

Министерство образования и науки 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)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
НА ТЕМУ:
Многоканальный регистратор температуры

Студент ИУ6-72Б	_____	Р.Д. Векшин
	(Подпись, дата)	(И.О. Фамилия)
Руководитель курсовой работы	_____	В.Я. Хартов
	(Подпись, дата)	(И.О. Фамилия)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-6

 (А.В. Пролетарский)

« 4 » 09 2019 г.

ЗАДАНИЕ на выполнение курсовой работы

по дисциплине Микропроцессорные системы

Студент _____ Векшин Р. (ИУ6-72) _____
(фамилия, инициалы, индекс группы)

Руководитель Хартов В.Я.

График выполнения работы: 25% - 4 нед., 50% - 7 нед., 75% - 11 нед., 100% - 14 нед.

Тема курсовой работы Многоканальный регистратор температуры

Разработать на основе микроконтроллера (например, ATmega8) регистратор температуры от 7-и цифровых датчиков типа DS1621 (или DS1821). Сравнить показания датчиков с заданными пороговыми значениями температуры. Показания, превышающие пороговые значения, поочередно вывести на дисплей из 7-сегментных индикаторов, указав номер датчика и его показание, и переслать в ПЭВМ по последовательному каналу.

Разработать схемы, алгоритмы и программы. Отладить модули разработанной программы с помощью симулятора.


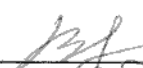
Оценить потребляемую мощность устройства.

Оформление курсовой работы

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

Дата выдачи задания 4 сентября 2019 г.

Руководитель курсовой работы

Задание получил  / Векшин Р.В. /  / Хартов В.Я. / « 04 » сентября 2019 г.

Даты защиты - 20 декабря 2019 г.

Примечание: 1. Задание оформляется в двух экземплярах; один выдаётся студенту, второй хранится на кафедре.

РЕФЕРАТ

РПЗ 68 с., 49 рис., 3 табл., 4 источника, 2 прил.

Объектом разработки курсовой работы является регистратор температуры от 7 цифровых датчиков.

Цель работы — создание полного комплекта конструкторской документации для регистратора температуры, создание программного обеспечения для микроконтроллера семейства AVR.

При проектировании решены следующие задачи: анализ объекта разработки на функциональном уровне, разработка функциональной схемы, выбор элементной базы для реализации объекта, разработка принципиальной схемы, расчет потребляемой мощности; разработка алгоритмов работы микроконтроллера и написание соответствующих программ.

Результатом проектирования является комплект конструкторской документации для изготовления устройства, исходные коды программ для программирования памяти микроконтроллера.

Устройство должно обладать следующими техническими характеристиками:

- а) опрашивать до 7 цифровых датчиков температуры;
- б) выводить показания датчиков на цифровой семисегментный индикатор;
- в) выводить показания датчиков на ПЭВМ с помощью последовательного интерфейса;
- г) обладать пультом (кнопками) для изменения режимов работы;
- д) выводить показания, которые находятся выше верхней температурной границы или ниже нижней температурной границы;
- е) работать от линии питания постоянного тока 12В.

Ключевые слова: микроконтроллер AVR, цифровой датчик температуры, TWI, SPI, UART, клавиатура.

СОДЕРЖАНИЕ

Введения	7
Основная часть	8
1 Конструкторская часть	9
1.1 Описание структурно-функциональной схемы микроконтроллерной системы	9
1.2 Выбор микроконтроллера	10
1.3 Описание архитектуры и технические характеристики микроконтроллера	11
1.4 Распределение адресного пространства МК	12
1.5 Описание принципиальной электрической схемы	14
1.5.1 Цифровой датчик температуры DS1621	14
1.5.1.1 Описание интерфейса TWI	16
1.5.2 Схема понижения входного напряжения до 5В	20
1.5.3 Подключения кнопок к МК	21
1.5.4 Вывод показаний на ССИ	22
1.5.4.1 Выбор способа подключения ССИ к МК	22
1.5.4.2 Сдвиговый регистр 74595	24
1.5.4.3 Описание интерфейса SPI	25
1.5.4.4 Расчет резисторов для ССИ	29
1.5.5 Схема для передачи данных на ПЭВМ	30
1.5.5.1 Описание интерфейса USART	30
1.6 Описание граф-схем алгоритмов функционирования	35
1.6.1 Главная программа	35
1.6.2 Таймерные функции	36
1.6.2.1 Асинхронный вывод показаний на ССИ	36
1.6.2.2 Циклическая смена отображаемого датчика и его показаний	37
1.6.2.3 Мерцание дисплея в режимах «Установка ТН» и «Установка TL»	37
1.6.2.4 Расчет делителей таймеров	38
1.6.3 Обработка сигналов от клавиатуры	39
1.7 Расчет потребляемой мощности	41
2 Технологическая часть	42
2.1 Характеристики использованных систем разработки	42
2.1.1 Симуляция в Proteus 8	43
2.2 Способы программирования памяти МК	46
2.2.1 Алгоритм последовательного программирования через SPI	47
2.2.2 Опрос данных флэш-памяти	49
2.2.3 Опрос данных EEPROM	49
Заключение	50
Список использованных источников	51

Приложения	52
Приложение А - Исходные коды программ	52
Приложение Б - Спецификация радиоэлементов схемы	66

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

МК — Микроконтроллер

ПЗУ — Постоянное запоминающее устройство

ПЭВМ — Персональная электронно-вычислительная машина

ССИ — Семисегментный индикатор

ШИМ — Широтно-импульсный модулятор

РОН — (General purpose registers) – регистры общего назначения

EEPROM — (Electrically Erasable Programmable Memory) Электрически стираемое программируемое ПЗУ

I2C — (Inter-Integrated Circui) Двухпроводной последовательных интерфейс, аналог TWI

INT — (Interrupt) Сигнал прерывания

ISP — (In System Programming) Внутрисхемное программирование

SPI — (Serial Peripheral Interface) Последовательный периферийный интерфейс

TWI — (Two Wire Interface) Двухпроводной последовательных интерфейс, аналог I2C

UART — (Universal asynchronous receiver/transmitter) Универсальный асинхронный приёмопередатчик

USI — (Universal Serial Interface) Универсальный последовательный интерфейс

ВВЕДЕНИЕ

В данной работе производится разработка регистратора температуры от 7-и цифровых датчиков на основе 8-разрядного высокопроизводительного AVR микроконтроллера на основании учебного плана кафедры ИУ6.

Использование шины TWI позволяет подключить к регистратору температуры до 7 датчиков, не изменяя количество задействованных для обмена данными портов микроконтроллера. Используя кабель связи вида витая пара, можно подключить датчики температуры, находящиеся в десятках или сотнях метрах от регистратора. Благодаря тому, что датчики потребляют небольшое количество электроэнергии, то можно подвести к ним питание через ту же витую пару.

Преимуществом разрабатываемого регистратором температуры являются возможность вывода показаний, которые выходят за установленные границы минимальной и максимальной допустимых температур, и возможность изменить данные границы с пульта оператора, не прибегая к использованию программатора или ПЭВМ.

ОСНОВНАЯ ЧАСТЬ

В данной курсовой работе разработан регистратор температуры от 7-и цифровых датчиков на основе 8-разрядного высокопроизводительного AVR микроконтроллера.

В техническом задании можно выбрать цифровой датчик DS1621 или DS1821.

Микросхема DS1621 представляет собой термометр и термостат в одном корпусе с цифровым вводом и выводом, которая гарантирует точность измерения и контроля с погрешностью плюс – минус 0,5 гр. Цельсия. Если использовать датчик DS1621 в роли термометра, то данные должны обрабатываться через I2C/TWI последовательную шину в дополнительном девяти – битном коде с точностью младшего разряда плюс – минус 0,5 гр. Цельсия. Применяя датчик DS1621 в роли термостата, в DS1621 имеются регистры TH (повышенная температура) и TL (пониженная температура). При превышении текущей температуры уровня TH выход датчика перейдет в активное состояние, и будет продолжать оставаться в нем, пока текущая температура не опустится ниже отметки TL. Таким образом, реализуется управление с заданным гистерезисом.

Микросхема DS1821 представляет собой термометр или термостат. В режиме термометра DS1821 осуществляет контроль над температурой от -55 до +125°C. Обработка показателей выполняется при помощи 1-Wire интерфейса фирмы DALLAS, значение измеряемой температуры представляет собой 8-и битное число.

Принципиальным отличием этих типов датчиков является интерфейс общения: TWI у DS1621 и 1-Wire у DS1821. Т.к. по условию ТЗ необходимо обеспечить подключение 7 датчиков температуры, то DS1621 будет интересен тем, что количество занятых портов ввода/вывода МК не изменяется в зависимости от количества датчиков. Для DS1821 потребуется до 7 контактов (по 1 на каждый датчик). Примем решение, что в данной курсовой работе следует использовать датчик с интерфейсом TWI.

1 Конструкторская часть

1.1 Описание структурно-функциональной схемы микроконтроллерной системы

Согласно техническому заданию требуется разработать устройство - регистратор температуры. Итоговое устройство должно обеспечивать вывод показаний датчиков на ССИ. Необходимо предусмотреть возможность выбора пользователем датчика, для отображения показаний, и изменения верхней и нижней границ температуры. Дополнительно следует пересылать показания датчиков в ПЭВМ по последовательному каналу.

Соответственно, устройство должно содержать следующие блоки:

- а) 7 цифровых датчиков температуры;
- б) цифровой семисегментный индикатор (Блок индикации);
- в) блок передачи информации на ПЭВМ;
- г) пульт оператора;

Структура проектируемого устройства представлена на рисунке 1

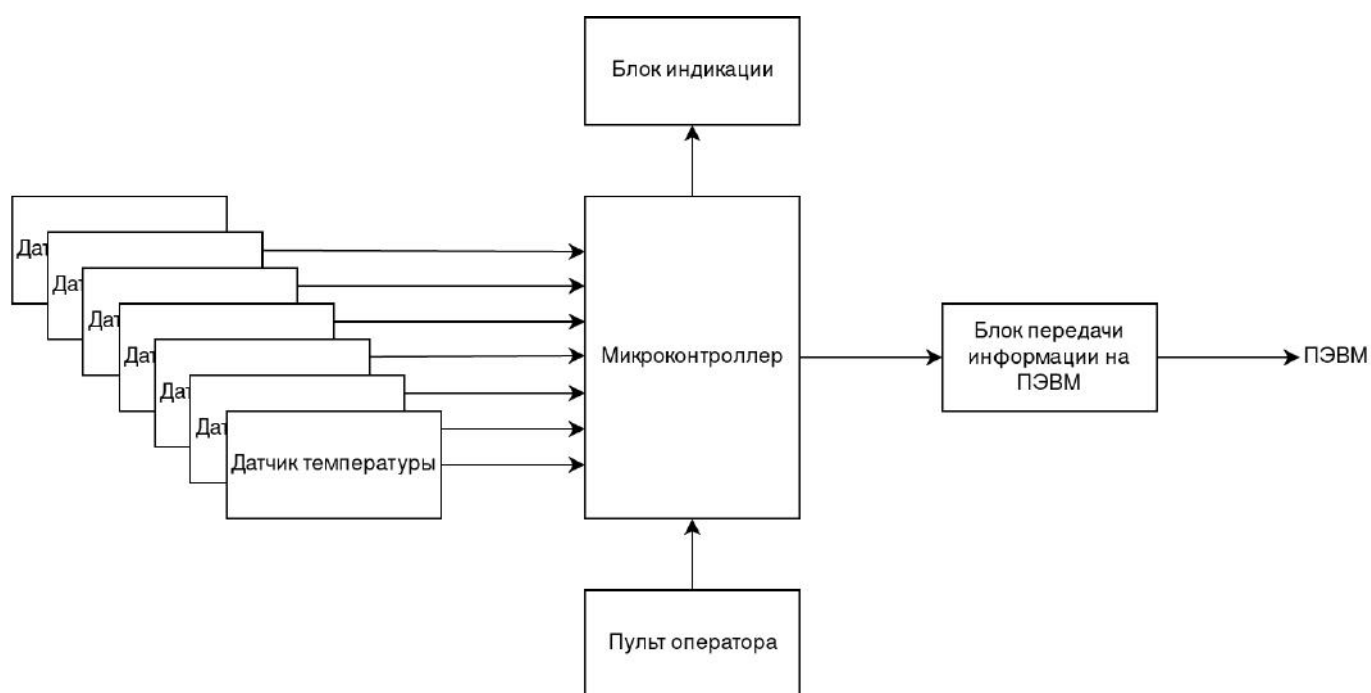


Рисунок 1 — Структурная схема регистратора температуры

1.2 Выбор микроконтроллера

Согласно техническому заданию регистратор температуры необходимо разработан на основе микроконтроллера, например, Atmega8. В таблице 1 представлено сравнение важных для нашей системы параметров МК семейства AVR.

Таблица 1 — Сравнительная таблица микроконтроллеров семейства AVR

Микроконтроллер	Контактов	ПЗУ, КБ	SRAM, Байт	Таймеры	EEPROM, Байт
ATtiny2313A	20	2	128	1/1	128
ATmega8A	32	8	1024	2/1	512
ATmega8535	40	8	512	2/1	512

Представленные в таблице 1 микроконтроллеры семейств ATtiny и ATmega выбраны из-за аппаратной реализации UART/USART, I2C, SPI. Семейство МК ATxmega исключено из сравнительной таблицы, так как данные микроконтроллеры предоставляют избыточную производительность и количество портов ввода/вывода.

Микроконтроллер ATtiny2313A не подходит для использования в данном проекте, так как в данном микроконтроллере аппаратно реализован интерфейс USI, на котором в дальнейшем можно программно реализовать интерфейсы SPI или I2C, но использовать их одновременно нельзя.

Микроконтроллер ATmega8535 не подходит для использования в данном проекте, так как имеет избыточное для данного проекта количество вводов/выводов и его стоимость в 3-4 раза больше ATmega8A.

1.3 Описание архитектуры и технические характеристики микроконтроллера

В проектируемом устройстве используется 8-битный микроконтроллер семейства AVR ATmega8. Рассмотрим его функциональную схему (см. рис. 2).

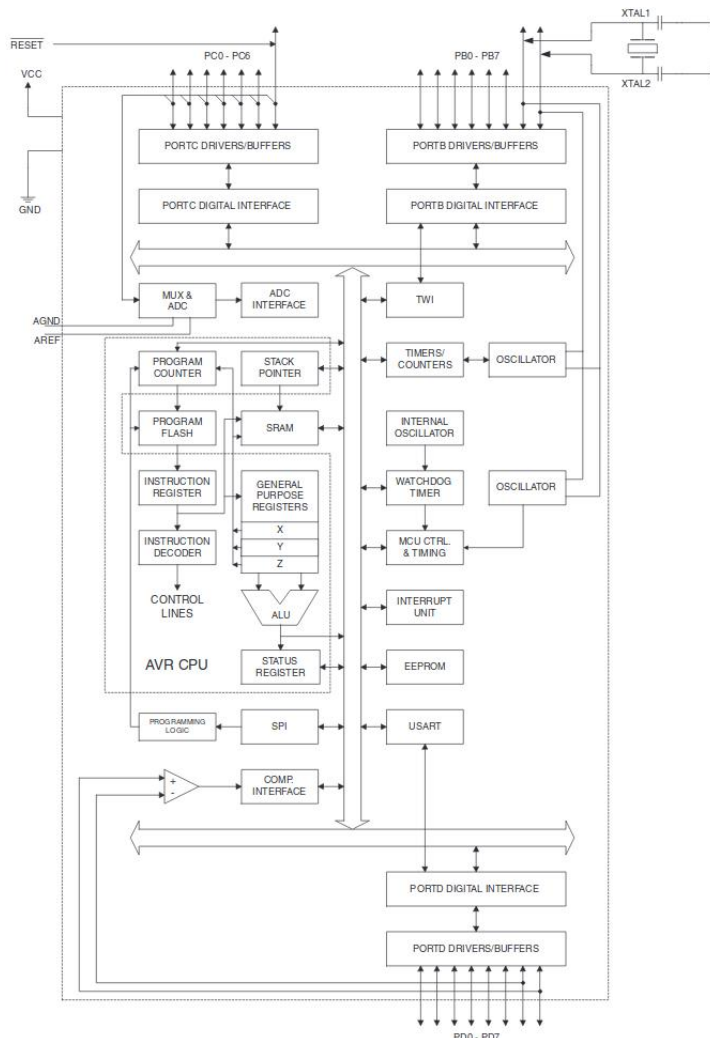


Рисунок 2 — Функциональная схема микроконтроллера ATmega8

Из функциональной схемы видно, что микроконтроллер обладает 3 цифровыми портами, один из которых имеет АЦП с мультиплексором; аппаратно реализованы интерфейсы USART, TWI, SPI; имеет встроенный компаратор, встроенные генератор (осцилятор), счетчики, сторожевой таймер, блок прерываний и энергонезависимую память.

Семейство микроконтроллеров Mega AVR фирмы Atmel, это 8-битные RISC микроконтроллеры для встраиваемых приложений. Они представляют собой одну из лучших основ, для создания современных экономичных и высокопроизводительных устройств различного назначения.

Микроконтроллеры семейства Mega AVR изготавливаются по малопотребляющей КМОП-технологии, которая в сочетании с RISC-архитектурой позволяет получить наилучшее соотно-

шение стоимости, быстродействия и энергопотребления. Микроконтроллеры данного семейства являются наиболее развитыми представителями микроконтроллеров AVR общего применения.

Одним из популярных представителей семейства Mega AVR является ATmega8. Данный МК обладает следующими характеристиками:

- работает на частотах от 0 до 16 МГц,
- содержит 23 программируемых линии ввода/вывода,
- 8 Кбайт внутрисистемно программируемой Flash памяти,
- 512 байт EEPROM,
- 1 Кбайт встроенной SRAM,
- два 8-разрядных таймера/счетчика с отдельным предварительным делителем, один с режимом сравнения,
- один 16-разрядный таймер/счетчик с отдельным предварительным делителем и режимами захвата и сравнения,
- счетчик реального времени с отдельным генератором,
- три канала ШИМ,
- байт-ориентированный 2-проводный последовательный интерфейс,
- программируемый последовательный USART,
- последовательный интерфейс SPI (ведущий/ведомый),
- программируемый сторожевой таймер с отдельным встроенным генератором,
- встроенный калиброванный RC-генератор,
- внутренние и внешние источники прерываний,
- пять режимов пониженного потребления: Idle, Power-save, Power-down, Standby и снижения шумов ADC.

Более подробное описание микроконтроллера можно прочитать в спецификации [3].

1.4 Распределение адресного пространства МК

В микроконтроллерах AVR семейства Mega реализована Гарвардская архитектура, в соответствии с которой разделены не только адресные пространства памяти программ и памяти данных, но также и шины доступа к ним. 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 (ОЗУ и EEPROM) также расположена в своём адресном пространстве. Карта памяти МК ATmega8 представлена на рисунке 3.

Память данных микроконтроллеров семейства Mega разделена на три части: регистровая память, оперативная память (статическое ОЗУ) и энергонезависимое ЭСППЗУ (EEPROM).

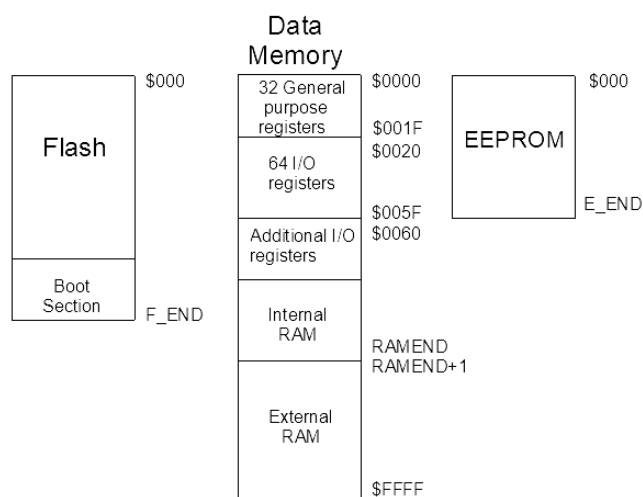


Рисунок 3 — Распределение адресного пространства МК ATmega8

Регистровая память включает 32 регистра общего назначения (РОН), объединённых в файл, и служебные регистры ввода/вывода (РВВ).

В обеих областях регистров ввода/вывода располагаются различные служебные регистры (регистры управления микроконтроллера, регистр состояния и т. п.), а также регистры управления периферийными устройствами, входящими в состав микроконтроллера. Общее количество РВВ и ДРВВ зависит от конкретной модели микроконтроллера.

Для хранения переменных помимо регистров общего назначения также используется статическое ОЗУ объёмом от 512 байт до 8 Кбайт. Ряд микроконтроллеров семейства, кроме того, имеют возможность подключения внешнего статического ОЗУ объёмом до 64 Кбайт.

Для долговременного хранения различной информации, которая может изменяться в процессе функционирования готовой системы (например, калибровочные константы, серийные номера, ключи и т. д.), в микроконтроллерах семейства может использоваться встроенная EEPROM-память. Её объём составляет для различных моделей от 256 байт до 4 Кбайт. Эта память расположена в отдельном адресном пространстве, а доступ к ней осуществляется с помощью определённых РВВ.

В адресном пространстве ОЗУ расположены все регистры микроконтроллеров (РОН, регистры ввода/вывода и дополнительные регистры ввода/вывода), под них отведены младшие адреса. Остальные адреса отведены под ячейки статического ОЗУ.

1.5 Описание принципиальной электрической схемы

1.5.1 Цифровой датчик температуры DS1621

В качестве цифрового датчика температуры выбран датчик DS1621. Микросхема DS1621 это термометр и термостат с цифровым вводом/выводом, обеспечивающий точность $\pm 0.5^{\circ}\text{C}$. При использовании в качестве термометра, данные считываются через I2C/SMBus последовательную шину в дополнительном 9-битном коде с ценой младшего разряда $\pm 0.5^{\circ}\text{C}$. Для приложений требующих более высокого разрешения, пользователь может прочитать дополнительные регистры и произвести простые арифметические действия, чтобы достичь более чем 12-битового разрешения (с ценой самого младшего разряда 0.0625°C). Микросхема DS1621 обеспечивает 3 адресных входа, чтобы позволить пользователям подключить до 8 DS1621 к одной шине.

Для приложений, которым не требуется точность $\pm 0.5^{\circ}\text{C}$, доступна микросхема DS1721 с пониженной точностью $\pm 1^{\circ}\text{C}$, более дешёвая полностью совместимая микросхема (только в корпусе SOIC).

При использовании в качестве термостата, микросхема DS1621 имеет во внутренней энергонезависимой памяти (EEPROM) программируемые пользователем контрольные точки по превышению температуры (TH) и по понижению температуры (TL). Один специальный логический выход сработает, когда TH достигнут, и выход будет оставаться активным до тех пор, пока температура не упадёт ниже TL (программируемый гистерезис).

Микросхема DS1621 предлагается в 300mil, 8-контактном PDIP и 150mil, 8-контактном SOIC (см. рис. 4). Описание выходов:

- SDA - линия данных шины I2C
- SCL - линия тактового сигнала шины I2C
- Tout - выход термостата
- Vdd - плюсовой вывод питания
- Vss - минусовой вывод питания
- A0..A2 - линии формирования младших битов адреса

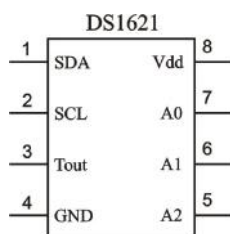


Рисунок 4 — Назначение выводов датчика DS1621

Максимальная тактовая частота (clock rate) обмена данными для данного датчика 400кГц. Скорость обмена вычисляется по формуле $F_{scl} = F_{cpu} / (16 + 2 * TWBR * 4^{TWPS})$. При

частоте МК 4МГц и минимальном делителе 16 получим максимальную тактовую частоту 250 кГц.

Принцип работы

Датчик температуры DS1621 для измерения использует принцип нестабильности частоты колебаний при изменении температуры. Для этого в ее состав входят два генератора. Первый имеет высокую температурную стабильность. Его частота соответствует температуре -55 градусов и практически не подвержена изменениям. Частота работы второго генератора, наоборот, изменяется пропорционально температуре. Специальные счетчики импульсов производят подсчет за одинаковый временной интервал и на основе разности, вычисляют значение температуры. Это значение в 9-разрядном двоичном коде доступно пользователю. Данные разбиваются на старший и младший байты. Если достаточно целое значение температуры, то можно пользоваться только старшим байтом. Младший байт имеет только один информационный бит LSB, обеспечивающий дискретность 0.5 градуса. Остальные биты младшего байта всегда равны 0.

Регистр состояния

7	6	5	4	3	2	1	0
DONE	THF	TLF	NVB	1	0	POL	ISHOT

Микросхема DS1621 имеет несколько режимов работы. Настройка и отслеживание этих режимов производится с помощью регистра состояний. Имеются следующие доступные биты:

- **DONE** – флаг окончания преобразования. Устанавливается по завершении преобразования.
- **THF** – флаг «высокая температура». Устанавливается при превышении порога TH. Сбрасывается программно или отключением питания.
- **TLF** – флаг «низкая температура». Устанавливается при температуре меньшей, чем значение порога TL. Сбрасывается программно или отключением питания.
- **NVB** – флаг записи данных в энергонезависимую память. Установленный флаг свидетельствует о незавершенности записи. Время записи ячейки составляет ориентировочно 10 мс.
- **POL** – полярность выхода Tout. Высокое значение соответствует прямой полярности, низкое – обратной. Бит энергонезависим.
- **ISHOT** – управление циклом измерений. При высоком логическом уровне измерение выполняется однократно. Данный режим используется в энергосберегающих системах. Низкий логический уровень бита, разрешает выполнение преобразования в непрерывном режиме. Бит энергонезависим.

Команды обмена

Обмен данными с DS1621 производится по стандартному протоколу I2C. Микросхема участвует в нем в качестве slave-устройства. Slave-адрес DS1621 имеет вид 1001xxx, где xxx – состояние линий A0-A2 микросхемы. Для работы с DS1621 используются следующие команды:

- **22h** – «**Останов преобразования**» - команда производит завершение работы схемы преобразования температуры. Дополнительных данных для работы не требуется.
- **AAh** – «**Чтение температуры**» - Результатом работы команды являются два байта данных, содержащих значение измеренной температуры.
- **A1h** – «**Установка TH**» - команда установки верхнего порога срабатывания термостата. После данной команды требуется передача двух байтов значения порога.
- **A2h** - «**Установка TL**» - команда установки нижнего порога срабатывания термостата. После данной команды требуется передача двух байтов значения порога.
- **A8h** – «**чтение температурного счетчика**». Команда работает только на чтение и позволяет считать данные счетчика, частота работы которого зависит от температуры.
- **A9h** - «**чтение стабильного счетчика**». Команда работает только на чтение и позволяет считать данные счетчика, частота работы которого не зависит от температуры.
- **ACh** – «**Регистр конфигурации**». В зависимости от состояния бита R/W производится запись или чтение регистра конфигурации. Формат используемых данных – байт.
- **EЕh** – «**Старт счетчика**» - команда начала измерения температуры. Дополнительных данных не требуется.

1.5.1.1 Описание интерфейса TWI

Интерфейс I2C (или по другому IIC) — это широко распространённый сетевой последовательный интерфейс, придуманный фирмой Philips и завоевавший популярность относительно высокой скоростью передачи данных (обычно до 100 кбит/с, в современных микросхемах до 400 кбит/с), дешевизной и простотой реализации.

Физически сеть представляет собой двухпроводную шину, линии которой называются DATA и CLOCK (необходим ещё и третий провод — земля, но интерфейс принято называть двухпроводным по количеству сигнальных проводов). Соответственно, по линии DATA передаются данные, линия CLOCK служит для тактирования. К шине может быть подключено до 128 абонентов, каждый со своим уникальным номером. В каждый момент времени информация передаётся только одним абонентом и только в одну сторону.

Устройства I2C имеют выход с "открытым коллектором". Когда выходной транзистор закрыт — на соответствующей линии через внешний подтягивающий резистор устанавливается высокий уровень, когда выходной транзистор открыт — он притягивает соответствующую линию к земле и на ней устанавливается низкий уровень (смотрите рисунок). Резисторы име-

ют номинал от нескольких килоОм до нескольких десятков килоОм (чем выше скорость — тем меньше номинал резисторов, но больше энергопотребление). На рисунке треугольниками на входе показано, что входы высокоомные и, соответственно, влияния на уровни сигналов на линиях они не оказывают, а только "считывают" эти уровни. Обычно используются уровни 5В или 3,3В.

Любое устройство на шине I2C может быть одного из двух типов: Master (ведущий) или Slave (ведомый). Обмен данными происходит сеансами. Мастер-устройство полностью управляет сеансом: инициирует сеанс обмена данными, управляет передачей, подавая тактовые импульсы на линию Clock, и завершает сеанс.

Каждый сеанс обмена начинается с подачи "Мастером" так называемого Start-условия. "Старт-условие" — это изменение уровня на линии Data с высокого на низкий при наличии высокого уровня на линии Clock. После подачи "Старт-условия" первым делом "Мастер" должен сказать с кем он хочет пообщаться и указать, что именно он хочет - передавать данные в устройство или читать их из него. Для этого он выдаёт на шину 7-ми битный адрес "Слэйв" устройства (по другому говорят: "адресует "Слэйв" устройство"), с которым хочет общаться, и один бит, указывающий направление передачи данных (0 - если от "Мастера" к "Слэйву" и 1 - если от "Слэйва" к "Мастеру"). Первый байт после подачи сигнала "Старт" всегда воспринимается ведомыми устройствами как адресация.

После того, как "Мастер" скажет, к кому именно он обращается и укажет направление передачи данных, — начинается собственно передача: "Мастер" выдаёт на шину данные для "Слэйва" или получает их от него. Эта часть обмена (какие именно данные и в каком порядке "Мастер" должен выдавать на шину, чтобы устройство его поняло и сделало то, что ему нужно) уже определяется каждым конкретным устройством.

Заканчивается каждый сеанс обмена подачей "Мастером" так называемого Stop-условия, которое заключается в изменении уровня на линии Data с низкого на высокий, опять же при наличии высокого уровня на линии Clock. Если на шине сформировано Stop-условие, то закрываются все открытые сеансы обмена.

Внутри сеанса любые изменения на линии Data при наличии высокого уровня на линии Clock запрещены, поскольку в это время происходит считывание данных "Приёмником". Соответственно, во время сеанса обмена установка данных "Передачиком" (выставление нужного уровня на линии Data) может происходить только при низком уровне на линии Clock.

Для настройки и управлением TWI интерфейсом используют пять регистров:

- регистр скорости передачи — TWBR;
- регистр данных — TWDR;
- регистр адреса — TWAR;
- регистр статуса — TWSR;
- регистр управления — TWCR.

Регистр TWBR

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 5 — Регистр TWBR

В TWI регистры задающие скорость передачи данных это регистр TWBR и два младших разряда статусного регистра TWSR - TWPS1 и TWPS0. Под скоростью передачи в данном случае подразумевается частота SCL сигнала, когда микроконтроллер работает в режиме ведущего устройства (мастера). В режиме ведомого (слейва) микроконтроллер "получает" SCL сигнал извне - от другого ведущего устройства. (Тактовая частота микроконтроллера в этом случае, должна быть больше частоты внешнего SCL сигнала минимум в 16 раз.)

Регистр TWDR

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

Рисунок 6 — Регистр TWDR

Данный регистр полностью отведён под запись/чтение в/из него байта данных.

Регистр TWAR

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

Рисунок 7 — Регистр TWAR

Микроконтроллер может выполнять функции как ведущего, так и ведомого устройства. Естественно по очереди, а не одновременно.

Если микроконтроллер выступает в роли ведомого, ему необходим адрес, на который он будет "отзываться". Регистр TWAR и предназначен для хранения 7-и разрядного адреса. Младший разряд (TWGCE) этого регистра разрешает/запрещает микроконтроллеру отзываться на общие вызовы (широковещательные пакеты), то есть на пакеты с адресом 0x00.

Регистр TWSR

Статусный регистр TWSR отражает состояние TWI модуля и двухпроводной шины, а также содержит разряды, задающие коэффициент деления частоты SCL сигнала.

Биты TWS7..TWS3 содержат статусный код. Биты доступны только для чтения, статусный код устанавливается TWI модулем аппаратно, после выполнения различных операций. Например, формирование состояния СТАРТ, передачи пакета данных и так далее. По значению статусного кода можно судить о результате операции. Выполнилась ли она успешно или нет.

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	—	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Рисунок 8 — Регистр TWSR

Регистр TWCR

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	—	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 9 — Регистр TWCR

Регистр TWCR состоит из следующих бит:

- **TWINT** — флаг прерывания TWI модуля. Этот бит устанавливается аппаратно, когда TWI модуль завершает текущую операцию (формирование состояния СТАРТ, передачи адресного пакета и так далее). При этом если установлен бит глобального разрешения прерываний (бит I регистра SREG) и разрешены прерывания TWI модуля, то вызывается соответствующий обработчик.

Бит TWINT очищается программно, записью единицы. При выполнении обработчика прерывания этот бит не сбрасывается аппаратно, как в других модулях. Сброс флага TWINT запускает работу TWI модуля, поэтому все операции с регистром данных, статуса или адреса, должны быть выполнены до его сброса. Пока бит TWINT установлен, на линии SCL удерживается низкий уровень;

- **TWEA** — разрешение бита подтверждения. Если бит TWEA установлен в 1, TWI модуль формирует сигнал подтверждения (ACK), когда это требуется. А требуется это в трех случаях: ведущее или ведомое устройство получило байт данных, ведомое устройство получило общий вызов, ведомое устройство получило свой адрес;
- **TWSTA** — флаг состояния СТАРТ. Когда этот бит устанавливается в 1, TWI модуль проверяет не занята ли шина и формирует состояние СТАРТ. Если шина занята, он будет ожидать появления на ней состояния СТОП и после этого выдаст состояние СТАРТ. Бит TWSTA должен быть очищен программно, когда состояние СТАРТ передано;
- **TWSTO** — флаг состояния СТОП. Когда этот бит устанавливается в 1 в режиме ведущего, TWI модуль выдает на шину состояние СТОП и сбрасывает этот бит. В режиме ведомого установка этого бита может использоваться для восстановления после ошибки. При этом состояние СТОП не формируется, но TWI модуль возвращается к начальному не адресованному состоянию;
- **TWWC** — флаг конфликта записи. Этот флаг устанавливается аппаратно, когда выполняется запись в регистр данных (TWDR) при низком значении бита TWINT. То есть когда TWI модуль уже выполняет какие-то операции. Флаг TWWC сбрасывается

аппаратно, когда запись в регистр данных выполняется при установленном флаге прерывания TWINT;

- **TWEN** — бит разрешения работы TWI модуля. Когда бит TWEN устанавливается в 1, TWI модуль включается и берет на себя управление выводами SCL и SDA. Когда бит TWEN сбрасывается, TWI модуль выключается;
- **TWIE** — разрешение прерывания TWI модуля. Когда бит TWIE и бит I регистра SREG установлены в 1 - прерывания модуля TWI разрешены. Прерывания будут вызываться при установке бита TWINT;

1.5.2 Схема понижения входного напряжения до 5В

По ТЗ на схему устройства подаётся напряжение 12В, но большинство элементов схемы работают от напряжения 5В. Следовательно, необходимо добавить в схему разрабатываемого регистратора температуры устройство, которое снижает входные 12В до необходимых 5В.

Популярным и дешёвым решением является установка линейного стабилизатора напряжения (например, LM7805 или КР142ЕН5А). Для поддержания нормального режима работы стабилизатора необходимо шунтировать полярными конденсаторами вход и выход стабилизатора (см. рис. 10).

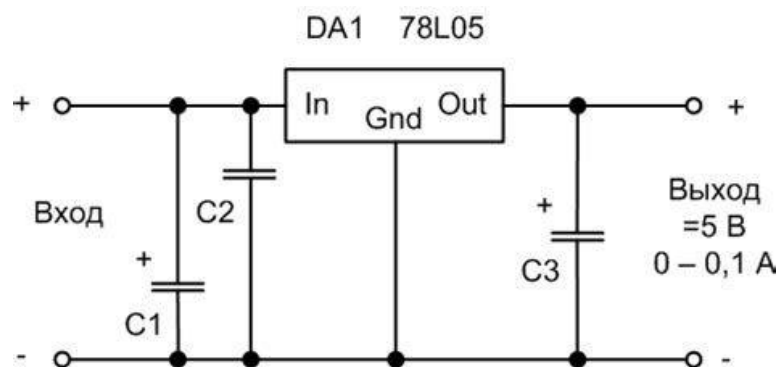


Рисунок 10 — Пример подключения линейного стабилизатора

Заметим, что стабилизатор напряжения не является трансформатором и лишняя мощность рассеивается на радиаторе устройства. При перегреве ($+145^{\circ}\text{C}$) устройство уходит в защитный режим на небольшой промежуток времени для того, чтобы снизить температуру. В схемах с большим потреблением (более 500 мА в постоянном режиме) следует установить линейному стабилизатору напряжения дополнительный радиатор, или заменить стабилизатор на импульсный понижающий преобразователь (buck-конвертер).

1.5.3 Подключения кнопок к МК

При уточнении технического задания выяснено, что должна иметься возможность изменять пороговые значения температур. Для этого необходимы кнопки «SET TH» и «SET TL» для установки верхней и нижней границ температур соответственно, кнопки «TEMP+» и «TEMP-» для увеличения и уменьшения температур, кнопка «OK» для подтверждения изменения пороговых значений.

Так же необходимо реализовать возможность вывода на дисплей показаний конкретного датчика. Для этого добавлены кнопки «PREV» и «NEXT» для выбора предыдущего и следующего датчика за текущим соответственно. Итого необходимо подключить к микроконтроллеру 7 кнопок.

Таблица 2 — Сравнительная таблица способов подключения кнопок

Способ подключения	Кол-во выходов	Диоды	Необходимость опроса
Традиционный	7	0	+
С использованием диодов	3	9	+
Матричный	6	0	+
Матричный с прерыванием	7	3	-
С использованием АЦП	1	0	+

Наиболее оптимальным по количеству использованных выходов микроконтроллера является способ с использованием АЦП см. табл. 2, но он имеет ограничения на количество подключаемых кнопок и резисторов, т. к. АЦП имеет ограниченную точность.

В качестве реализуемого варианта выбран матричный способ подключения с прерыванием, т.к. он позволяет не опрашивать кнопки в процессе работы программы, если они не нажаты.

1.5.4 Вывод показаний на ССИ

1.5.4.1 Выбор способа подключения ССИ к МК

В данном проекте необходимо вывести на ССИ номер датчика и его показание. Под номер датчика зарезервируем 1 разряд на индикаторе, т.к. по ТЗ количество датчиков не превышает 7 штук, и 1 разряд под разделитель. По документации [4] выбранный датчик температуры (DS1621) позволяет измерить температуру в диапазонах от -55°C до $+125^{\circ}\text{C}$ с повышенной точностью до 0.5°C . Для простоты считывания и обработки показаний будем считывать с датчиков только 1 байт показаний, т.е. работать с точностью 1°C . Для отображения температуры необходимо 3 разряда и 1 разряд для указания температурной шкалы. В итоге необходимо использовать ССИ на 6 разрядов и более.

В продаже имеются ССИ на несколько разрядов с объединёнными сегментными входами (A, B, C, D, E, F, G, DP). Предположим, что мы возьмём такой ССИ на 6 разрядов в одном корпусе. Тогда для работы с этим индикатором потребуется 8 выходов МК для работы с сегментами и 6 выходов МК для указания адреса разряда. Учитывая тот факт, что к выбранному микроконтроллеру подключена матрица кнопок через 7 выходов, датчики температуры через 2 выхода и 2 выхода МК используется для общения с ЭВМ, то остаётся всего 9 выходов, которых хватит только на работу с 1 разрядом ССИ при данном способе подключения.

Добавим в схему регистратора температуры сдвиговый регистр (например, 74НС595). Заметим, что сдвиговый регистр работает по SPI, который реализован в выбранном МК на аппаратном уровне, что позволит передавать данные в десятки раз быстрее, чем это можно сделать программным путём. Установив сдвиговый регистр мы используем для указания сегментов ССИ 4 выхода МК (MOSI для передачи данных, SCK для тактовых импульсов, SS для "защелкивания" данных и 1 выход для перевода выходов сдвигового регистра в высокоимпедансное состояние) вместо 8. В полученной схеме удаётся адресовать до 5 разрядов ССИ, что мало для нашей задачи.

Т.к. адресация разрядов осуществляется с помощью позиционного n-разрядного кода, то можем уменьшить количество занятых для адресации выходов МК с 6 до 3 поставив дешифратор (например, 74НС138) между выходами МК и адресными входами ССИ. Однако, заметим что можно установить десятичный счетчик с дешифратором в одном корпусе (элемент 4017В) и уменьшить количество занятых для адресации ССИ выходов МК с 6 до 2 (CLK для выбора следующего разряда и MR для сброса адреса).

Примечательно, что сигнал перевода выходов сдвигового регистра в высокоимпедансное состояние и сигнал синхронизации в счетчике-дешифраторе следуют друг за другом и одинаковы по форме. Используем этот факт и подключим вход CLK счетчика-дешифратора к линии для перевода сдвигового регистра в Z-состояние. Получим, что для работы ССИ используется всего 5 выходов МК (3 для передачи данных и 2 для адресации).

Универсальным решением, но которое не будет использовано в данной курсовой работе, является использование микросхемы MAX7219 (MAX7221) для управления ССИ. Использование данного драйвера в электронных устройствах на МК значительно упрощает вывод информации на индикаторы. Пропадает необходимость в реализации динамической индикации (не нужно микроконтроллеру N раз в секунду полностью обновлять показания на ССИ), нет необходимости в обвязке ССИ токоограничивающими резисторами и транзисторами для подключения адресных выводов к линии питания или земле. В случае использования данной микросхемы используется всего 3 выхода МК (MOSI, SCK, SS).

Таблица 3 — Сравнительная таблица способов подключения СИИ

Способ подключения	Выходы (Адр. + Дан. + Упр.)	Микросхемы	Цена, руб
Прямой	6 + 8 + 0	0	0
74595	6 + 3 + 1	1	2.63
74595 + 74138	3 + 3 + 1	2	2.63 + 3.13
74595 + 4017В	3 + 1 + 1	2	2.63 + 3.28
MAX7219	3 + 0 + 0	1	14.41

Семисегментные индикаторы бывают двух видов: с общим анодом (ОА) и с общим катодом (ОК). В данном проекте был выбран ССИ с общим катодом (см. рис. 11), т.к. они управляются с помощью счетчика, на выходе которого в нужном разряде 1, а в остальных 0, но если имеется в наличии только ССИ с ОА, то можно изменить программу так, что бы на порт, передающий данные на сегменты ССИ, отправлять инвертированный код.

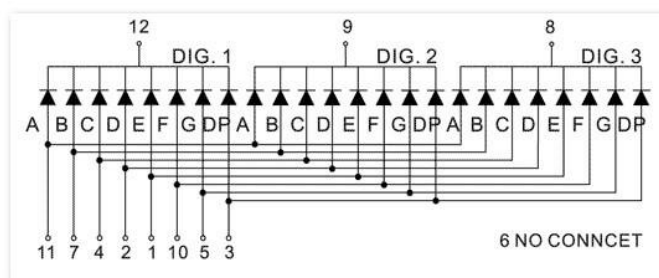


Рисунок 11 — Внутреннее строение ССИ с ОК

Суммарный ток со всех сегментов ССИ может превышать допустимый ток для порта МК. Для того, чтобы управлять этим током устанавливают транзисторы: NPN для ОК и PNP для ОА (см. рис. 12). Между базами транзисторов и выводами порта микроконтроллера необходимо включать резисторы, сопротивление которых зависит от типа транзистора (номиналы резисторов рассчитываются).

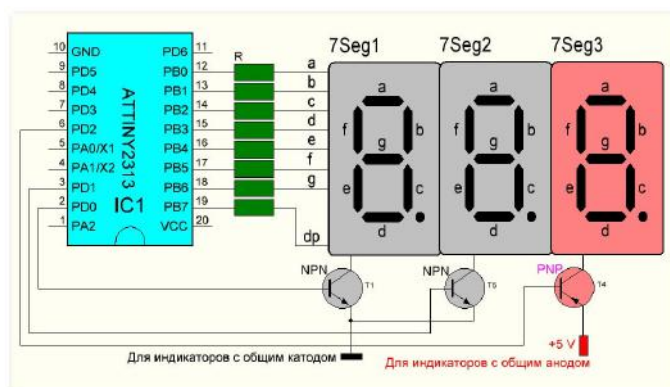


Рисунок 12 — Управление СИИ через транзисторы

1.5.4.2 Сдвиговый регистр 74595

Микросхема 74НС595 содержит 8 битный регистр хранения и 8 битный сдвиговый регистр. Данные последовательно передаются в сдвиговый регистр, затем фиксируются в регистре хранения. К регистру хранения подключены 8 выходных линий. На картинке ниже показано расположение выводов микросхемы 74НС595.

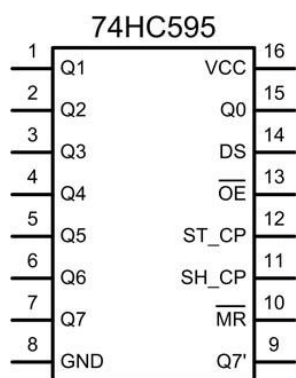


Рисунок 13 — Сдвиговый регистр 74595

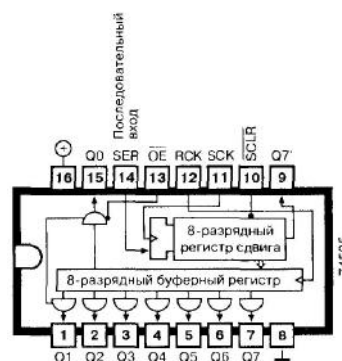


Рисунок 14 — Сдвиговый регистр 74595

Вывод 14 (DS) это вывод данных. В некоторых описаниях он обозначается как «SER».

Когда уровень на выводе 11 (SH_CP, иногда обозначается как SRCLK) переходит из низкого в высокий, значение на выводе DS сохраняется в сдвиговом регистре, при этом данные сдвигаются на один разряд, чтобы предоставить место для нового бита.

Пока на выводе 12 (ST_CP, иногда обозначается как RCLK) низкий уровень, данные записываются в регистр сдвига. Когда уровень переходит в высокий, данные из сдвигового регистра фиксируются в регистре хранения, из которого поступают на выводы Q0...Q7.

На представленной ниже временной диаграмме (см. рис. 15), показано, каким образом можно установить на выходах Q0...Q7 микросхемы значение 11000011, учитывая что изначально там было значение 0x00.

В документации к сдвиговому регистру 74595 указано, что данная микросхема может получать данные от внешнего источника по интерфейсу SPI (режим 0).

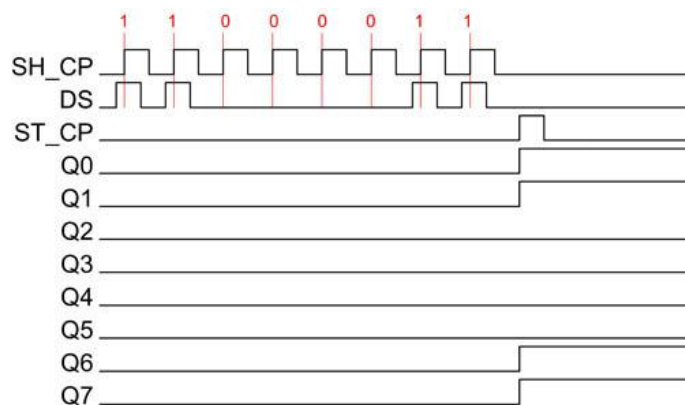


Рисунок 15 — Временная диаграмма для сдвигового регистра 74595

1.5.4.3 Описание интерфейса SPI

Serial Peripheral Interface или SPI — последовательный периферийный интерфейс, служит для связи периферии и микроконтроллера. Например, в качестве периферии может быть: дисплей, различные датчики, FLASH память, SD карта и т.д.

В SPI всегда есть один ведущий и один/несколько ведомых. Передачу данных всегда инициализирует ведущий.

В SPI используются четыре линии связи:

- **MOSI** — выход ведущего, вход ведомого (англ. MasterOutSlaveIn). Служит для передачи данных от ведущего устройства ведомому.
- **MISO** — вход ведущего, выход ведомого (англ. MasterInSlaveOut). Служит для передачи данных от ведомого устройства ведущему.
- **SCLK** — последовательный тактовый сигнал (англ. SerialClock). Служит для передачи тактового сигнала для ведомых устройств.
- **CS** или **SS** — выбор микросхемы, выбор ведомого (англ. Chip Select, Slave Select).

Для обеспечения односторонней связи с одним устройством, достаточно использовать SCLK, MOSI (в случае если ведомое устройство только принимает) или SCLK, MISO (в случае если ведомое устройство ничего не принимает, а только передает информацию).

SPI может быть реализован в микроконтроллере аппаратно, тогда задача по управлению интерфейсом решается для каждого микроконтроллера отдельно. При работе с SPI нам нужно самим программным путем установить на SS логический ноль при начале приема/передачи и установить логическую единицу обратно при окончании передачи.

Ведомые микросхемы могут по разному «интерпретировать» принятый сигнал по SPI, отличие может заключаться в следующих моментах:

- в размере передающих данных или размер пакета, обычно это 8 бит, но бывает и больше
- в порядке следования бит, сначала старший бит или сначала младший бит

- по какому уровню синхросигнала передаются данные (по логической единицы (HIGH) или логическому нулю (LOW))
- по какому фронту импульса происходит синхронизация (по подъему или спуску), кратко это называют «фазой синхронизации»

В микроконтроллере ATmega8 (микроконтроллер фирмы Atmel серии AVR) можно управлять следующими параметрами SPI:

- тип устройства (какую роль выполняет микроконтроллер, в качестве ведущего или ведомого)
- порядок следования бит при передаче данных, сначала старший или младший бит
- режим работы SPI, с помощью комбинации 2-ух параметров — «полярность синхросигнала» и «фаза синхронизации».
- частоту обмена данными по SPI

Для настройки и управлением SPI интерфейсом используют три регистра:

- регистр управления — SPCR
- регистр состояния — SPSR
- регистр данных — SPDR

Регистр SPCR

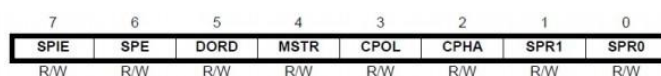


Рисунок 16 — Регистр SPCR

- **SPIE** – разрешить прерывания от SPI (прерывание будет в том случае если установлен бит глобального разрешения прерываний регистра SREG (7-й бит)). После окончания передачи байта будет сгенерировано прерывание.
- **SPE** — подключить SS, MOSI, MISO и SCK к портам микроконтроллера ATmega328P — PB2,PB3,PB4,PB5.
- **DORD** – определит, что по SPI сначала передается младший разряд, а потом старший – режим «LSB». Логический ноль, наоборот, что сначала передается старший разряд, а далее младший – режим «MSB».
- **MSTR** — режим ведущий: единица - включить, ноль – включить режим ведомого.
- **CPOL** (полярность сигнала синхронизации или уровень синхронизации) – синхронизация ведется по отсутствию импульса (по логическому нулю) или тактовый сигнал в состоянии ожидания равен 1-цы. Логический ноль — синхронизация ведется по присутствию импульса (по логической единицы) или тактовый сигнал в состоянии ожидания равен 0-лю . По какому фронту (спад или подъем) ведется синхронизация определяется в 2-ом бите (CPHA).

- **CPHA** (фаза синхронизации) – определяет, что сигнал синхронизации определяется по спадающему фронту SCK, а логический ноль по нарастающему фронту SCK. Причем если в CPOL установлена 1-ца, то спадающий и нарастающий фронт если изобразить на диаграмме «вверхногами» (см. рис. 18 и 19).
- **SPR0 и SPR1** совместно с битом SPI2X в регистре SPSR определяют скорость передачи данных по SPI (или скорость тактовых сигналов по SCK) (см. рис. 17, где f_{osc} тактовая частота задающего генератора SPI (обычно она равна частоте тактирования процессора)). Данные биты имеют смысл только для ведущего, для ведомого они бессмысленны, т.к. скорость приема зависит от частоты SCK ведущего.

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Рисунок 17 — Скорость интерфейса SPI

Figure 18-3. SPI Transfer Format with CPHA = 0

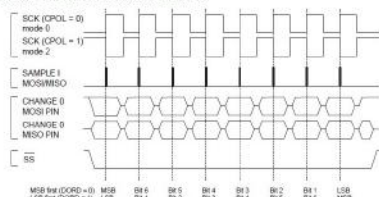


Figure 18-4. SPI Transfer Format with CPHA = 1

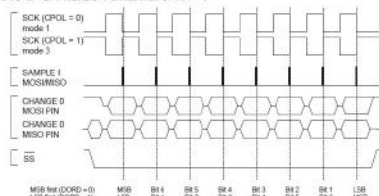


Figure 18-3. SPI Transfer Format with CPHA = 0

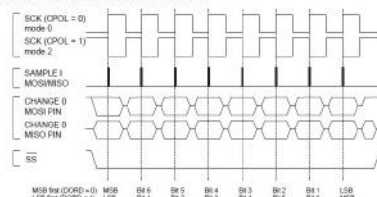


Figure 18-4. SPI Transfer Format with CPHA = 1

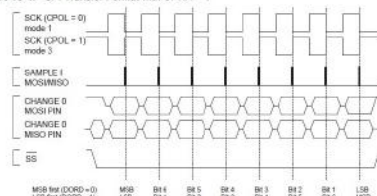


Рисунок 18 — Режимы SPI 0 и 2. CPHA=0

Рисунок 19 — Режимы SPI 1 и 3. CPHA=1

Сдвиговой регистр может работать на частоте 20 МГц. Установим биты SPI2X = 1, SPR1 = 0, SPR0 = 0. Получим частоту передачи $F_{OSC}/2 = 4\text{МГц}/2 = 2\text{МГц}$.

Регистр SPSR

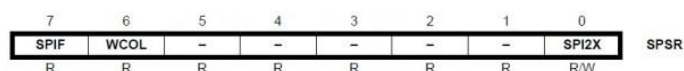


Рисунок 20 — Регистр SPSR

- **SPIF** - SPI Interrupt Flag - В бит устанавливается единица, когда передача байта данных по MOSI закончена. Если установлен бит разрешения прерывания SPI (бит SPIE) в регистре SPCR, то установка флага SPIF приводит к генерации запроса на прерывание.

- **WCOL** - Write COLLision Flag - Бит конфликта записи в регистр SPDR. В бит устанавливается единица, если во время передачи данных выполняется попытка записи в регистр данных SPDR.
- с 5-ого по 1-ый бит – зарезервированные биты, их значение всегда равняется 0-лю
- **SPI2X** - Double SPI Speed Bit - Бит «двойная скорость передачи данных». Если в бит записана единица, то скорость передачи данных удвоенная. С помощью сочетания данного бита и 1-ого и 0-ого бита (SPR1, SPR0) регистра SPCR, определяют скорость передачи данных по SPI. См. рис. 17.

Регистр SPDR

Размер регистра данных SPDR, как и выше указанных — 8 бит. Данный регистр используется для передачи и чтения данных по SPI. Помещая данные в него, вы запускаете процесс передачи.

1.5.4.4 Расчет резисторов для ССИ

Между выходами сдвигового регистра и входами семисегментных индикаторов необходимо установить токоограничивающие резистор, т.к. ССИ рассчитан на меньшее напряжение, чем выдает МК. Для расчетов необходимых номиналов резисторов необходимо ознакомиться со спецификацией на диоды семисегментных индикаторов.

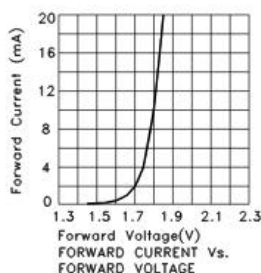


Рисунок 21 — Диаграмма зависимости тока от напряжения для светодиода

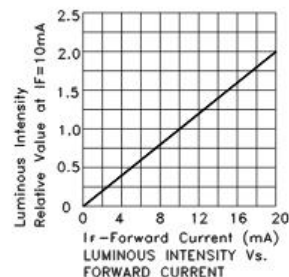


Рисунок 22 — Диаграмма зависимости яркости светодиода от тока

В предоставленной спецификации на семисегментные индикаторы рекомендуемый ток через светодиод установлен как 20мА (при данном токе обеспечивается максимальная яркость (см. рис. 22)). На рис. 21 можем наблюдать, что прямой при прямом токе в 20мА падение напряжения на диоде составляет 1.85В. Сопротивление резистора будет рассчитано по следующей формуле:

$$R = \frac{U_{out} - U_{drop}}{I_{rec}} = \frac{5V - 1.85V}{20 * 10^{-3}A} = 157.5\Omega \quad ((1))$$

Ближайшим стандартным номиналом является 180 Ом. Интересным решением является подключение резисторов не в разрыв линии от МК к светодиодам ССИ, а подтянув эту линию через резистор к питанию.

1.5.5 Схема для передачи данных на ПЭВМ

В качестве преобразователя уровней напряжения при связи устройства с ПЭВМ используется микросхема-драйвер MAX232. Схема подключения данной микросхемы представлена на рисунке 23

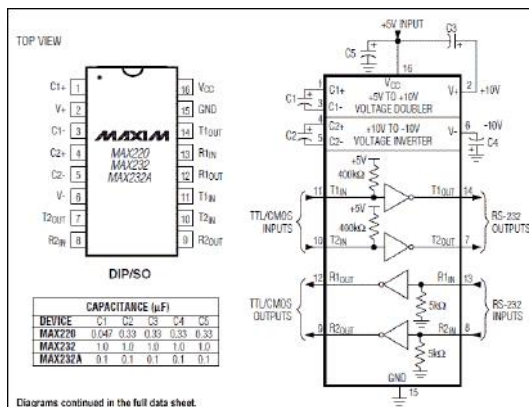


Рисунок 23 — Пример подключения драйвера MAX232

Шинные драйверы/ресиверы MAX232 предназначены для применения в RS-232 коммуникационных приложениях в жестких условиях эксплуатации. Каждый выход передатчика и вход приемника имеют защиту от электростатических разрядов до 15 кВ, обеспечивающую работу ИС без эффекта «залипания».

Передача информации на ПЭВМ производится по протоколу UART по стандарту RS-232. Скорость передачи возьмем стандартной для данного интерфейса 9600 бод. Статусный регистр UBRR рассчитывается по формуле $UBRR = (F_{OSC}/16(BAUD)) - 1 = (4000000/16(9600)) - 1 = 25$.

1.5.5.1 Описание интерфейса USART

UART (универсальный асинхронный приёмопередатчик) — одна из самых распространенных на сегодняшний день технологий передачи данных. Слово «асинхронный» означает, что интерфейс не использует линию для синхросигнала, приемник и передатчик заранее настраиваются на одну частоту.

В современных микроконтроллерах, часто вместо UART используют полностью с ним совместимый — USART (универсальный асинхронный/синхронный приёмопередатчик).

USART это более гибкий в настройки UART с дополнительными возможностями. Например в USART можно регулировать длину слова с более большим диапазоном (от 5 до 9) чем в UART (от 8 до 9). В USART как видно из названия возможна как асинхронная так и синхронная передача данных (в UART только асинхронная). При синхронной передаче помимо 2-ух лини — данные и питания, используется дополнительная линия (ХСК) с синхросигналом.

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: 300; 600; 1200; 2400; 4800; 9600; 19200; 38400; 57600; 115200; 230400; 460800; 921600 бод;

Скорость (S, бод) и длительность бита (T, секунд) связаны соотношением $T=1/S$.

Байт данных отправляются в пакетах (1-й стартовый бит перед байтом данных и 2-а стоповых бита после, количество бит опциональны) (см. рис. 24).

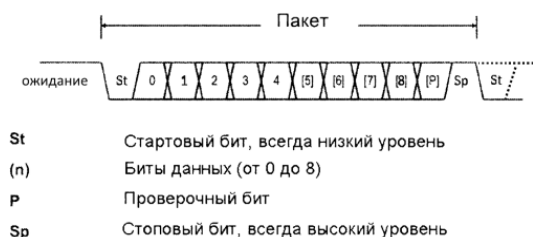


Рисунок 24 — Формат кадра UART

Для приема и передачи данных UART использует две линии данных и земля:

- передающая данные (TXD или TX);
- принимающая данные (RXD или RX);
- земля (GND).

Уровню логической единицы и нуля соответствует уровням TTL:

- 1-ца это +5В;
- 0 это 0В.

Работа с USART в AVR

Для передачи и приема данных необходимо записать или считать данные из регистра UDR. При чтении вы обращаетесь к буферу приемника, при записи к буферу передатчика, важно заметить, что при считывании состояние UDR изменяется. Это означает что считывать информацию можно только однажды.

Для управления работой с USART используются следующие регистры:

- **UCSRA** — содержит в основном флаги состояния приема/передачи данных.
- **UCSRB** — определяет какие прерывания генерировать при наступлении событий, разрешает/запрещает передачу/прием, совместно с регистром UCSRC определяет разрядность передаваемого/принимаемого слова.
- **UCSRC** — задает режим работы синхронный/асинхронный, определяет правила работы контроля данных — проверка на четность/не четность или отключено, количество стоп битов, совместно с регистром UCSRB определяет разрядность передаваемого/принимаемого слова, определяет по какому фронту принимать/передавать данные — по спадающему или по нарастающему.
- **UBRR** — определяет скорость приема/передачи данных

Регистр UCSRA

Биты	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Запись/Чтение	R	R/W	R	R	R	R	R/W	R/W	
Нач. значение	0	0	1	0	0	0	0	0	

Рисунок 25 — Регистр UCSRA

Регистр UCSRA состоит из следующих бит:

- **RXC** — флаг завершения приема, устанавливается в 1 при наличие непрочитанных данных в буфере приемник — UDR;
- **TXC** — флаг завершения передачи, устанавливается в 1 при передачи всех разрядов из передатчика — UDR;
- **UDRE** — флаг опустошения регистра передатчика, устанавливается в 1 при пустом буфере передатчика — UDR после передачи;
- **FE** — флаг ошибки кадрирования, устанавливается в 1 при обнаружение неправильного кадра, когда стоп бит равен 0-лю
- **DOR** — флаг переполнения регистра приемника, устанавливается в 1, когда байт данных принят, а предыдущий еще не прочитан из UDR;
- **PE** — флаг ошибки контроля четности, устанавливается в 1 при обнаружение ошибки контроля четности (если включена проверка);
- **U2X** — бит установки удвоенной скорости обмена, если установлена 1, то скорость передачи удваивается (частота делится на 8, а не на 16), данный бит используется только при асинхронном режиме работы;
- **MPCM** — бит мультипроцессорного обмена, если установлена 1, то контроллер аппаратно не принимает информацию, а только кадры с адресами, далее устанавливается бит завершения приема (или прерывание) и программа обрабатывает адрес, её ли это адрес. Отличие информации от адреса определяется с помощью 9-ого бита в режиме 9-и битового обмена.

Регистр UCSRB

Биты	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Запись/чтение	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Нач. значение	0	0	0	0	0	0	0	0	

Рисунок 26 — Регистр UCSRB

Регистр UCSRB состоит из следующих бит:

- **RXCIE** — бит разрешения прерывания по завершению приема, если установлена 1, то при установке флага RXC регистра UCSRA произойдет прерывание «прием завершен»;

- **TXCIE** — бит разрешения прерывания по завершению передачи, если установлена 1, то при установке флага TXC регистра UCSRA произойдет прерывание «передача завершена»;
- **UDRIE** — бит разрешения прерывания по опустошению регистра передатчика, если установлена 1, то при установке флага UDRE регистра UCSRA произойдет прерывание «регистр данных пуст»;
- **RXEN** — бит разрешения приема, при установке 1 разрешается работа приемника USART и переопределяется функционирование вывода RXD;
- **TXEN** — бит разрешения передачи, при установке 1 разрешается работа передатчика USART и переопределяется функционирование вывода TXD;
- **UCSZ2** — бит формат посылок, данный бит совместно с битами UCSZ1 и UCSZ0 регистра UCSRC определяют количество бит данных в кадрах
- **RXB8** — 9-ый разряд принимаемых данных при использовании 9-и битного режима, считывать из данного бита нужно до считывания регистра UDR;
- **TXB8** — 9-ый разряд передаваемых данных при

использование 9-и битного режима, записывать в данный бит нужно до записи в регистр UDR.

Регистр UCSRC

Бит	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Нач. значение	1	0	0	0	0	1	1	0	

Рисунок 27 — Регистр UCSRC

Регистр UCSRC состоит из следующих бит:

- **URSEL** — тут надо пояснить, это немного странный бит, он отвечает за выбор регистра UCSRC или UBRR, при установке 1 мы работаем с регистром UCSRC, при 0 мы работаем с регистром UBRR;
- **UMSEL** — бит выбора режима асинхронный или синхронный, если установлен 1 — режим синхронный (т.е. с использованием линии синхронизации ХСК), если 0 — режим асинхронный;
- **UPM1, UPM0** — биты выбора режима проверки на четность/нечетность;
- **USBS** — бит отвечающий за количество стоп-битов, если установлена 1 — два стоп-бита, если 0 — один стоп-бит;
- **UCSZ1, UCSZ0** — совместно с битом UCSZ2 регистра UCSRB определяют количество бит данных в кадрах (см. таблицу выше);
- **UCPOL** — бит полярность тактового сигнала, при синхронном режиме определяет по какому фронту принимать/передавать данные — по спадающему или по нарастающему.

Регистр UBRR

Биты	15	14	13	12	11	10	9	8	
	URSEL	-	-	-	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Чтение/запись	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Начал. значение	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Рисунок 28 — Регистр UBRR

Регистр UBRR отвечает за скорость обмена, он состоит из двух 8-и битных регистров — UBRRH и UBRRL.

- **URSEL** отвечает за выбор регистра UCSRC или UBRR, при установке 1 мы работаем с регистром UCSRC, при 0 мы работаем с регистром UBRR. Биты с UBRR0 до UBR11 устанавливают скорость передачи в бодах, но не прямую, а через следующие формулы:
- **Биты с UBRR0 до UBRR11** устанавливают скорость передачи в бодах. И их значение рассчитывается по формуле $UBRR = (f_{CK} / (BAUD * 16)) - 1$ или $UBRR = (f_{CK} / (BAUD * 8)) - 1$, где f_{CK} — тактовая частота микроконтроллера в герцах; $BAUD$ — требуемая скорость в бодах; 16 и 8 коэффициент делителя частоты, зависит от бита U2X регистра UCSRA,

1.6 Описание граф-схем алгоритмов функционирования

1.6.1 Главная программа

Опишем алгоритм работы главной программы (см. рис. 29).

Первым действием в цикле инициализируем датчики (определяем их режим работы) и передаем команду на начало конвертации температуры. Далее считываем значения TH и TL (верхняя и нижняя температурные границы) из энергонезависимой памяти. После этого производим инициализацию портов дисплея, матричной клавиатуры, UART интерфейса, инициализируем таймеры и прерывание для клавиш клавиатуры.

В бесконечном цикле если программа находится в состоянии показа температуры, то опрашиваем датчики температуры с интервалом 1 секунда, иначе программа находится в спящем режиме до наступления какого-либо прерывания.

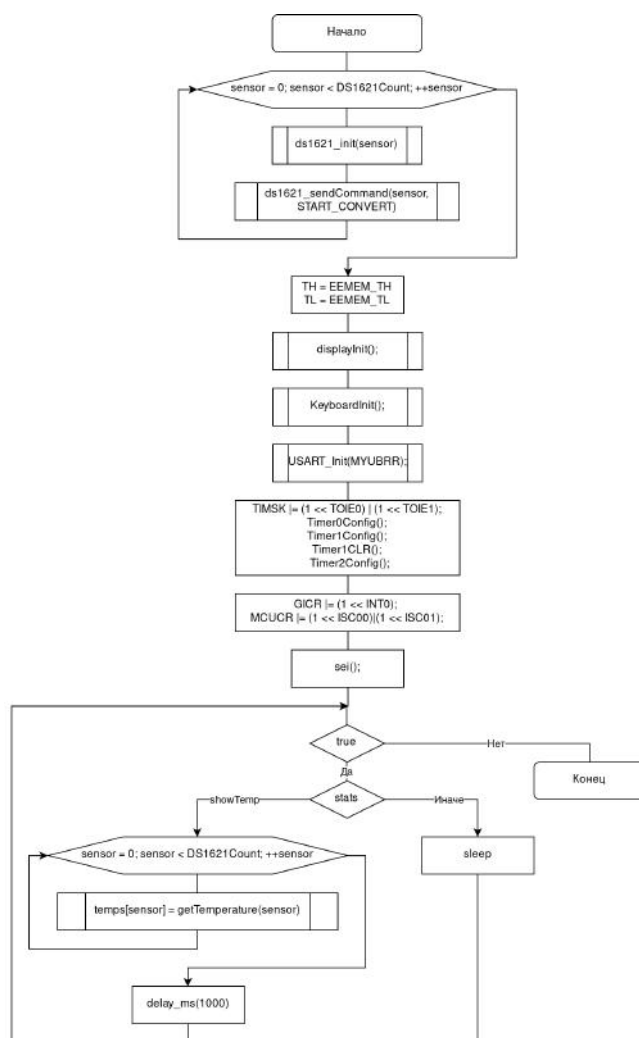


Рисунок 29 — Схема алгоритмов главной программы

1.6.2 Таймерные функции

1.6.2.1 Асинхронный вывод показаний на ССИ

Опишем алгоритм работы обработчика прерывания 8-битного таймера 0 (см. рис. 30).

Данный таймер отвечает за асинхронный вывод показаний на ССИ. Данные на ССИ необходимо выводить на дисплей с использованием прерывания не реже 60 раз в секунду, т.к. если описать вывод данных в цикле основной программы, то какое-нибудь процедура, которая выполняется в том же цикле, может исполняться слишком долго, что повлечёт "затухание" светодиодов ССИ.

В программе имеется глобальная переменная `hide`, которая используется для скрытия части информации с дисплея (эмуляция моргания дисплея в режимах изменения ТН или ТЛ). Если `hide=false`, то выводим значение на ССИ с помощью подпрограммы, иначе копируем выводимую строку, заменяем некоторые символы другим символом и выводим получившуюся строку на ССИ.

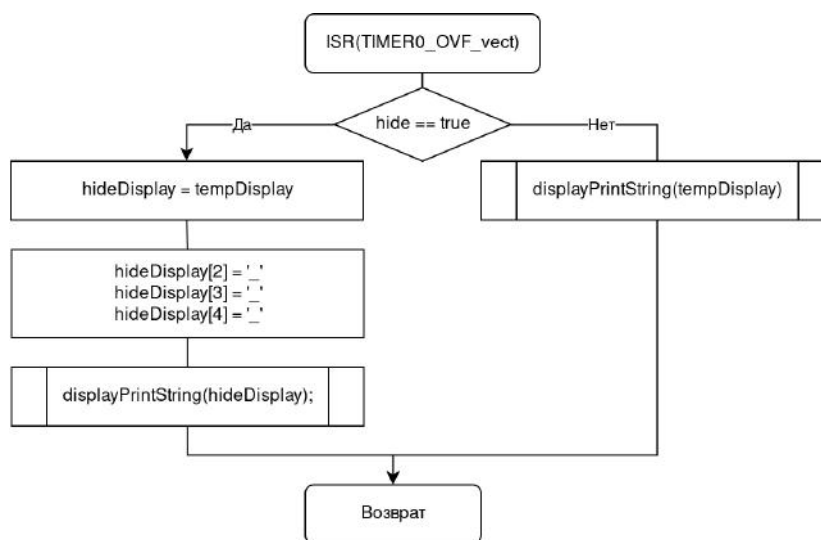


Рисунок 30 — Схема алгоритмов прерывания 8-битного таймера 0

1.6.2.2 Циклическая смена отображаемого датчика и его показаний

Опишем алгоритм работы обработчика прерывания 16-битного таймера 1 (см. рис. 31).

Данный таймер отвечает за циклическую смену отображаемого датчика и его показаний (асинхронно от чтения показаний).

В подпрограмме производится инкрементирование номера датчика до тех пор, пока показания датчика не выйдут за установленные температурные границы или номер датчика не станет равным тому, с которого началась данная подпрограмма. Производится запись показаний датчика в tempDisplay - переменную, которая хранит состояние ССИ. Производится вывод показаний в ПЭВМ и установка начального значения таймера 1.

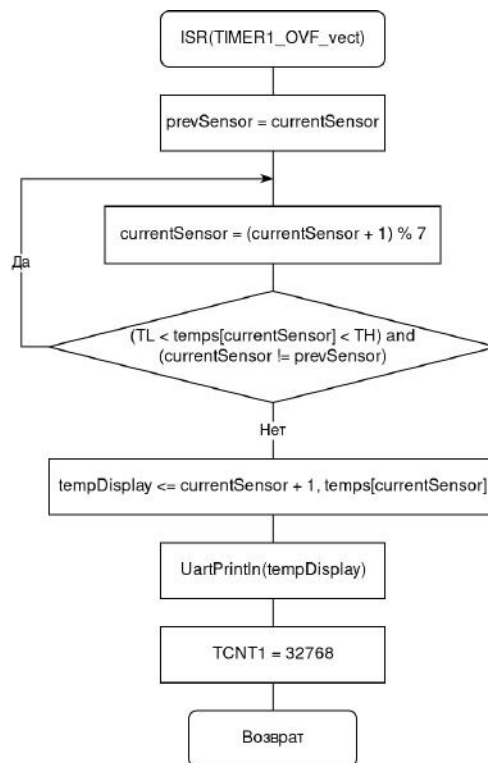


Рисунок 31 — Схема алгоритмов прерывания 16-битного таймера 1

1.6.2.3 Мерцание дисплея в режимах «Установка ТН» и «Установка ТL»

Опишем алгоритм работы обработчика прерывания 8-битного таймера 2 (см. рис. 32).

Данный таймер отвечает за "моргание" дисплея в режимах "Установка ТН" и "Установка ТL".

В программе есть статическая переменная `T2Counter`, для деления частоты работы счетчика. Когда переменная становится равной нулю, то инвертируется переменная `hide`, которая используется для скрытия части информации с дисплея (эмуляция моргания дисплея в режимах изменения ТН или ТL).

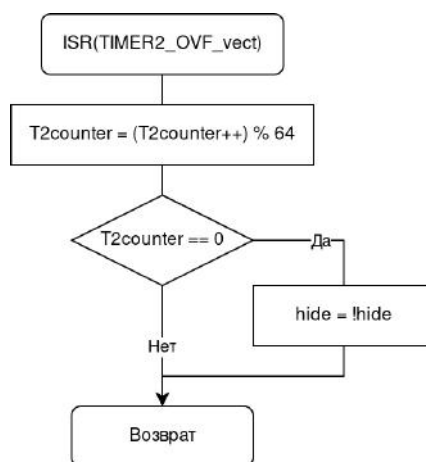


Рисунок 32 — Схема алгоритмов прерывания 8-битного таймера 0

1.6.2.4 Расчет делителей таймеров

Частота работы микроконтроллера = 4 Мгц. Все таймеры используются в режиме переполнения.

Асинхронный вывод показаний на ССИ

Вывод показаний на ССИ должен производиться не реже чем 60 раз в секунду. Используется 8-битный таймер 0.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk _{IO} (No prescaling)
0	1	0	clk _{IO} /8 (From prescaler)
0	1	1	clk _{IO} /64 (From prescaler)
1	0	0	clk _{IO} /256 (From prescaler)
1	0	1	clk _{IO} /1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Рисунок 33 — Делители частоты для таймера 0

Расчитаем предделитель таймера по следующей формуле: Делитель = $\frac{F_{OSC}}{2^8 * \text{частота}}$ = $\frac{4000000}{256 * 60}$ = 260. Ближайший предделитель равен 256 при CS02 = 1, CS01 = 0, CS00 = 0.

Проверим частоту обновления ССИ для данного предделителя Частота = $\frac{F_{OSC}}{2^8 * 256}$ = $\frac{4000000}{256 * 256}$ = 61.03. 61 раз в секунду - удовлетворительны результат для нашей задачи. В коде программы добавим данную настройку:

```
#define Timer0Config() (TCCR0 = (1 << CS02))
```

Циклическая смена отображаемого датчика и его показаний В данной курсовой работе не стоит задачи узнавать температуру как можно быстрее. Возьмем самый большой делитель для 16-битного таймера - 1024 при CS12 = 1, CS11 = 0, CS10 = 1. Проверим частоту обновления ССИ для данного предделителя Частота = $\frac{F_{OSC}}{2^{16} * \text{делитель}}$ = $\frac{4000000}{65536 * 1024}$ = 0.06. 1 раз в 16 секунду - удовлетворительны результат для нашей задачи. В коде программы добавим данную настройку:

CS12	CS11	CS10	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	clk _{IO} /1 (No prescaling)
0	1	0	clk _{IO} /8 (From prescaler)
0	1	1	clk _{IO} /64 (From prescaler)
1	0	0	clk _{IO} /256 (From prescaler)
1	0	1	clk _{IO} /1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Рисунок 34 — Делители частоты для таймера 1

```
#define Timer1Config() (TCCR1B = (1 << CS12) | (1 << CS10))
```

Мерцание дисплея в режимах «Установка ТН» и «Установка ТЛ»

Комфортной для пользователя частотой мерцания дисплея в режиме ввода границ возьмем 2Гц.

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk _{T2S} /1 (No prescaling)
0	1	0	clk _{T2S} /8 (From prescaler)
0	1	1	clk _{T2S} /32 (From prescaler)
1	0	0	clk _{T2S} /64 (From prescaler)
1	0	1	clk _{T2S} /128 (From prescaler)
1	1	0	clk _{T2S} /256 (From prescaler)
1	1	1	clk _{T2S} /1024 (From prescaler)

Рисунок 35 — Делители частоты для таймера 2

Расчитаем предделитель таймера по следующей формуле: Делитель = $\frac{F_{OSC}}{2^8 * \text{частота}}$ = $\frac{4000000}{256 * 2}$ = 7812.5. Ближайший предделитель равен 1024 при CS22 = 1, CS21 = 1, CS20 = 1.

Проверим частоту обновления ССИ для данного предделителя Частота = $\frac{F_{OSC}}{2^8 * 256}$ = $\frac{4000000}{256 * 1024}$ = 15.25. 15.25 Гц это почти в 8 раз больше нужной частоты. В коде обработки прерывания добавим счетчик до 8 и если произошел сброс этого счетчика, то меняем состояние дисплея. В коде программы добавим данную настройку:

```
#define Timer2Config() (TCCR2 = (1 << CS22) | (1 << CS21) | (1 << CS20))
```

1.6.3 Обработка сигналов от клавиатуры

Опишем алгоритм работы обработчика внешнего прерывания INT0 (см. рис. 32).

Данное прерывание поступает от пульта оператора (матрица кнопок).

Обработчик прерывания вызывает функцию опроса кнопок, которая возвращает номер нажатой кнопки. Заметим, что прерывание срабатывает на фронт сигнала кнопки, следовательно, не нужно ожидать когда кнопка будет отпущена. Алгоритм опроса кнопки основан на том, что нельзя отпустить кнопку за время обработки прерывания. Далее происходит "маршрутизация" и в зависимости от номера кнопки выполняется какое-либо действие:

- если нажата кнопка "prev" или "next" то проверяется переменная состояния на то, можно ли в данном режиме менять номер датчика, и если ответ положительный, то вы изме-



- если нажаты кнопки с номерами, соответствующим кнопкам установки верхней или нижней температурных границ, то выполняется остановка таймера 1 (отвечает за автоматическое переключение датчика, показания которого выводятся на ССИ) и запуск таймера 2 (отвечает за "моргание" дисплея). Затем переменной состояния программы присваивается значение "set TL" или "set TH" в зависимости от нажатой клавиши и обновляется переменная tempDisplay для вывода шаблона установки температурных границ.
- если нажата кнопка "Ок" то если МК находился в состоянии установки верхней или нижней температурных границ, то сохраняем границу в EEPROM, выводим показания на дисплей и запускаем таймер 1.
- если нажата кнопка "temp+" или "temp-" то проверяется переменная состояния на то, можно ли в данном режиме менять границу температуры, и если ответ положительный, то мы изменяем эту границу.

1.7 Расчет потребляемой мощности

Расчет потребляемой мощности устройства.

- а) Потребление микроконтроллера в Active режиме при нормальных условиях 5 мА.
- б) Потребление цифрового датчика температуры в Active режиме 1 мА.
- в) Потребление счетчика-дешифратора CD4017BE при 5В входного напряжения 1 мА.
- г) Собственное потребление сдвигового регистра 74595 при НУ и при 5В входного напряжения 80 мкА.
- д) Типичное потребление драйвера MAX232 при передачи данных 22 мА, но т.к. мы передаем данные не постоянно, а раз в секунду 8 символов по 8 бит на скорости 9600 бод/с, то получим потребление не более 1 мА.
- е) Ток через один светодиод равен $I = \frac{U_{out} - U_{drop}}{R} = \frac{5V - 1.85V}{180\Omega} = 17.5mA$ В данном проекте используется ССИ на 6 разрядов, но в один момент времени ток протекает только через сегменты одного разряда. В среднем у разряда будет светиться 4 сегмента из 8. Тогда мощность, потребляемая всем индикатором равна $P = 4 * 17.5mA * 5V = 350mW$
- ж) Рассеиваемая мощность на линейном стабилизаторе напряжения КР142ЕН5А равна суммарному току, проходящему через него 84 мА, умноженному на падение напряжения $12V - 5V = 7V$, и равна 588 мВт.

Таблица 4 — Потребляемая мощность регистратором температуры

Микросхема	P_{one} , мВт	Количество	P_{summ} , мВт
ATmega8	25	1	25
DS1621	5	7	35
CD4017BE	5	1	5
74HC595	0.4	1	0.4
MAX232ACPE+	5	1	5
BC56-12EWA	350	1	350
КР142ЕН5А	588	1	588
			1008,4

Суммарная потребляемая мощность регистратором температуры равна 1 Вт (см. табл. 4). Основные потребители - стабилизатор напряжения и семисегментный индикатор. Для снижения общего потребления устройства и увеличения времени автономной работы следует заменить линейный стабилизатор напряжения на DC-DC конвертер, а ССИ на монохромный ЖК дисплей.

2 Технологическая часть

2.1 Характеристики использованных систем разработки

Для проектирования и отладки регистратора температуры в качестве средств разработки использованы следующие среды:

- а) AVR Studio 4.19 - для отладки программного кода;
- б) GCC - для компиляции исходного кода на C;
- в) Proteus 8 Professional - для симуляции работы устройства.

Среда AVR Studio позволяет при компиляции программы определить процент используемой памяти микроконтроллера. Количество задействованной памяти МК представлено на рисунке 37.

```
Program:    3936 bytes (48.0% Full)
(.text + .data + .bootloader)

Data:       194 bytes (18.9% Full)
(.data + .bss + .noinit)

EEPROM:      2 bytes (0.4% Full)
(.eeprom)
```

Рисунок 37 — Занимаемая память МК ATmega8

2.1.1 Симуляция в Proteus 8

Для симуляции работы МК и датчиков построена упрощенная схема в Proteus 8 (см. рис. 38)

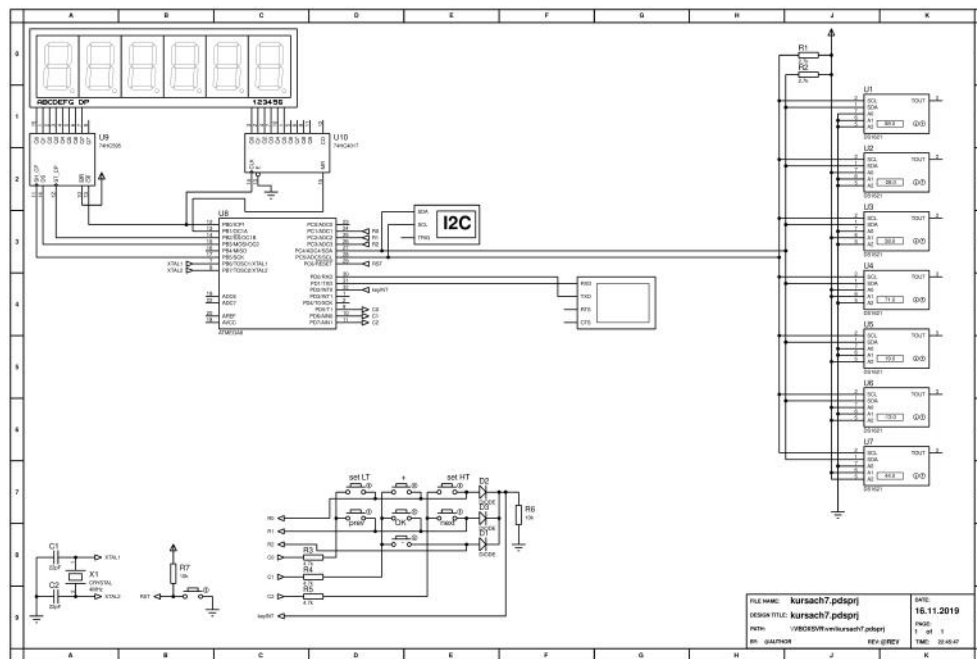


Рисунок 38 — Упрощенная схема регистратора температуры в Proteus 8

Рассмотрим более детально блоки схемы.

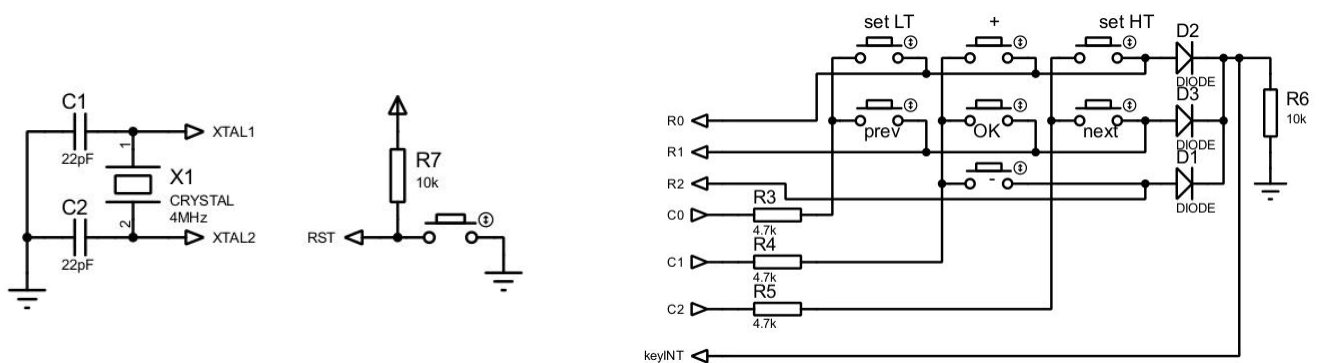


Рисунок 39 — Кнопка сброса и внешний генератор частоты

Рисунок 40 — Матрица кнопок

На рис. 39 представлен блок тактирования и сброса. Блок тактирования состоит из кварцевого резонатора на 4 МГц, шунтированного конденсаторами на 22 пФ. Блок сброса состоит из кнопки сброса, линия соединения с МК которой подтянута к линии питания через резистор.

На рис. 40 представлена матрица кнопок. Данная матрица кнопок (пульт оператора) состоит из 7 кнопок. Она позволяет проверить все основные функции устройства (установить граничные значения, сменить номер датчика).

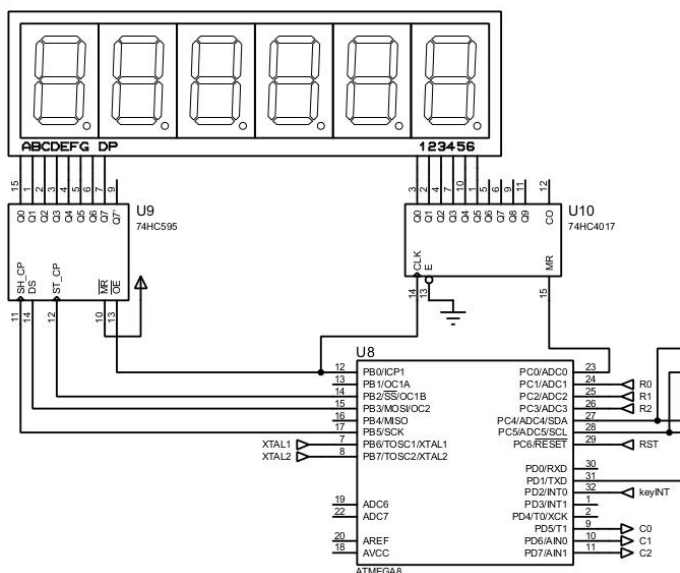


Рисунок 41 — МК и схема индикации

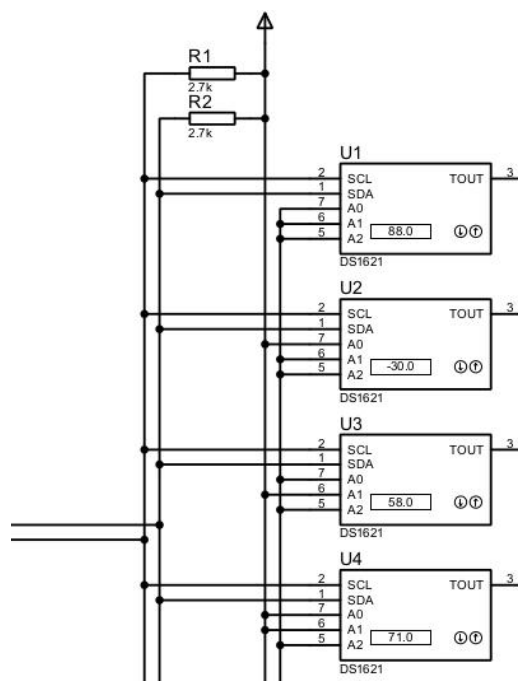


Рисунок 42 — Датчики температуры

На рис. 41 представлены МК и система индикации. Микроконтроллер - ATmega8. Система индикации - сдвиговый регистр 74595, ССИ на 6 позиций и счетчик с дешифратором 4017В. Заметим, что схема симмуляции является упрощенной и на ней не представлены токоограничивающие резисторы для светодиодов и транзисторы или транзисторные сборки для управления массивом светодиодов.

На рис. 42 представлены 4 из 7 цифровых датчиков температуры DS1621, подключенных по шине TWI. Модели датчиков температуры в среде Proteus позволяют прямо во время симмуляции изменять показания, что дает возможность в режиме реального времени убедиться, что регистратор температуры работает корректно.

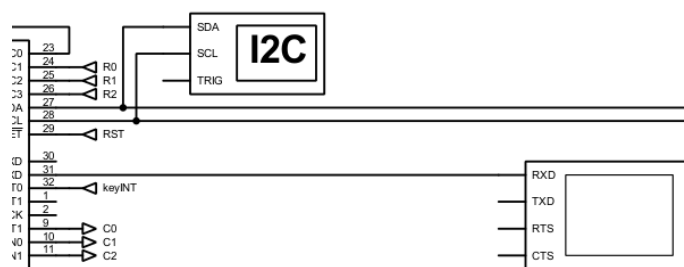


Рисунок 43 — Отладочные модули

На рис. 43 представлены отладочные модули: виртуальный терминал и отладчик шины TWI. Виртуальный терминал позволяет прочитать/записать информацию через интерфейс UART. Отладчик TWI позволяет читать пакеты, проходящие по шине в HEX формате. Заметим, что схема симмуляции является упрощенной и на ней не представлен драйвер MAX232 для конвертации уровней сигнала UART.

Для отладки использованы I2C Debugger (для просмотра обмена информацией между МК и цифровыми термометрами) и Virtual Terminal (для просмотра выводимой информации по UART) (см. рис. 44 и 45) соотв.).

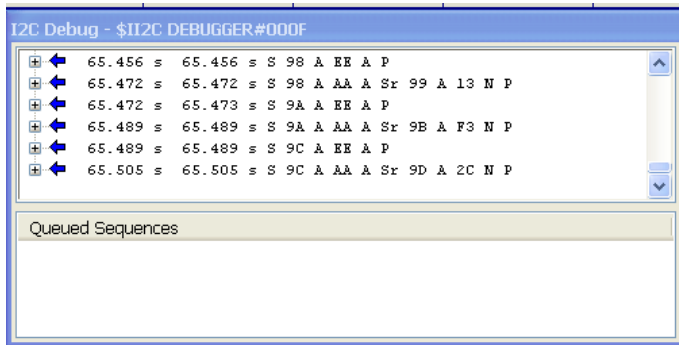


Рисунок 44 — Пример работы I2C Debugger

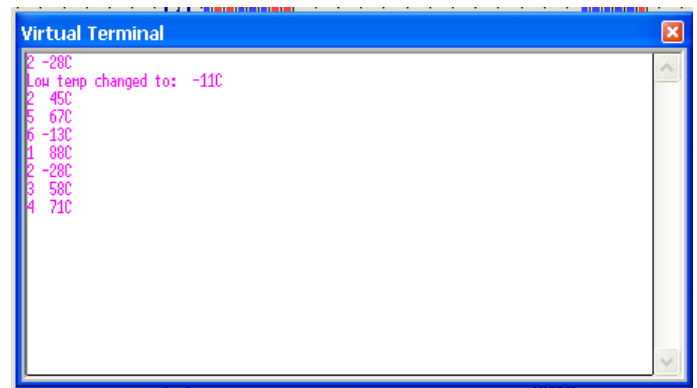


Рисунок 45 — Пример работы виртуального терминала

На рис. 44 открыт отладчик шины TWI. В данном отладчике отображается время начала передачи, время окончания передачи, стартовый бит (S), адрес передачи (0x98, 0x9A, 0x9C), подтверждение (A = ACK), повторные старт (Sr), адрес для чтения (0x99, 0x9B, 0x9D), подтверждение (A = ACK), байт данных (0x13, 0xF3, 0x2C), отсутствие подтверждения (N = NACK) и окончание передачи (p).

На рис. 45 открыт виртуальный терминал, в который выводятся считанные показания датчиков (2: 45°C; 5: 67°C; 6: -13°C и т.д.) и история изменения граничных температур (Low temp changed to: -11°C - Нижняя граница температуры изменена на -11°C).



Рисунок 46 — Пример 1 симуляции в Proteus 8 Рисунок 47 — Пример 2 симуляции в Proteus 8

На рис. 46 на ССИ отображается номер датчика (4) и его показания (+71°C). На рис. 47 вывод показаний температуры на ССИ от датчика (6) -13°C.

2.2 Способы программирования памяти МК

Самый популярный способ запрограммировать современные контроллеры - внутрисхемное программирование (ISP). Внутрисхемным данный метод называется потому, что микроконтроллер в этот момент находится в схеме целевого устройства. Для нужд программатора в этом случае выделяется несколько выводов контроллера. К этим выводам подключается прошивающий шлейф от программатора и происходит запись прошивки. После чего шнур отключается и контроллер начинает работу.

У МК AVR прошивка передается по интерфейсу SPI и для работы программатора нужно четыре линии и питание (достаточно только земли, чтобы уравнивать потенциалы земель программатора и устройства):

- MISO — данные идущие от контроллера (Master-Input/Slave-Output)
- MOSI — данные идущие в контроллер (Master-Output/Slave-Input)
- SCK — тактовые импульсы интерфейса SPI
- RESET — сигналом на RESET программатор вводит контроллер в режим программирования
- GND — земля

В качестве разъема для программирования микроконтроллера выбран Atmel 6-Pin ISP Connector (см. рис. 48).

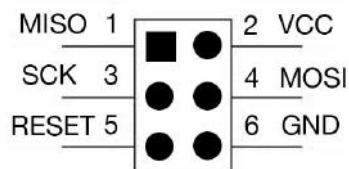


Рисунок 48 — Разъем Atmel 6-Pin ISP

2.2.1 Алгоритм последовательного программирования через SPI

При записи последовательных данных в ATmega8 данные синхронизируются по переднему фронту SCK. При чтении данных с ATmega8 данные синхронизируются по падающему фронту SCK (см. рис. 49).

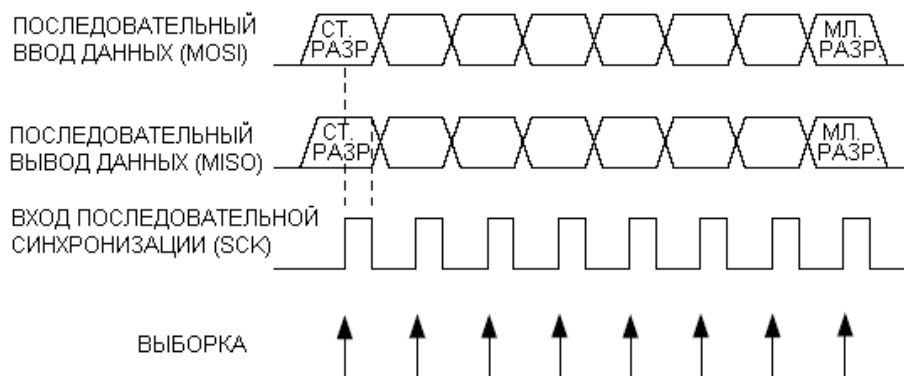


Рисунок 49 — Осциллограммы сигналов последовательного программирования интерфейса SPI

Для программирования и проверки ATmega8 в режиме последовательного программирования рекомендуется следующая последовательность действий:

- Последовательность включения: подайте питание между VCC и GND, когда RESET и SCK установлены на «0». В некоторых системах программист не может гарантировать, что SCK удерживается на низком уровне во время включения питания. В этом случае для RESET должен подаваться положительный импульс продолжительностью не менее двух тактов ЦП после того, как для SCK установлено значение «0».
- Подождите не менее 20 мс и включите последовательное программирование, отправив последовательную инструкцию ProgrammingEnable на контакт MOSI.
- Инструкции по последовательному программированию не будут работать, если связь не синхронизирована. Когда в связь синхронизирована второй байт (0x53) будет возвращаться при выдаче третьего байта команды включения программирования. Независимо от того, правильно ли ответил МК или нет, должны передаваться все четыре байта инструкции. Если 0x53 не был получен, дайте положительный импульс RESET и введите новую команду включения программирования
- Память FLASH программируется по одной странице (64 байта) за раз. Страница памяти загружается по одному байту за раз, предоставляя 5 LSB адреса и данных вместе с инструкцией загрузки страницы памяти программы. Чтобы обеспечить правильную загрузку страницы, младший байт данных должен быть загружен до того, как старший байт данных будет применен для данного адреса. Страница памяти программ сохраняется путем загрузки инструкции страницы памяти WriteProgram с 7MSB адреса.

- д) Память EEPROM программируется по одному байту за раз, предоставляя адрес вместе с соответствующей инструкцией записи. Место в памяти EEPROM автоматически стирается перед записью новых данных.
- е) Любая ячейка памяти может быть проверена с помощью инструкции чтения, которая возвращает содержимое по выбранному адресу на последовательном выходе MISO.
- ж) В конце сеанса программирования RESET можно установить на высокое значение, чтобы начать нормальную работу.
- з) Последовательность выключения (при необходимости): установите RESET на «1». Выключите питание VCC.

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 ЭСППЗУ и флэш-памяти
Чтение памяти программ	0010 H000	aaaa aaaa	bbbb bbbb	oooo oooo	Чтение старшего (H=1) или младшего (H=0) байта данных o из памяти программ по адресу a:b.
Загрузка страницы памяти программ	0100 H000	xxxx xxxx	bbbb bbbb	iiii iiii	Запись старшего (H=1) или младшего (H=0) байта данных i в страницу памяти программ по адресу b. Мл. байт данных должен быть загружен перед старшим байтом по тому же адресу.
Запись страницы памяти программ	0100 1100	aaaa aaaa	xxxx xxxx	xxxx xxxx	Запись страницы памяти программ по адресу a:b.
Чтение ЭСППЗУ	1010 0000	xxxx aaaa	bbbb bbbb	oooo oooo	Чтение данных o из ЭСППЗУ по адресу a:b.
Запись ЭСППЗУ	1100 0000	xxxx aaaa	bbbb bbbb	iiii iiii	Запись данных i в ЭСППЗУ по адресу a:b.
Чтение бит защиты	0101 1000	0000 0000	xxxx xxxx	xxoo oooo	Чтение бит защиты. "0" - запрограммирован, "1" - не запрограммирован. См. табл. 116.
Запись бит защиты	1010 1100	111x xxxx	xxxx xxxx	11ii iiii	Запись бит защиты. Запись "0" приводит к программированию бита защиты. См. табл. 116.
Чтение сигнатурного байта	0011 0000	xxxx xxxx	xxxx xxbb	oooo oooo	Чтение сигнатурного байта o по адресу b.
Запись конфигурационных бит	1010 1100	1010 0000	xxxx xxxx	iiii iiii	Указывайте "0" для программирования, "1" для стирания. См. табл. 120.
Запись старших конфигурационных бит	1010 1100	1010 1000	xxxx xxxx	iiii iiii	Указывайте "0" для программирования, "1" для стирания. См. табл. 120.
Запись расширенных конфигурационных бит	1010 1100	1010 0100	xxxx xxxx	xxxx xxii	Указывайте "0" для программирования, "1" для стирания. См. табл. 120.
Чтение конфигурационных бит	0101 0000	0000 0000	xxxx xxxx	oooo oooo	Чтение конфигурационных бит. "0" - запрограммирован, "1" - не запрограммирован. См. табл. 120.
Чтение расширенных конфигурационных бит	0101 0000	0000 1000	xxxx xxxx	oooo oooo	Чтение расширенных конфигурационных бит. "0" - запрограммирован, "1" - не запрограммирован. См. табл. 120.
Чтение старших конфигурационных бит	0101 1000	0000 1000	xxxx xxxx	oooo oooo	Чтение старших конфигурационных бит. "0" = запрограммирован, "1" = не запрограммирован. См. табл. 119.
Чтение калибровочного байта	0011 1000	xxxx xxxx	0000 00bb	oooo oooo	Чтение калибровочного байта o по адресу b.

Прим.:

a - адрес старших разрядов;
b - адрес младших разрядов;
H - 0 - мл. байт, 1 - ст. байт;
o - вывод данных;
i - ввод данных;
x - произвольное значение.

Рисунок 50 — Набор инструкций последовательного программирования через SPI

2.2.2 Опрос данных флэш-памяти

После того, как страница полностью запрограммирована во флэш-память, при чтении по адресам в пределах запрограммированной страницы возвращается \$FF. Микроконтроллер готов к записи новой страницы, если запрограммированное значение считано корректно. Это используется для определения момента, когда может быть загружена следующая страница. Обратите внимание, что запись выполняется всей страницы одновременно и любой адрес в пределах страницы может использоваться для опроса. Опрос данных флэш-памяти не действует для значения \$FF, т.к. при записи этого значения пользователь может не вводить задержку t_{WD_FLASH} (см. рис. 51)) перед программированием новой страницы. Данная возможность объясняется тем, что очищенная память микроконтроллера содержит \$FF во всех ячейках.

2.2.3 Опрос данных EEPROM

При чтении значения по адресу, который использовался для записи нового байта и последующего его программирования в ЭСППЗУ, возвращается значение \$FF. В это же время, микроконтроллер готов к записи нового байта, если запрограммированное значение корректно считывается. Это используется для определения момента, когда может быть осуществлена запись следующего байта. Данное не распространяется на значение \$FF, но программист должен обратить внимание на следующее: поскольку очищенная память заполнена \$FF по всем адресам, то программирование ячейки значением \$FF может быть пропущено. Пропуск нельзя делать, если ЭСППЗУ перепрограммируется без предварительного стирания всей памяти. В этом случае, значение \$FF нельзя использовать для опроса данных и программист должен предусмотреть задержку не менее t_{WD_EEPROM} (см. рис. 51)) перед программированием следующего байта.

Обозначение	Минимальная задержка
t_{WD_FLASH}	4.5 мс
t_{WD_EEPROM}	9.0 мс
t_{WD_ERASE}	9.0 мс

Рисунок 51 — Задержки при программировании/чтении

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта получены функциональное и принципиальное описания устройства, алгоритмы и исходные коды программ на языке С для программирования памяти микроконтроллера.

Устройство представляет из себя регистратор температуры от 7 цифровых датчиков на основе популярного 8-битного микроконтроллера ATmega8.

Устройство имеет следующие технические характеристики:

- а) работает на частоте 4 МГц;
- б) опрашивает до 7 цифровых датчиков температуры по TWI;
- в) выводит показания датчиков на цифровой семисегментный индикатор;
- г) выводит показания датчиков на ПЭВМ с помощью последовательного интерфейса (UART);
- д) обладает пультом (матрица кнопок) для изменения режимов работы;
- е) выводит показания, которые находятся выше верхней температурной границы или ниже нижней температурной границы;
- ж) работает от линии питания постоянного тока 12В.

Список использованных источников

- 1 Хартов В.Я. *Микроконтроллеры AVR. Практикум для начинающих. 2-е издание*, Издательство МГТУ им. Баумана, 2012 г. – 278с.
- 2 Хартов В.Я. *Микропроцессорные системы: учеб. пособие для студ. учреждений высш. проф. образования*, Академия, М., 2014 г. – 368с.
- 3 Atmel *ATmega8A datasheet - DOC001083586*, [Электронный ресурс] // ATmega8A datasheet - DOC001083586: электронн. документ 8159E–AVR–02/2013 URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Microchip_8bit_mcu_AVR_ATmega8A_data_sheet_40001974A.pdf (дата обращения: 3 декабря 2019 г.)
- 4 Maxim Integrated *DS1621 datasheet*, [Электронный ресурс] // DS1621 datasheet - 2737: электронн. документ 19-7540; Rev 4; 3/15 URL: <https://datasheets.maximintegrated.com/en/ds/2737.pdf> (дата обращения: 3 декабря 2019 г.)
- 5 Прокопенко В.С. *Программирование микроконтроллеров ATMEL на языке C.*, МК-Пресс, М., 2012 г. – 320с.

ПРИЛОЖЕНИЯ

Приложение А - Исходные коды программ

main.h

```
#define FOSC 4000000UL           // Clock Speed
#define Timer0Config() (TCCR0 = (1 << CS02)) // Примерно 60Гц
#define Timer1Config() (TCCR1B = (1 << CS12) | (1 << CS10)) // Меньше 1Гц
#define Timer1CLR() (TCNT1 = 32768);
#define Timer2Config() (TCCR2 = (1 << CS22) | (1 << CS21) | (1 << CS20))
#define DS1621Count 7
```

main.c

```
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/sleep.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <util/delay.h>

#include "Display.h"
#include "I2C.h"
#include "SPI.h"
#include "UART.h"
#include "ds1621.h"
#include "main.h"
#include "matrixKeyboard.h"

#define DISPLAY_LOW "L %3dC" // Формат вывода нижнего лимита
#define DISPLAY_HIGH "H %3dC" // Формат вывода верхнего лимита
// Переменные для отображения на ССИ
char tempDisplay[] = " Load ", hideDisplay[];
enum { set_TL, set_TH, showTemp } stats = showTemp; // Состояние
signed char temps[DS1621Count]; // Массив температур
unsigned char currentSensor = 0, prevSensor = 0, T2counter = 0;
signed char TH, TL; // Границы
signed char EEMEM EEMEM_TL = -10, EEMEM_TH = 50; // Границы в EEPROM
bool hide = false; // Для моргания дисплеем

// Обработчик прерывания 8-битного timer0
ISR(TIMERO_OVF_vect) {
    if (!hide) {
```

```

    displayPrintString(tempDisplay);
} else {
    strncpy(hideDisplay, tempDisplay, 6);
    hideDisplay[2] = '_';
    hideDisplay[3] = '_';
    hideDisplay[4] = '_';
    displayPrintString(hideDisplay);
}
}

// Обработчик прерывания 8-битного timer2 Моргание
ISR(TIMER2_OVF_vect) {
    T2counter = (T2counter++) % 8;
    if (T2counter == 0) {
        hide = !hide;
    }
}

// Обработчик прерывания 16-битного timer1
ISR(TIMER1_OVF_vect) {
    prevSensor = currentSensor;
    do {
        currentSensor = (currentSensor + 1) % DS1621Count;
    } while (((TL < temps[currentSensor]) && (temps[currentSensor] < TH)) &&
        (currentSensor != prevSensor));
    sprintf(tempDisplay, "%d %3dC", currentSensor + 1, temps[currentSensor]);
    UartPrintln(tempDisplay);
    Timer1CLR();
}

ISR(INT0_vect) {
    unsigned char key = which_key_pressed_unsafe();
    switch (key) {
        case 0: // Set TL
            TIMSK &= ~(1 << TOIE1); // Отключение прерывания вывода
            TIMSK |= (1 << TOIE2); // Включение моргания
            stats = set_TL;
            sprintf(tempDisplay, DISPLAY_LOW, TL);
            break;
        case 2: // Set TH
            TIMSK &= ~(1 << TOIE1); // Отключение прерывания вывода
            TIMSK |= (1 << TOIE2); // Включение моргания
            stats = set_TH;
            sprintf(tempDisplay, DISPLAY_HIGH, TH);
            break;
        case 1: // temp+
        case 7: // temp-
    }
}

```

```

switch (stats) {
    case set_TL:
        TL += (key + 1) % 8 - 1;
        sprintf(tempDisplay, DISPLAY_LOW, TL);
        break;
    case set_TH:
        TH += (key + 1) % 8 - 1;
        sprintf(tempDisplay, DISPLAY_HIGH, TH);
        break;
    default:
        break;
}
break;
case 4: // ok
    switch (stats) {
        case set_TL:
            eeprom_write_byte(&EEMEM_TL, TL);
            UartPrint("Low temp changed to:");
            sprintf(hideDisplay, " %3dC", TL);
            UartPrintln(hideDisplay);
            break;
        case set_TH:
            eeprom_write_byte(&EEMEM_TH, TH);
            UartPrint("High temp changed to:");
            sprintf(hideDisplay, " %3dC", TH);
            UartPrintln(hideDisplay);
            break;
        default:
            break;
    }
    stats = showTemp;
    Timer1CLR();
    TIMSK &= ~(1 << TOIE2);
    hide = false;
    TIMSK |= (1 << TOIE1);
case 3: // prev
case 5: // next
    if (stats == showTemp) {
        currentSensor = (currentSensor + (DS1621Count - 4) + key) % DS1621Count;
        sprintf(tempDisplay, "%d %3dC", currentSensor + 1,
            temps[currentSensor]);
        Timer1CLR();
        UartPrintln(tempDisplay);
    }
    break;
}
}

```

```

int main() {
    for (unsigned char sensor = 0; sensor < DS1621Count; ++sensor) {
        ds1621_init(sensor);
        ds1621_sendCommand(sensor, START_CONVERT);
    }

    TH = eeprom_read_byte(&EEMEM_TH);
    TL = eeprom_read_byte(&EEMEM_TL);

    displayInit();
    KeyboardInit();
    USART_Init(MYUBRR);
    i2c_init();

    TIMSK |= (1 << TOIE0) | (1 << TOIE1);
    Timer0Config(); // Таймер обновления семисегментного индикатора
    Timer1Config(); // Таймер для смены датчика
    Timer1CLR();
    Timer2Config(); // Таймер для моргания дисплея

    GICR |= (1 << INTO); // Разрешаем прерывание INTO
    MCUCR |= (1 << ISC00) |
        (1 << ISC01); // Генерация сигнала по возрастающему фронту PD2

    sei(); // Разрешить прерывания

    while (1) {
        switch (stats) {
            case showTemp:
                for (unsigned char sensor = 0; sensor < DS1621Count; ++sensor) {
                    temps[sensor] = getTemperature(sensor);
                }
                _delay_ms(1000);
                break;
            default:
                sleep_enable();
                sleep_cpu();
                break;
        }
    }
    return 0;
}

```

Display.h

```
#define DISP4DIG7SEG_PORT PORTB
#define DISP4DIG7SEG_DDR DDRB
#define DISP4DIG7SEG_RST PB1

//Инициализация адресных выходов для семисегментика
void displayInit();
void setAddress(unsigned int);
void displayPrintString(char*);
```

Display.c

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

#include "Display.h"
#include "HC595.h"

//Инициализация адресных выходов для семисегментика
void displayInit() {
    DISP4DIG7SEG_DDR |= 1 << DISP4DIG7SEG_RST; // Разрешаем запись в пин сброса
    HC595Init(); // Инициализируем сдвиговый регистр
}

void displayPrintString(char* chars) {
    HC5950EHigh(); // Снятие сигнала с выхода HC595
    DISP4DIG7SEG_PORT |= 1 << DISP4DIG7SEG_RST; // сброс разряда дисплея
    DISP4DIG7SEG_PORT &= ~(1 << DISP4DIG7SEG_RST); // окончание сброса
    HC595_sendSymbol(chars[0]);
    HC5950ELow(); // Установка сигнала на выходах HC595
    _delay_ms(1);
    for (unsigned int digit_number = 1; digit_number < 6; ++digit_number) {
        HC5950EHigh(); // Снятие сигнала с выхода HC595
        HC595_sendSymbol(chars[digit_number]);
        HC5950ELow(); // Установка сигнала на выходах HC595
        _delay_ms(1); // задержка для протейуса
    }
}
```


HC595.h

```
#define HC595_PORT PORTB
#define HC595_DDR DDRB
#define HC595_NOT_OE_POS PBO // Store pin (Not OE) pin location

#define HC595OEHigh() (HC595_PORT |= (1 << HC595_NOT_OE_POS))
#define HC595OELow() (HC595_PORT &= ~(1 << HC595_NOT_OE_POS))

void HC595Init();
void HC595_sendSymbol(unsigned int);
```

HC595.c

```
#include <avr/io.h>

#include "HC595.h"
#include "SPI.h"

// Инициализация сдвигового регистра HC595
void HC595Init() {
    HC595_DDR |= (1 << HC595_NOT_OE_POS);
    SPI_Init();
}

// Краткая ASCII таблица
const char table[96] = {
    0x00, 0x86, 0x22, 0x49, 0x2D, 0x6B, 0x53, 0x46, 0x70, 0x0F, 0x63, 0x46,
    0x80, 0x40, 0x80, 0x52, 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
    0x7F, 0x6F, 0x09, 0x58, 0x58, 0x48, 0x4C, 0xA7, 0x5D, 0x77, 0x7C, 0x39,
    0x5E, 0x79, 0x71, 0x7B, 0x74, 0x30, 0x0E, 0x75, 0x38, 0x55, 0x37, 0x5C,
    0x73, 0x67, 0x49, 0x6D, 0x78, 0x1C, 0x3E, 0x7E, 0x76, 0x6E, 0x5B, 0x02,
    0x64, 0x39, 0x23, 0x08, 0x02, 0x77, 0x7C, 0x58, 0x5E, 0x79, 0x71, 0x7B,
    0x74, 0x30, 0x0E, 0x75, 0x38, 0x55, 0x54, 0x5C, 0x73, 0x67, 0x50, 0x6D,
    0x78, 0x1C, 0x3E, 0x7E, 0x76, 0x6E, 0x5B, 0x46, 0x30, 0x70, 0x41, 0x00};

void HC595_sendSymbol(unsigned int symbol) {
    symbol -= (symbol >= 0x61) ? 32 : 0; // сдвиг для прописных букв
    // сдвиг для удаления непечатных символов
    symbol -= (symbol <= 0x20) ? symbol : 0x20;
    // SPI_Send_byte(table[symbol]); // Общий катод
    SPI_Send_byte(~table[symbol]); // Общий анод
}
```

SPI.h

```
#define SPI_PORT PORTB
#define SPI_DDR DDRB
#define SPI_MOSI PB3
#define SPI_SS PB2
#define SPI_SCK PB5

void SPI_Init(void);
void SPI_Send_byte(char);
```

SPI.c

```
#include <avr/io.h>

#include "SPI.h"

void SPI_Init(void) {
    SPI_DDR |= (1 << SPI_SS) | (1 << SPI_MOSI) | (1 << SPI_SCK);
    SPI_PORT |= (1 << SPI_SS); // Установить "1" на линии SS
    SPCR = (1 << MSTR) | (1 << SPE); // Режим мастер / Включить SPI
    SPSR = (1 << SPI2X); // Удвоение частоты  $F=F_{osc}/2$ 
}

void SPI_Send_byte(char data) {
    SPI_PORT &= ~(1 << SPI_SS); // Установить "0" на линии SS
    SPDR = data; // Отправить байт
    while (!(SPSR & (1 << SPIF)))
        ; // Дождаться окончания передачи
    SPI_PORT |= (1 << SPI_SS); // Установить "1" на линии SS
}
```

DS1621.h

```
#define DS1621_W 0x90
#define DS1621_R 0x91

#define READ_TEMP 0xAA
#define READ_COUNTER 0xA8
#define READ_SLOPE 0xA9
#define START_CONVERT 0xEE
#define STOP_CONVERT 0x22
#define ACCESS_TH 0xA1
#define ACCESS_TL 0xA2
#define ACCESS_CONFIG 0xAC

void ds1621_init(unsigned char);
void ds1621_sendCommand(unsigned char, unsigned char);
void ds1621_writeSingleByte(unsigned char, unsigned char, unsigned char);
unsigned char ds1621_readValue(unsigned char, unsigned char);
signed char getTemperature(unsigned char);
```

DS1621.c

```
#include <util/delay.h>

#include "DS1621.h"
#include "I2C.h"

// Инициализация датчика в состояние 0x03
void ds1621_init(unsigned char address) {
    ds1621_writeSingleByte(address, ACCESS_CONFIG, 0x03);
}

// Отправка 1 байта данных в датчик
void ds1621_writeSingleByte(unsigned char address, unsigned char command,
                             unsigned char data) {
    if (i2c_start() != 0) {
        i2c_stop();
        return;
    }

    address = (address % 0x08) << 1; // Маскирование адреса
    if (i2c_start_wait(address | DS1621_W) != 0) {
        i2c_stop();
        return;
    }

    if (i2c_write(command) != 0) {
```

```

    i2c_stop();
    return;
}

if (i2c_write(data) != 0) {
    i2c_stop();
    return;
}

i2c_stop();
}

// Отправка команды в датчик
void ds1621_sendCommand(unsigned char address, unsigned char command) {
    if (i2c_start() != 0) {
        i2c_stop();
        return;
    }

    address = (address % 0x08) << 1; // Маскирование адреса
    if (i2c_start_wait(address | DS1621_W) != 0) {
        i2c_stop();
        return;
    }

    if (i2c_write(command) != 0) { // Успешная отправка команды?
        i2c_stop();
        return;
    }

    i2c_stop();
}

// Чтение байта данных из датчика
unsigned char ds1621_readValue(unsigned char address, unsigned char value) {
    if (i2c_start() != 0) {
        i2c_stop();
        return (0);
    }

    address = (address % 0x08) << 1; // Маскирование адреса
    if (i2c_start_wait(address | DS1621_W) != 0) {
        i2c_stop();
        return (0);
    }

    if (i2c_write(value) != 0) {

```

```

    i2c_stop();
    return (0);
}

if (i2c_rep_start() != 0) {
    i2c_stop();
    return (0);
}

if (i2c_start_wait(address | DS1621_R) != 0) {
    i2c_stop();
    return (0);
}

unsigned char data = i2c_readNak();
i2c_stop();
return (data);
}

// Запрос температуры у датчика
signed char getTemperature(unsigned char address) {
    ds1621_sendCommand(address, START_CONVERT);
    _delay_ms(10); // Время ожидания по даташиту
    return ds1621_readValue(address, READ_TEMP);
}

```

I2C.h

```
#define FOSC 4000000UL // Clock Speed
#define SCL_CLK 250000L

#define BITRATE(TWSR) \
    ((FOSC / SCL_CLK) - 16) / (2 * pow(4, (TWSR & ((1 << TWPS0) | (1 << TWPS1)))))
// SCL_CLK = F_cpu/(16 + 2* TWBR* 4^TWPS)

void i2c_init(void);
unsigned char i2c_start(void);
unsigned char i2c_rep_start(void);
unsigned char i2c_start_wait(unsigned char);
unsigned char i2c_write(unsigned char);
unsigned char i2c_readAck(void);
unsigned char i2c_readNak(void);
void i2c_stop(void);
```

I2C.c

```
#include <util/twi.h>

#include "I2C.h"
#include "math.h"

void i2c_init(void) { TWBR = BITRATE(TWSR = 0x00); };

unsigned char i2c_start(void) {
    TWCR = (1 << TWINT) | (1 << TWSTA) |
            (1 << TWEN); // Прерывание, Старт, Включение
    while (!(TWCR & (1 << TWINT)))
        ;
    return ((TWSR & 0xF8) == TW_START) ? 0 : 1; // Старт произошел?
}

unsigned char i2c_rep_start(void) {
    TWCR = (1 << TWINT) | (1 << TWSTA) |
            (1 << TWEN); // Прерывание, Старт, Включение
    while (!(TWCR & (1 << TWINT)))
        ;
    return ((TWSR & 0xF8) == TW_REP_START) ? 0 : 1;
}

unsigned char i2c_start_wait(unsigned char address) {
    unsigned char STATUS;
    STATUS = ((address & 0x01) == 0) ? TW_MT_SLA_ACK : TW_MR_SLA_ACK;
    TWDR = address;
```

```

    TWCR = (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)))
        ;
    return ((TWSR & 0xF8) == STATUS) ? 0 : 1;
}

unsigned char i2c_write(unsigned char data) {
    TWDR = data;
    TWCR = (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)))
        ;
    return ((TWSR & 0xF8) != TW_MT_DATA_ACK) ? 1 : 0;
}

unsigned char i2c_readAck(void) {
    TWCR = (1 << TWEA) | (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)))
        ;
    return TWDR;
}

unsigned char i2c_readNak(void) {
    TWCR = (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)))
        ;
    return TWDR;
}

void i2c_stop(void) {
    TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
    while (TWCR & (1 << TWSTO))
        ;
}

```

matrixKeyboard.h

```
#define KeyRow_PORT PORTC
#define KeyRow_DDR DDRC
#define KeyRow_PIN PINC
#define KeyRowStart PC1
#define KeyRowCount 3
#define KeyCol_PORT PORTD
#define KeyCol_DDR DDRD
#define KeyColStart PD5
#define KeyColCount 3
#define KeyRowCNT ((1 << KeyRowCount) - 1)
#define KeyColCNT ((1 << KeyColCount) - 1)

void KeyboardInit(void);
unsigned char which_key_pressed_unsafe();
```

matrixKeyboard.c

```
#include <avr/io.h>

#include "matrixKeyboard.h"

void KeyboardInit() {
    KeyCol_DDR |= (1 << KeyColStart) * KeyColCNT;
    KeyCol_PORT |= (1 << KeyColStart) * KeyColCNT;
    KeyRow_DDR &= ~(1 << KeyRowStart) * KeyRowCNT;
    KeyRow_PORT &= ~(1 << KeyRowStart) * KeyRowCNT;
}

unsigned char which_key_pressed_unsafe() {
    for (unsigned char col = 0; col < KeyColCount; ++col) {
        KeyCol_PORT &= ~(1 << KeyColStart) * KeyColCNT;
        KeyCol_PORT |= (1 << (col + KeyColStart));
        // Если в строке хотя бы 1 кнопка нажата, то вычисляем конкретную
        if (KeyRow_PIN & ((1 << KeyRowStart) * KeyRowCNT)) {
            for (unsigned char row = 0; row < KeyRowCount; ++row) {
                if (KeyRow_PIN & (1 << (row + 1))) {
                    KeyCol_PORT |= (1 << KeyColStart) * KeyColCNT;
                    return (row * KeyColCount + col);
                }
            }
        }
    }
    KeyCol_PORT |= (1 << KeyColStart) * KeyColCNT; // Восстановить 1 на колонках
    return (KeyColCNT * KeyRowCNT); // Вернуть максимум если не нашли клавишу
}
```


UART.h

```
#define BAUD 9600
#define MYUBRR FOSC / 16 / BAUD - 1

void UartPrintln(char*);
void USART_Init(unsigned int);
void USART_Transmit(char);
void UartPrint(char*);
```

UART.c

```
#include <avr/io.h>

#include "UART.h"

void USART_Init(unsigned int ubrr) {
    // Конфигурация скорости
    UBRRH = (unsigned char)(ubrr >> 8);
    UBRRL = (unsigned char)ubrr;
    // 8 бит данных, 1 стоп бит, без контроля четности
    UCSRC = (1 << URSEL) | (1 << UCSZ1) | (1 << UCSZ0);
    UCSRB = (1 << TXEN);
}

void USART_Transmit(char data) {
    while (!(UCSRA & (1 << UDRE)))
        ; // Дождаться пустого буф.
    UDR = data;
}

void UartPrint(char* chars) {
    int i = 0;
    while (chars[i] != 0x00) {
        USART_Transmit(chars[i++]);
    }
}

void UartPrintln(char* chars) {
    UartPrint(chars);
    UartPrint("\n\r");
}
```

Приложение Б - Спецификация радиоэлементов схемы

Листов 2

Формат		Зона	Поз.	Обозначение	Наименование	Кол	Приме- чение	
					Документация			
A3					Схема электрическая функциональная	1		
A3					Схема электрическая принципиальная	1		
					Схемы интегральные			
				DD1	АТmega8-16PU	1		
				DD2	KP142EH5A	1		
				DD3	74HC595N	1		
				DD4	CD4017BE	1		
				DD5	MAX232ACPE+	1		
				DD6	ULN2803A	1		
				DD7 — DD13	DS1621	7		
					Устройства индикационные			
				HG1, HG2	BC56-12EWA	2		
					Конденсаторы			
				C1, C2	Конденсаторы неполярные 22 пкФ	2		
				C3	Конденсаторы полярные 100 мкФ	1		
				C4	Конденсаторы полярные 10 мкФ	1		
				C5 — C9	Конденсаторы полярные 1 мкФ	5		
					Разъёмы			
				XP1	DS-201	1		
				XP2	IDC-06MS	1		
				XP3	RS232 DB9	1		
				XP4 — XP17	B4B-ZR	14		
Подп. и дата					Многоканальный регистратор температуры			
		Изм.	Лист	№ докум	Подпись	Дата	Спецификация радиоэлементов схемы	
	Разраб.	Векшин Р.Д.						
	Пров.	Хартов В.Я.						
	Н.контр							
	Утв.							
Инф. № подл.						Литера	Лист	Листов
						Т	1	2
						МГТУ им. Н.Э. Баумана, Группа ИУ6-72Б		

[illegible]