управление угловой скоростью.

Так как двигатель имеет высокое потребление тока в сравнении с током, который может выдать контакт микроконтроллера, необходимо использовать мощные транзисторы или готовые микросхемы силовых каскадов. Для управления шаговыми двигателями существуют специальные микросхемы-драйвера. Один из таких драйверов – это микросхема ULN2003, которая представляет собой матрицу из 7 транзисторов Дарлингтона. Выходной ток такого драйвера равен 0,5 А.

Также существует полномостовой драйвер L298. Рабочий ток каждого канала равен 2 А (пиковый ток: 3 А). В сравнении с ULN2003 драйвер L298 может использоваться для управления более мощных двигателей. Поэтому был выбран драйвер L298, УГО которого представлено на рисунке 13.

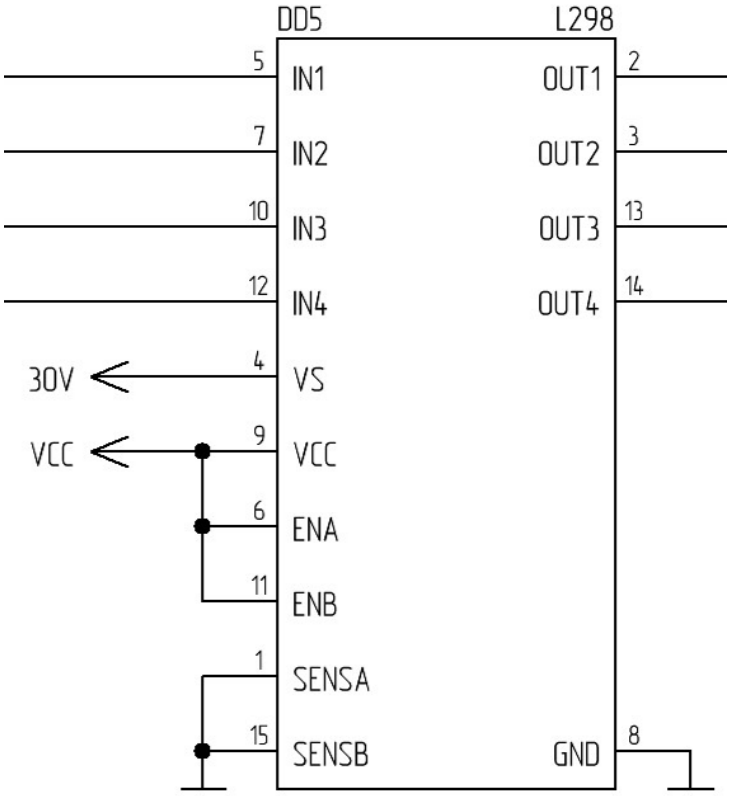

Рисунок 13 – УГО L298

Внутреннее устройство драйвера L298 показано на рисунке 14.

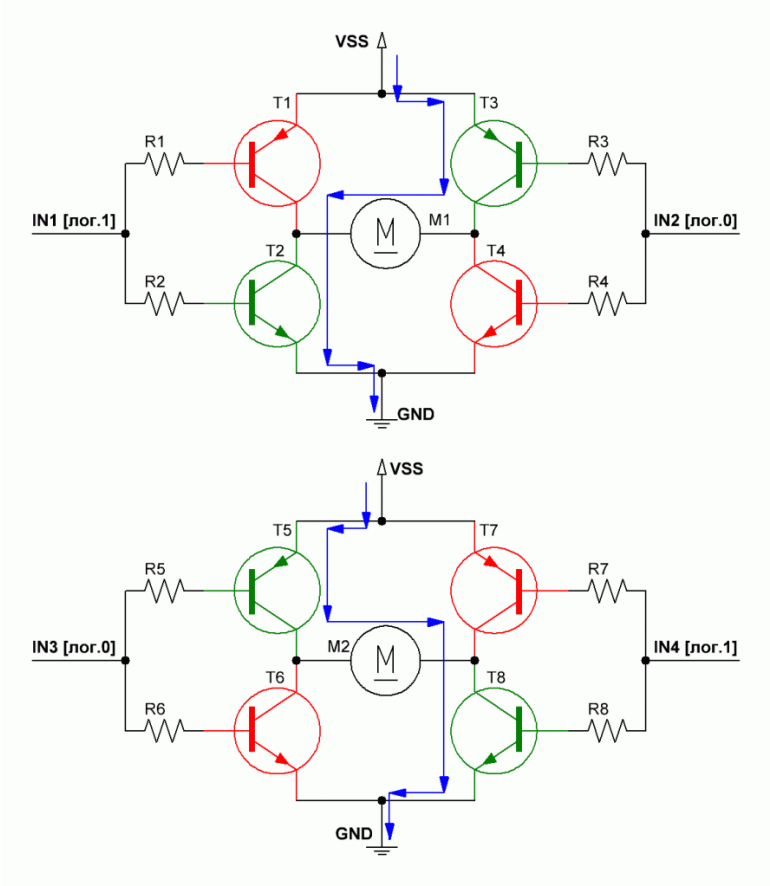

Рисунок 14 – Внутренне устройство драйвера L298

Для управления двигателем можно использовать только драйвер L298, либо можно дополнительно использовать контроллер шагового двигателя L297 (указано в документации драйвера L298 и L297). Микросхема L297 предназначена для использования с двойным мостовым драйвером, квадратной матрицей Дарлингтона или дискретными устройствами в системах управления шаговыми электродвигателями. Она получает сигналы синхронизации шагами двигателя, направления и режима работы от внешнего микроконтроллера и генерирует сигналы управления для силовых каскадов. Внутренне устройство микросхемы L297 показано на рисунке 15.

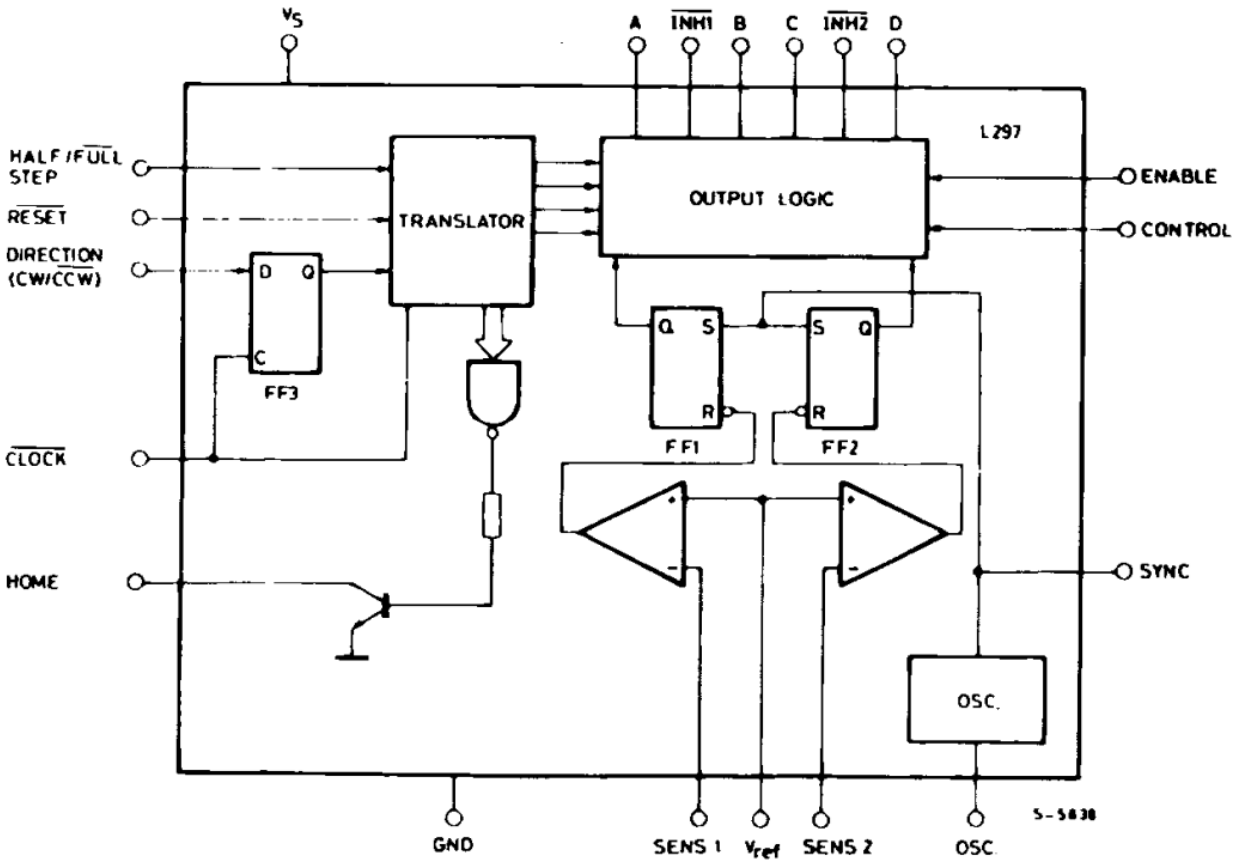

Рисунок 15 – Внутренне устройство микросхемы L297

Микросхема L298 – это силовой каскад, содержащий в своей структуре два H-моста, а L297 – это контроллер, генерирующий сигналы, управляющие микросхемой L298. УГО Микросхемы L297 показано на рисунке 16.

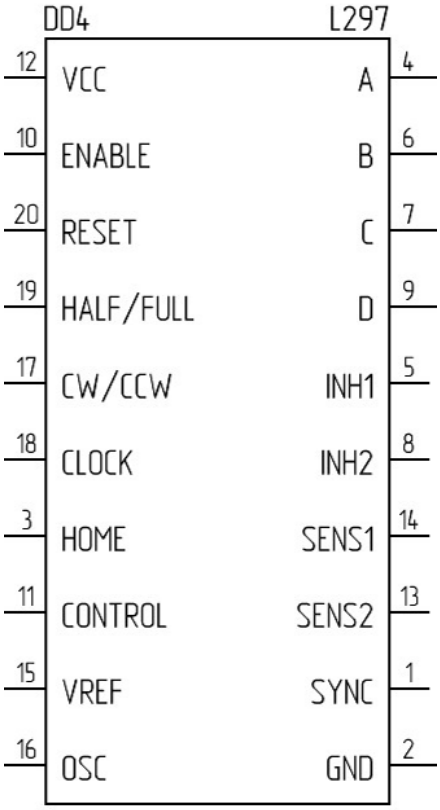

Рисунок 16 – Микросхема L297

Интегральный контроллер шагового двигателя L297 формирует четыре ведущих фазовых сигнала для двухфазных биполярных и четырехфазных униполярных шаговых электродвигателей в системах, управляемых микроконтроллерами, поэтому микроконтроллер можно подключать к контроллеру напрямую (микросхема L297 для этого предназначена). Двигатель может работать в полношаговом, полушаговом или колебательном режимах, а встроенный в контроллер L297 прерыватель широтно-импульсной модуляции (ШИМ) позволяет осуществлять импульсное управление током в обмотках шагового двигателя. Особенностью данного устройства является то, что ему необходимы, только входные сигналы синхронизации, направления и режима. На вход CW/CCW подаем направление вращения – 0 в одну сторону, 1 – в другую. На вход CLOCK – импульсы. Один импульс – один шаг. Вход HALF/FULL задает режим работы – полный шаг/полушаг.

Поскольку фазы генерируются внутри контроллера, программная нагрузка на микропроцессор значительно снижается. И управлять двигателем

становится возможно с помощью процедуры «moveDoor», которая представлена в листинге 5.

Листинг 5 – Процедура открытия/закрытия форточки

```
void moveDoor()
{
	if (doorStatus == DOOR_CLOSED)  PORTD |= (1<<PIND6);
	else                            PORTD &= ~(1<<PIND6);

	for (uint8_t i = 0; i < 100; i++)
	{
		_delay_ms(10);
		PORTD ^= (1<<PIND7);
	}
}
```

1.3.7 Расчет потребляемой мощности

Для определения потребляемой мощности МК воспользуемся формулой 2:

$$P_{\text{МК}} = I_{CC} * U_{VCC} * N \quad (2)$$

Где, I_{CC} – ток на выводах МК равный 10 мА, U_{VCC} – напряжение питания равное 5В, N – число задействованных выводов МК равное 16. Рассчитаем потребляемую мощность МК: $P_{\text{МК}} = 0{,}01 * 5 * 16 = 0{,}8$ Вт.

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

Таблица 3 – Потребляемая мощность

	Потребляемая мощность P , Вт	Количество микросхем	Суммарная потребляемая мощность $\sum P$, Вт
MAX232	0,05	1	0,05
HC-06	0,225	1	0,225
L297	0,4	1	0,4
L298	0,18	1	0,18
DHT11	0,0015	1	0,0015

Суммарная мощность всей системы равна:

$$\sum P_C = 0{,}8 + 0{,}05 + 0{,}225 + 0{,}4 + 0{,}18 + 0{,}0015 = 1{,}6565 \text{ Вт}$$

1.4 Разработка алгоритмов основных программных модулей

1.4.1 Главная процедура

При запуске системы производится выполнение главной процедуры программы, где сначала происходит инициализация всей необходимой переферии, а также разрешение глобальных прерываний. После в бесконечном цикле while выполняются основные процедуры разрабатываемой программы. Схема алгоритма главной процедуры программы изображена на рисунке 17.

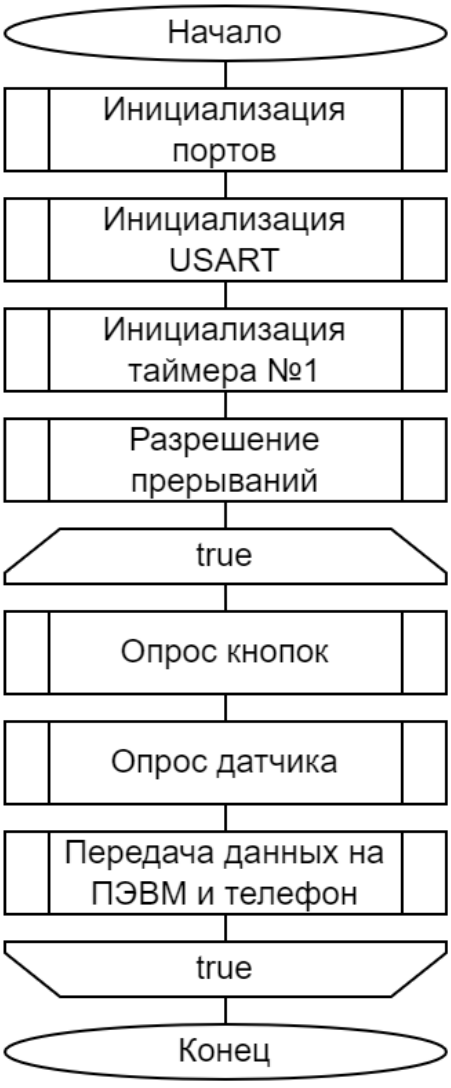

Рисунок 17 – Схема алгоритма главной процедуры

В бесконечном цикле while выполняются последовательно следующие процедуры: опрос кнопок, опрос датчика, передача данных (логов) на ПЭВМ и телефон.

1.4.2 Процедура опроса кнопок

В процедуре «checkingButtons» происходит последовательная проверка нажатий на кнопки. В зависимости от того, какая кнопка была нажата, происходят соответсвующие события. При нажатие на кноку «OPEM» происходит открытие форточки. При нажатие на кноку «CLOSE» происходит закрытие форточки. При нажатие на кнопку «MODE» происходит переключение режима управления форточкой. Схема алгоритма процедуры опроса кнопок изображена на рисунке 18.

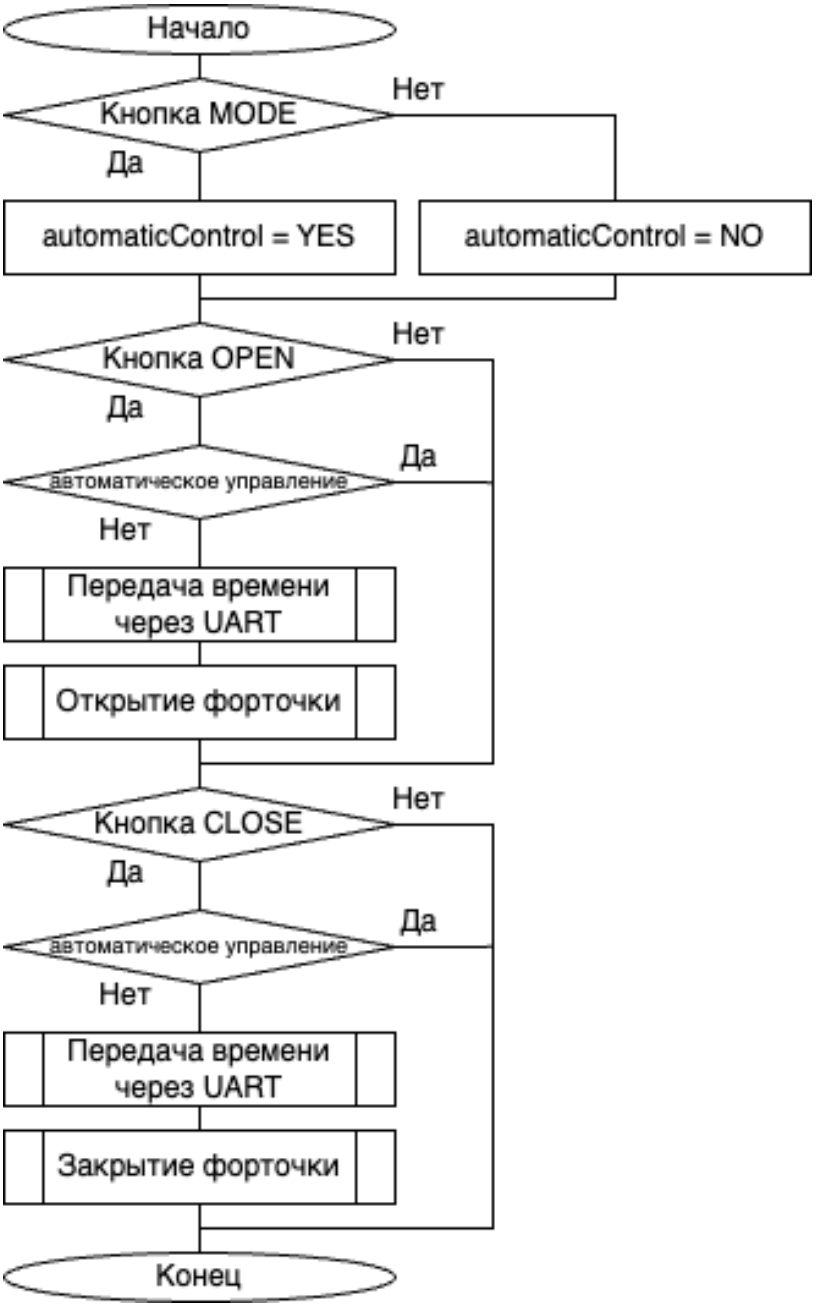

Рисунок 18 – Схема алгоритма процедуры опроса кнопок

1.4.3 Процедура опроса датчика

В процедуре «askDHT11» происходит опрос датчика и сравнение полученных показаний с граничными значениями. Если текущие показания превыщают граничные значения, то происходит автоматическое открытие форточки. Если текущие значения меньше граничных, то происходит автоматическое закрытие форочки. Схема алгоритма процедуры опроса датчика изображена на рисунке 19.

Рисунок 19 – Схема алгоритма процедуры опроса датчика

2 Технологическая часть

2.1 Характеристика использованных систем для разработки и отладки программ

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

– Atmel Studio – интегрированная среда разработки программного обеспечения для микроконтроллеров семейства AVR фирмы Atmel.

– Proteus 8 – для симуляции и тестирования проекта.

2.2 Тестирование устройства в симуляторе Proteus

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

– установка системного времени;

– управление форточкой;

– вывод данных в ПЭВМ и телефон.

На рисунке 20 изображена схема устройства в симуляторе Proteus.

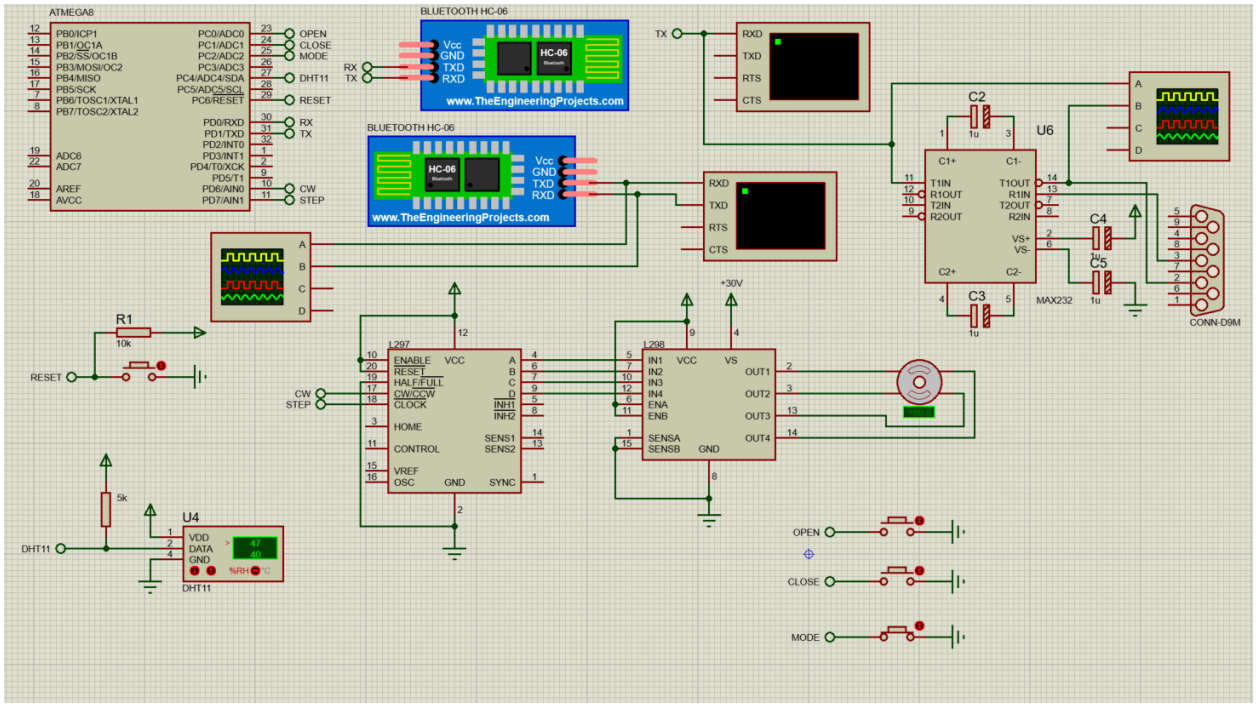

Рисунок 20 – Схема устройства в симуляторе Proteus

На рисунке 21 показана осциллограма сигнала до и после драйвера MAX232. На рисунке 22 показана осциллограма передачи данных через Bluetooth модуль. На рисунке 23 показана установка текущего времени. На рисунке 24 показана настройка граничных значений. На рисунке 25 показано открытие и закрытие форточки.

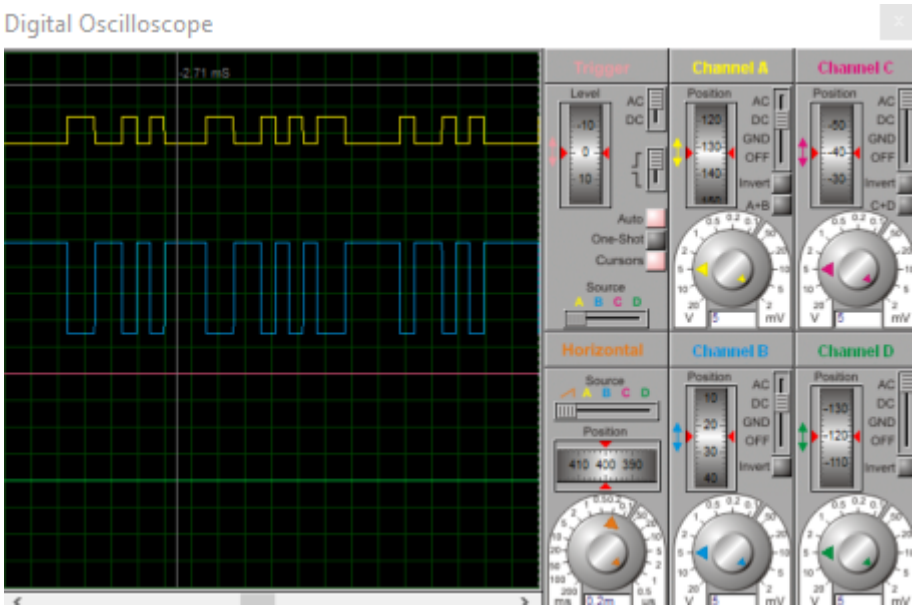

Рисунок 21 – Осциллограма сигнала до и после драйвера MAX232

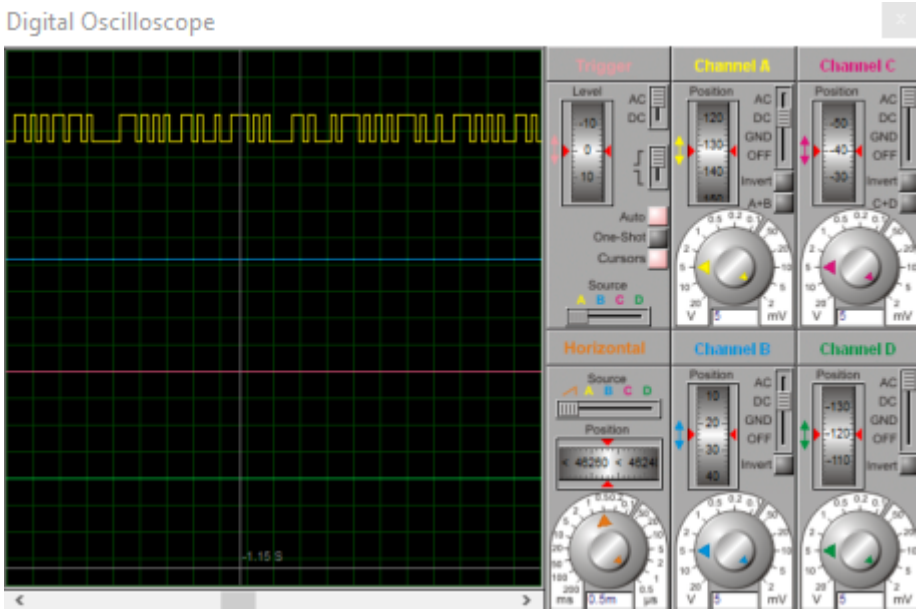

Рисунок 22 – Осциллограма передачи данных через Bluetooth модуль

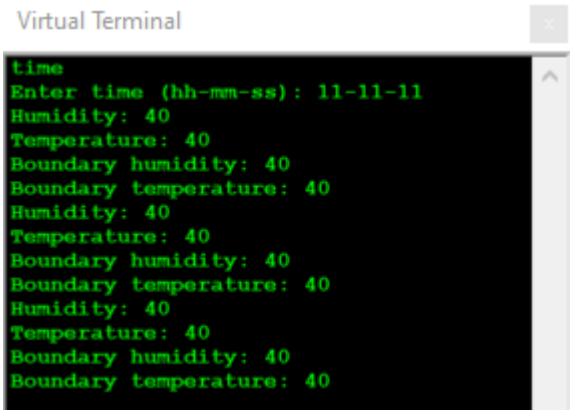

Рисунок 23 – Установка системного времени

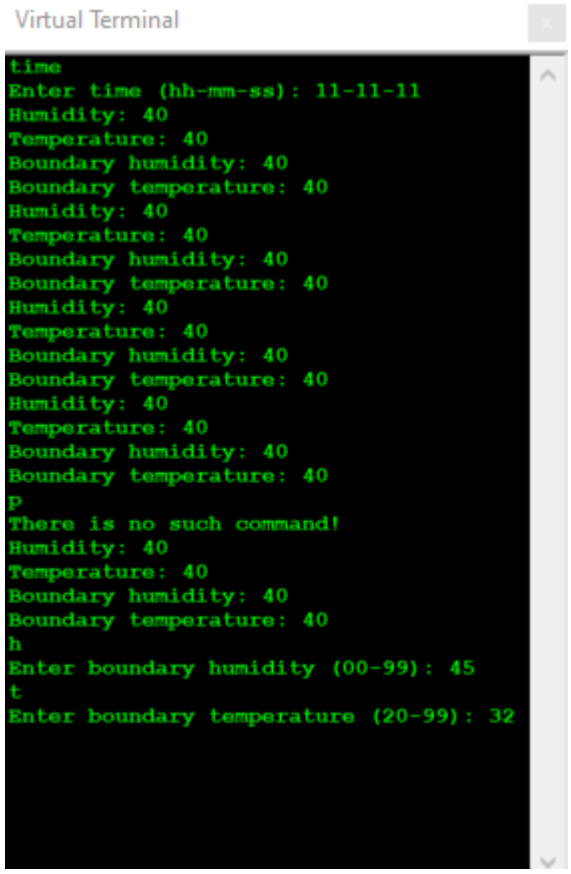

Рисунок 24 – Настройка граничных значений

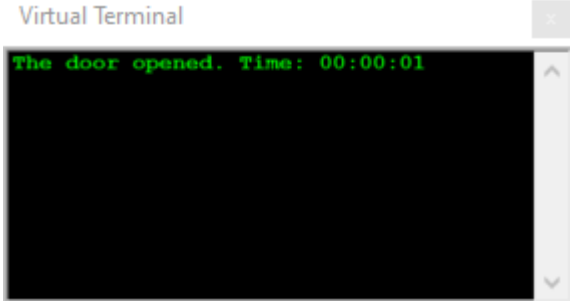

Рисунок 25 – Открытие форточки

2.3 Программирования микроконтроллера

AVR-микроконтроллеры предоставляют пользователю несколько различных интерфейсов для программирования. Это последовательное программирование при высоком напряжении, последовательное программирование при низком напряжении через SPI, параллельное программирование при высоком напряжении и программирование по интерфейсу JTAG. Первый тип программирования встречается только в моделях AVR семейства ATtiny, последний – доступен некоторым моделям старшего семейства. Модели Atmega с наиболее развитой периферией могут поддерживать до трех различных интерфейсов программирования.

Низковольтное последовательное программирование через SPI, наиболее распространено. Это способ стоит признать основным при программировании AVR-микроконтроллеров. Его поддерживают все модели с ядром AVR, за исключением двух устаревших представителей младшего семейства ATtiny11x и ATtiny28x.

Взаимодействие устройств по интерфейсу SPI требует установки одного из устройств в режим ведущего, а остальных – в режим ведомого. При этом ведущее устройство отвечает за выбор ведомого и инициализацию передачи. При программировании AVR программатор всегда функционирует как ведущее устройство, а микроконтроллер как ведомое. SPI является синхронным интерфейсом: все операции синхронизированы фронтами тактового сигнала (SCK), который вырабатывается ведущим устройством. Максимальная скорость передачи ограничена величиной 1/4 тактовой частоты контроллера. На минимальную скорость нет никаких ограничений: без тактового сигнала обмен данными «замораживается», и интерфейс может оставаться в статическом состоянии сколько угодно долго.

Программирование микроконтроллера по SPI осуществляется путем посылки 4-байтовых команд на вывод MOSI МК, в который один или два байта определяют тип операции, остальные – адрес, записываемый байт,

установочные биты и биты защиты, пустой байт. При выполнении операции чтения считываемый байт снимается через вывод MISO. Так же можно запрограммировать память данных EEPROM. В каждой команде указывается адрес записываемой ячейки и записываемое значение.

Форматы байтов команд для программирования микроконтроллера Atmega8 представлены в таблице 4.

Таблица 4 – Формат байтов команд для программирования Atmega8

Инструкция	Формат инструкции				Функция
	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	Стирание EEPROM и Flash
Чтение памяти программ	0010 H000	0000 aaaa	bbbb bbbb	oooo oooo	Чтение H старшего или младшего байта данных o из памяти программ по адресу a:b
Загрузка страницы памяти программ	0100 H000	0000 xxxx	xxxb bbbb	iiii iiii	Записывает H (1 или 0) и данные i в память программ по слову адреса b
Записать страницу памяти программ	0100 1100	0000 aaaa	bbbx xxxx	xxxx xxxx	Записывает страницу памяти программ по адресу a:b
Читать EEPROM память	1010 0000	00xx xxxa	bbbb bbbb	oooo oooo	Читает данные o из EEPROM памяти по адресу a:b
Записать в EEPROM память	1100 0000	00xx xxxa	bbbb bbbb	iiii iiii	Записывает данные i в EEPROM память по адресу a:b
Читать биты блокировки	0101 1000	0000 0000	xxxx xxxx	xxoo oooo	Читает биты блокировки.0 - запрограммирован, 1 - незапрограммирован
Записать биты блокировки	1010 1100	111x xxxx	xxxx xxxx	11ii iiii	Записывает биты блокировки. При программировании биты 1,2 = 0
Читать биты-предохранители	0101 0000	0000 0000	xxxx xxxx	oooo oooo	Читает биты-предохранители. 0 - запрограммирован, 1 - незапрограммирован
Записать биты-предохранители	1010 1100	1010 0000	xxxx xxxx	iiii iiii	Записывает биты-предохранители.Устанавливает биты 6, 4,3=0 для программирования битов, 1 для очистки битов
Читать байт сигнатуры	0011 0000	00xx xxxx	xxxx xxbb	oooo oooo	Читает байт сигнатуры o по адресу b

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

Последовательность подачи питания - подать напряжение питания между VCC и GND, когда на входах RESET и SCK установлен "0". В

некоторых системах, программатор не может гарантировать, что SCK = 0 при подаче питания. В этом случае необходимо сформировать положительный импульс на RESET длительностью не менее двух тактов ЦП после того, как SCK принял значение "0".

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

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

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

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

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

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

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

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был разработан контроллер теплицы на основе микроконтроллера Atmega8. Устройство предусматривает несколько ручной и автоматический режимы работы.

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

Разработанное устройство удовлетворяет требованиям, предъявленным в задании на курсовой проект, и может использоваться в качестве контроллера теплицы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

3 Документация на микроконтроллер Atmega8 [Электронный ресурс]. – URL: http://ww1.microchip.com/downloads/en/devicedoc/doc2512.pdf (дата обращения 11.10.2023).

4 ГОСТ 2.702-2011 Правила выполнения электрических схем

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

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

7 Шпак, Ю.А. Программирование на языке С для AVR и PIC микроконтроллеров, Издательство «КОРОНА-ВЕК», 2011. – 519с.

ПРИЛОЖЕНИЕ А. Текст исходной программы

```
#define F_CPU 8000000UL    //частота работы МК

#include <stdio.h>
#include <avr/io.h>
#include <string.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define DHT_PORT PORTC
#define DHT_DDR DDRC
#define DHT_PIN PINC
#define DHT_BIT 4

#define BUT_DDR            DDRC
#define BUT_PORT    PORTC
#define BUT_PIN            PINC
#define BUT_OPEN    0
#define BUT_CLOSE   1
#define BUT_MODE    2

#define DOOR_CLOSED 0
#define DOOR_OPEN   1

#define NO    0
#define YES   1

uint32_t timeSec = 0;
uint8_t boundaryTemperature = 40;
uint8_t boundaryHumidity = 40;
uint8_t doorStatus = DOOR_CLOSED;
uint8_t askSensor = NO;
uint8_t getInfoFromUsart = NO;
uint8_t automaticControl = YES;

uint8_t datadht[5];              //массив для значений датчика

const char CMD_SET_HUMIDITY[]     = "h\r";      //команда для установки порога влажности
const char CMD_SET_TEMPERATURE[]= "t\r"; //команда для установки порога температуры
const char CMD_SET_TIME[]         = "time\r";   //команда для установки системного времени
const char CMD_OPEN_DOOR[]        = "o\r";      //команда для открытия форточки
const char CMD_CLOSE_DOOR[]       = "c\r";      //команда для закрытия форточки
const char CMD_GET_INFO[]         = "g\r";      //команда для запроса показаний датчика
const char CMD_GET_TIME[]         = "gt\r";     //команда для запроса системного времени

char data[12]; //массив символов для передачи данных по usart
static int usartPutchar(char c, FILE *stream);
static FILE mystdout = FDEV_SETUP_STREAM(usartPutchar, NULL, _FDEV_SETUP_WRITE);

//функция опроса датчика
int dhtRead()
{
       for (uint8_t i = 0; i < 5; i++)
       {
              datadht[i] = 0;
       }

       //Шаг №1
       DHT_DDR              |=     (1<<DHT_BIT); //выход
       DHT_PORT      &=~    (1<<DHT_BIT); //низкий уровень – подтягиваем линию-разбудим
датчик
       _delay_ms (18);                                  //18 мс по условиям документации
```

```
	DHT_PORT	|=	(1<<DHT_BIT); //отпускаем линию
	_delay_us (40);                                 //задержка по условию

	//Шаг №2
	DHT_DDR &=~(1<<DHT_BIT);          // вход
	if (DHT_PIN&(1<<DHT_BIT))         //датчик должен ответить 0
	{
		return 0;
	}
	_delay_us (80);                                 // задержка
	if (!(DHT_PIN&(1<<DHT_BIT)))      //по истечению 80 мкс, датчик должен отпустить
шину
	{
		return 0;
	}

	//Шаг№3
	while (DHT_PIN&(1<<DHT_BIT)); //ждем пока контроллер датчика начнет передавать
данные
	//передача начинается с нуля
	for (uint8_t j = 0; j < 5; j++)
	{
		datadht[j] = 0;

		for (uint8_t i = 0; i < 8; i++)
		{
			cli();                                                  //запрещаем
прерывания
			while (!(DHT_PIN & (1<<DHT_BIT)));      //ждем когда датчик отпустит
шину
			_delay_us(30);                                          //задержка
высокого уровня при 0 30 мкс
			if (DHT_PIN & (1<<DHT_BIT))             //если по истечению времени
сигнал на линии высокий, значит передается 1
			datadht[j] |= 1 << (7 - i);             //тогда i-й бит
устанавливаем 1
			while (DHT_PIN & (1<<DHT_BIT));         //ждем окончание 1
			sei();                                                  //разрешаем
прерывание
		}
	}
	return 1;
}

//инициализация usart
void usartInit()
{
	UBRRL=51;                                               //8 000 000 / 9600 / 16 - 1
= 51
	UCSRB=(1<<TXEN)|(1<<RXEN);              //разрешаем прием-передау
	UCSRC=(1<<URSEL)|(3<<UCSZ0);            //8 бит
	UCSRB |= (1<<RXCIE);                    //Разрешаем прерывание при передаче

	stdout = &mystdout;
}

//передача потока char через usart
static int usartPutchar(char c, FILE *stream)
{
	if (c == '\n')
	usartPutchar('\r', stream);
	while(!(UCSRA & (1<<UDRE)));
	UDR = c;
	return 0;
```

```
}

//принятие данных через usart
void receivingUsart()
{
	memset(data, 0, sizeof data);

	int i=0;
	do {
		while(!(UCSRA&(1<<RXC))) {};
		data[i]=UDR;
		i++;
	} while (data[i-1] != '\r');
}

void setTemperatureUsart()
{
	printf("Enter boundary temperature (20-99): ");
	receivingUsart();
	boundaryTemperature =	(data[0]&0b00001111) * 10 +
						(data[1]&0b00001111);
}

void setHumidityUsart()
{
	printf("Enter boundary humidity (00-99): ");
	receivingUsart();
	boundaryHumidity =  (data[0]&0b00001111) * 10 +
					(data[1]&0b00001111);
}

void moveDoor()
{
	if (doorStatus == DOOR_CLOSED)	PORTD |= (1<<PIND6);
	else 					PORTD &= ~(1<<PIND6);

	for (uint8_t i = 0; i < 100; i++)
	{
		_delay_ms(10);
		PORTD ^= (1<<PIND7);
	}
}

void getTimeUsart()
{
	uint8_t seconds	= timeSec % 3600 % 60;
	uint8_t minutes = timeSec % 3600 / 60;
	uint8_t hours = timeSec / 3600;

	printf("Time: %d%d:%d%d:%d%d\n", hours / 10, hours % 10, minutes / 10, minutes %
10, seconds / 10, seconds % 10);
}

void openDoorUsart()
{
	if (doorStatus == DOOR_CLOSED)
	{
		if (automaticControl == NO)
		{
			printf("The door opened. ");
			getTimeUsart();
			moveDoor();
			doorStatus = DOOR_OPEN;
		}
```

```
			else printf("Turn off automatic control!\n");
			//moveDoor();
			//doorStatus = DOOR_OPEN;
		}
}

void closeDoorUsart()
{
		if (doorStatus == DOOR_OPEN)
		{
			if (automaticControl == NO)
			{
				printf("The door closed. ");
				getTimeUsart();
				moveDoor();
				doorStatus = DOOR_CLOSED;
			}
			else printf("Turn off automatic control!\n");
			//moveDoor();
			//doorStatus = DOOR_CLOSED;
		}
}

void getInfoUsart()
{
		printf("Humidity: %d\n", datadht[0]);
		//текущая влажность
		printf("Temperature: %d\n", datadht[2]);				//текущая
температура
		printf("Boundary humidity: %d\n", boundaryHumidity);		//пороговая влажность
		printf("Boundary temperature: %d\n", boundaryTemperature);	//пороговая
температура
}

void setTimeUsart()
{
		printf("Enter time (hh-mm-ss): ");
		receivingUsart();

		timeSec =	((data[0] & 0b00001111) * 10 * 3600L)	+
				((data[1] & 0b00001111) * 3600L)	+
				((data[3] & 0b00001111) * 10 * 60)		+
				((data[4] & 0b00001111) * 60)			+
				((data[6] & 0b00001111) * 10)			+
				(data[7] & 0b00001111);
}

//прерывание usart
ISR(USART_RXC_vect)
{
		receivingUsart();

		if (strcmp (data, CMD_SET_TEMPERATURE)==0)		setTemperatureUsart();
		else if (strcmp (data, CMD_SET_HUMIDITY)==0)	setHumidityUsart();
		else if (strcmp (data, CMD_SET_TIME)==0)		setTimeUsart();
		else if (strcmp (data, CMD_OPEN_DOOR)==0)		openDoorUsart();
		else if (strcmp (data, CMD_CLOSE_DOOR)==0)		closeDoorUsart();
		else if (strcmp (data, CMD_GET_INFO)==0)		getInfoUsart();
		else if (strcmp (data, CMD_GET_TIME)==0)		getTimeUsart();
		else
		printf("There is no such command!\n");
}

//инициализация timer1 на переполнение 1 раз в секунду
```

```
void timer1Init()
{
	TCCR1B |= (1<<WGM12);	//установка режима СТС (сброс по совпадению)
	TIMSK |= (1<<OCIE1A);	//установка бита разрешения прерывания 1-ого счетчика
по совпадению с OCR1A
	//частота прерывания timer1:
	//8_000_000 / 256 (делитель) = 31250
	//31250 / 31250 (регистр сравнения) = 1 Гц
	TCCR1B |= (1<<CS12); //установка делителя на 256
	OCR1AH = 0b01111010; //запись в регистр числа (31250) для сравнения
	OCR1AL = 0b00010010; //запись в регистр числа (31250) для сравнения
}

//перывание timer1
ISR(TIMER1_COMPA_vect)
{
	timeSec++;
	askSensor = YES;

	if (timeSec % 5 == 0)
	{
		getInfoFromUsart = YES;
	}

	if (timeSec == 86400)
	{
		timeSec = 0;
	}
}

//опрос кнопок
void checkingButtons()
{
	if (!(BUT_PIN&(0x01<<BUT_MODE)))
	{
		automaticControl = YES;
	}
	else
	{
		automaticControl = NO;
	}

	if (!(BUT_PIN&(0x01<<BUT_OPEN)))
	{
		while (!(BUT_PIN&(0x01<<BUT_OPEN)));

		if (doorStatus == DOOR_CLOSED)
		{
			if (automaticControl == NO)
			{
				printf("The door opened. ");
				getTimeUsart();
				moveDoor();
				doorStatus = DOOR_OPEN;
			}
			else printf("Turn off automatic control!\n");
		}
	}
	else if (!(BUT_PIN&(0x01<<BUT_CLOSE)))
	{
		while (!(BUT_PIN&(0x01<<BUT_CLOSE)));

		if (doorStatus == DOOR_OPEN)
		{
```

```
                        if (automaticControl == NO)
                        {
                                printf("The door closed. ");
                                getTimeUsart();
                                moveDoor();
                                doorStatus = DOOR_CLOSED;
                        }
                        else printf("Turn off automatic control!\n");
                }
        }
}

void askDHT11()
{
        if (askSensor == YES)
        {
                cli (); // запрещаем прерывания
                dhtRead (); //опрос датчика
                sei ();// разрешаем общее прерывание

                if ((datadht[0] > boundaryHumidity) || (datadht[2] > boundaryTemperature))
                {
                        if (doorStatus == DOOR_CLOSED)
                        {
                                if (automaticControl == YES)
                                {
                                        printf("The door opened. ");
                                        getTimeUsart();
                                        moveDoor();
                                        doorStatus = DOOR_OPEN;
                                }
                        }
                }
                else
                {
                        if (doorStatus == DOOR_OPEN)
                        {
                                if (automaticControl == YES)
                                {
                                        printf("The door closed. ");
                                        getTimeUsart();
                                        moveDoor();
                                        doorStatus = DOOR_CLOSED;
                                }
                        }
                }

                askSensor = NO;
        }
}

void getInfoEvery5sec()
{
        if (getInfoFromUsart == YES)
        {
                getInfoUsart();
                getInfoFromUsart = NO;
        }
}

void portInit()
{
        BUT_DDR=0x00;
        BUT_PORT=0x0f;
```

```
	DDRD = 0xf0;
	PORTD = 0x00;
}

int main(void)
{
	portInit();
	usartInit();
	timer1Init();

	sei();

    while (1)
    {
		checkingButtons();
		askDHT11();
		getInfoEvery5sec();
    }
}
```

ПРИЛОЖЕНИЕ Б. Спецификация радиоэлементов схемы

Поз. обозначение	Наименование	Кол.	Примечание
	Микросхемы		
DD1	ATMega8	1	
DD2	MAX232	1	
DD3	HC-06	1	
DD4	L297	1	
DD5	L298	1	
DD6	DHT11	1	
	Резисторы		
R1-R4	Р1-4-0,125 10кОм±5,0%	4	
R5	Р1-4-0,125 5кОм±5,0%	1	
	Конденсаторы		
C1, C8	К10-17Б 100 мкФ, 50В, 5%	2	
C2, C3	К10-17Б 33 пФ, 50В, 5%	2	
C4-C7	К10-17Б 1 мкФ, 50В, 5%	4	
	Прочие элементы		
XS1	Гнездо DB9F	1	
XP1	Вилка IDC-06MS	1	
XP2, XP3	Вилка MPW-2R	2	
SB1-SB4	Кнопка TC-0120	4	
ZQ1	Кварцевый резонатор НС-4.95 (8.00МГЦ)	1	

Перв. примен. | Справ. № | Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Андреев Д.О.		
Пров.		Трамов И.Б.		
Н.контр.				
Утв.				

Контроллер теплицы

Спецификация радиоэлементов схемы

Лит.	Лист	Листов
		1

МГТУ им. Н.Э. Баумана ИУ6-73Б

Копировал Формат А4