МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт кибербезопасности и цифровых технологий
Кафедра КБ-6 «Приборы и информационно-измерительные системы»

КУРСОВАЯ РАБОТА

по дисциплине Методы и средства автоматизации проектирования интеллектуальных измерительных устройств

Тема курсовой работы Разработка и отладка встраиваемого программного обеспечения интеллектуального измерительного устройства

Студента _____ Д.А. Косяков
дата, подпись инициалы и фамилия

Группа БПБО-01-22 шифр 22Б0945

Обозначение работы КР–02068717–12.03.01–КБ-6–19–23

Работа защищена на оценку _____

Руководитель курсовой работы _____ С.А. Канаев
дата, подпись инициалы и фамилия

Члены комиссии _____ О.В. Москаленко
подпись инициалы и фамилия

_____
подпись инициалы и фамилия

Работа представлена к защите «___» _____20__ г.

Допущен к защите «___» _____20__ г.

Москва 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ "МИРЭА - РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ"

ИНСТИТУТ КИБЕРБЕЗОПАСНОСТИ И ЦИФРОВЫХ ТЕХНОЛОГИЙ
КАФЕДРА КБ-6 «ПРИБОРЫ И ИНФОРМАЦИОННО-ИЗМЕРИТЕЛЬНЫЕ СИСТЕМЫ»

УТВЕРЖДАЮ

Заведующий кафедрой __КБ-6__

_____ /А.Б. Снедков/
подпись ФИО

«___»_____ 20___г.

ЗАДАНИЕ

на выполнение курсовой работы по дисциплине

“Методы и средства автоматизации проектирования интеллектуальных измерительных устройств”

Студент Косяков Денис Александрович
Ф.И.О.

шифр студенческого билета 22Б0945 группа БПБО-01-22

1. Тема: Разработка и отладка встраиваемого программного обеспечения интеллектуального измерительного устройства

2. Срок сдачи студентом законченной работы **01 июня 2023г.**

3. Исходные данные для проектирования Разработать управляющую программу на языке Си для устройства на базе микроконтроллера с ядром AVR. Тип микроконтроллера ATmega32. Базовая платформа – EasyAVR v7 Development System фирмы microElectronika. Наименование: термометр на базе платинового термопреобразователя сопротивления. Диапазон измерений от минус 200 до 600 °С. Отображаемые параметры: температура (с разрешающей способностью 0.1 °С). Обеспечить возможность отображения температуры в различных единицах измерений (не менее 3-х). Тип индикатор – графический ЖКИ (ME-GLCD 128x64). Тип сенсорной панели управления – ME-TOUCH SCREEN. Обеспечить возможность считывания показаний прибора по интерфейсу USB (технология “виртуальный COM - порт”). Протокол обмена Modbus-RTU. Размещение параметров в адресном пространстве Modbus выполнить самостоятельно

Руководитель работы _____ / Канаев С.А. /
подпись Ф.И.О.

Задание принял к исполнению **30 марта 2023г.**

Студент _____ /_____/
подпись Ф.И.О.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Датчики температуры широко используются во многих областях науки и техники, начиная от простых бытовых приборов и заканчивая сложными устройствами в промышленности и научных исследованиях. Назначение датчиков температуры - определение температуры объектов и окружающей среды с высокой точностью и скоростью.

Актуальность работы обусловлена тем, что в промышленности все чаще применяются датчики температуры, которые являются частью автоматизированной системы управления технологическим процессом [1-2].

Цель работы – разработать программное обеспечение для интеллектуального датчика температуры на базе платинового термопреобразователя сопротивления.

Проектируемое устройство обладает следующими особенностями:

- диапазон измерений от минус 200 до 600 °C с разрешающей способностью 0.1 °C;
- показания прибора отображаются на графическом индикаторе в трех единицах измерения (°C, °F, K). Смена отображаемой единицы измерения происходит при нажатии на соответствующую виртуальную кнопку на сенсорной панели управления;
- с помощью протокола Modbus RTU возможно как считывать измеренное значение температуры сразу в трех единицах измерения, так и изменять такие параметры устройства как адрес Modbus RTU и скорость передачи данных.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- разработать структурную схему и обобщенный алгоритм работы программы;

- разработать алгоритм вычисления измеряемого параметра;
- написать программное обеспечение на языке С;
- произвести отладку программного обеспечения с помощью лабораторного стенда [3].

1 СТРУКТУРНАЯ СХЕМА И ОБОБЩЕННЫЙ АЛГОРИТМ РАБОТЫ

1.1 Структурная схема

На рисунке 1 представлена структурная схема проектируемого устройства.

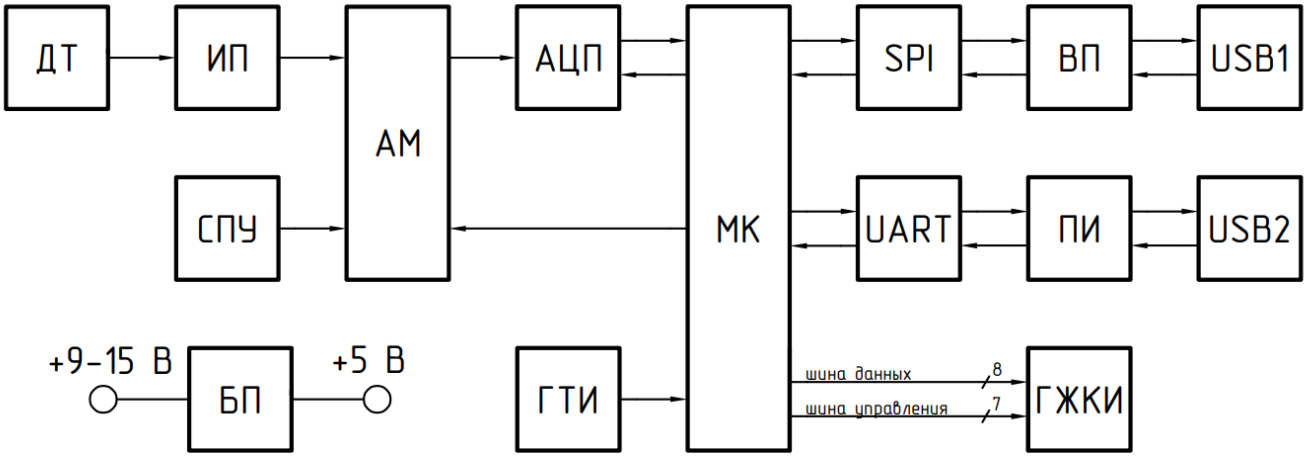

Рисунок 1 – Структурная схема устройства

На рисунке 1 обозначены следующие блоки:

- блок питания (БП);
- датчик температуры (ДТ);
- измерительный преобразователь (ИП);
- сенсорная панель управления (СПУ);
- аналоговый мультиплексор (АМ);
- аналогово-цифровой преобразователь (АЦП);
- генератор тактовых импульсов (ГТИ);
- микроконтроллер ATmega 32[4] (МК);
- модуль SPI (SPI).
- внутрисхемный программатор (ВП);

– модуль UART (UART);
– преобразователь интерфейсов (ПИ);
– интерфейсы USB (USB1, USB2).

Блок питания формирует опорное напряжение величиной пять вольт для всех электронных блоков устройства.

После включения питания устройства датчик температуры (ДТ) формирует первичный сигнал, который прямо пропорционально зависит от температуры. Данный сигнал преобразуется с помощью измерительного преобразователя (ИП) таким образом, что при температуре минус 200 °С напряжение на выходе ИП было равно нулю вольт, а при температуре 600 °С напряжение на выходе ИП было равно напряжению питания аналого-цифрового преобразователя (АЦП). АЦП преобразует аналоговый сигнал, полученный после ИП, в двоичный код. Обработка двоичного кода производится микроконтроллером (МК), который тактируется с помощью внешнего генератора тактовых импульсов (ГТИ).

Сенсорная панель управления формирует собственный аналоговый сигнал, который оцифровывается с помощью АЦП и анализируется с помощью АЦП. Выбор сигнала, который преобразуется в двоичный код, производится микроконтроллером с помощью аналогового мультиплексора (АМ).

Загрузка программного обеспечения в память микроконтроллера производится с помощью внутрисхемного программатора (ВП), который подключен к интерфейсу USB (USB1) со стороны компьютера и к интерфейсу SPI со стороны МК.

После проведения измерения температуры, полученные данные хранятся в памяти микроконтроллера и выводятся на графический жидкокристаллический индикатор (ГЖКИ), который управляется по параллельной шине.

Микроконтроллер также обрабатывает пакеты протокола Modbus RTU [5]. При запросе данных о температуре, адресе устройства или скорости передачи данных, микроконтроллер формирует пакет протокола Modbus RTU с соответствующими данными и передает его в модуль UART. Преобразователь интерфейсов (ПИ) конвертирует интерфейс UART в интерфейс USB (USB2), после чего ведущее устройство принимает пакет и обрабатывает его.

1.1.1 Базовая аппаратная платформа

В данной курсовой работе для отладки встроенного программного обеспечения используется аппаратная платформа Easy AVR v7 Development System фирмы microElectronika.

На рисунке 2 представлена плата аппаратной платформы.

Рисунок 2 – Плата аппаратной платформы Easy AVR v7 Development System фирмы microElectronika

1.1.2 Микроконтроллер

Ядром платы аппаратной платформы является микроконтроллер ATMega32 с ядром AVR в корпусе PDIP40 [4, с.304]. Данный микроконтроллер подключается в панель с маркировкой DIP40B. На рисунке 3 приведена схема выводов микроконтроллера.

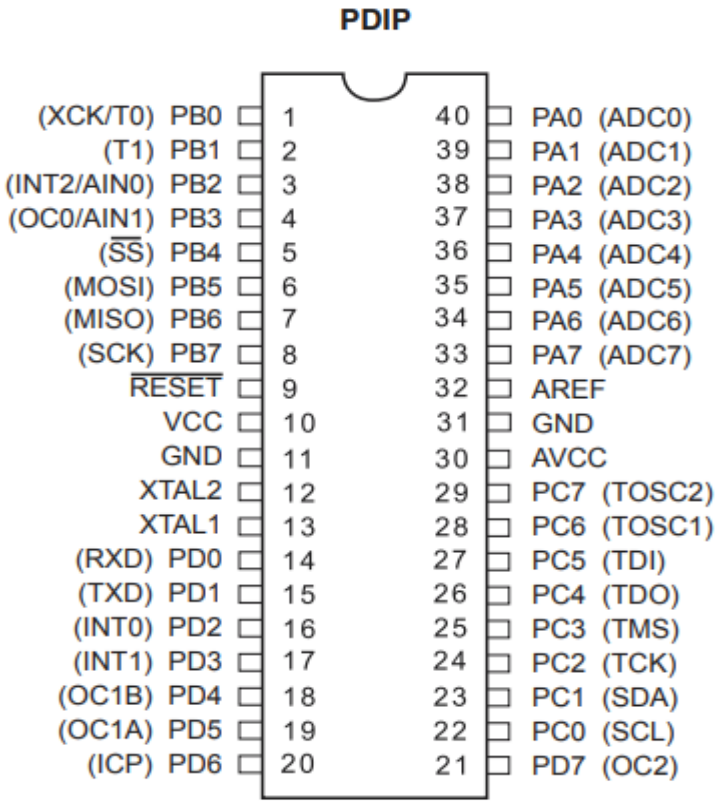

Рисунок 3 – Схема выводов микроконтроллера ATMega32 в корпусе PDIP40

Технические характеристики аппаратных блоков микроконтроллера представлены в таблице 1.

Микроконтроллер выполняет следующие функции:

– обработка измерительного сигнала, полученного с помощью внутреннего АЦП;
– вывод показаний прибора на индикатор;
– формирование пакетов Modbus RTU, необходимые для связи прибора с АСУ ТП.

Таблица 1 – Некоторые технические характеристики микроконтроллера

Параметр	Значение
Тип памяти для программ	Flash
Размер памяти для программ	32 Кб
Размер ОЗУ	2 Кб
Максимальная тактовая частота	16 МГц
Размер EEPROM	1 Кб
Таймеры общего назначения	2, 8 бит; 1, 16 бит
АЦП	8 каналов, 10 бит
Блоки интерфейсов	SPI, USART, I2C
Напряжение питания	от 1.8 до 5.5 В
Диапазон рабочих температур	от минус 40 до 85 °С

1.1.3 Блок питания

На рисунке 4 представлена схема блока питания лабораторного стенда [6, с. 6].

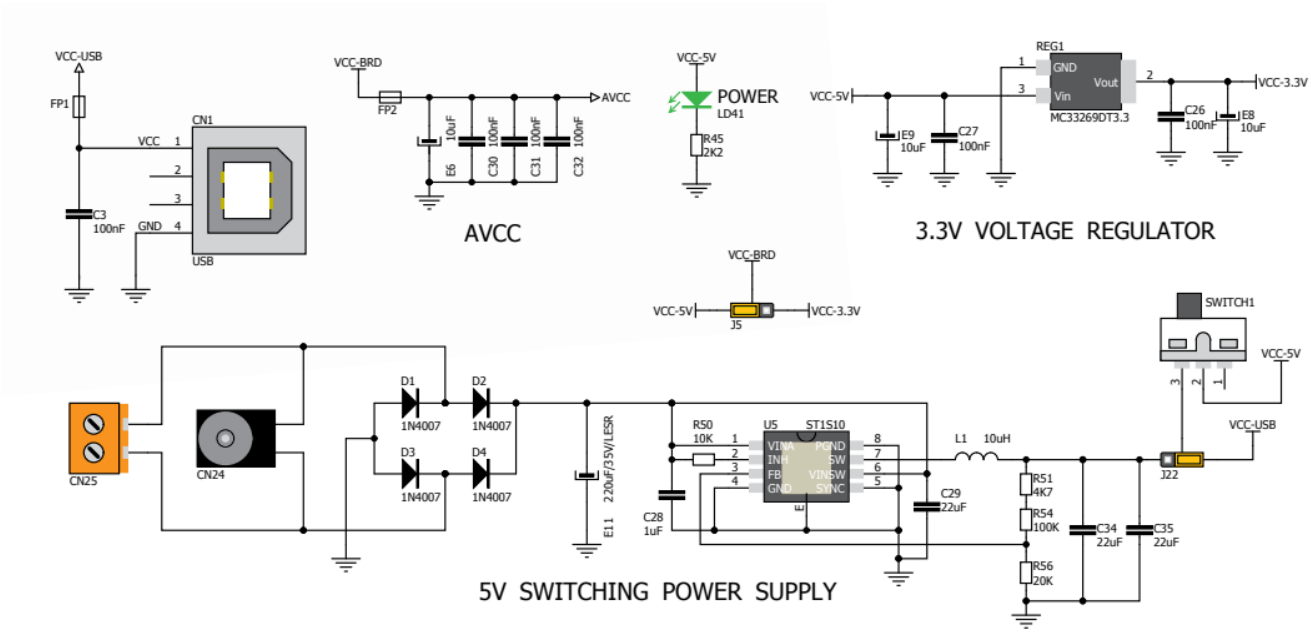

Рисунок 4 – Схема блока питания аппаратной платформы

Блок питания формирует напряжение питания для всех электронных блоков лабораторного стенда.

Выбор внешнего источника напряжения настраивается с помощью перемычки J22. Переключатель SWITCH1 включает питание лабораторного стенда

Внешними источниками напряжения могут быть:

– Шина USB (разъем CN1);

– Источник постоянного напряжения с выходным напряжением от 9 до 15 В;

– Источник переменного напряжения с выходным напряжением от 7 до 12 В.

Стоит отметить, что любые манипуляции, связанные с изменением положения переключателей и подключением внешних аппаратных блоков, следует производить при выключенном блоке питания.

1.1.4 Блок формирования тестового аналогового сигнала

На рисунке 5 представлен блок формирования тестового аналогового сигнала [6, с. 26].

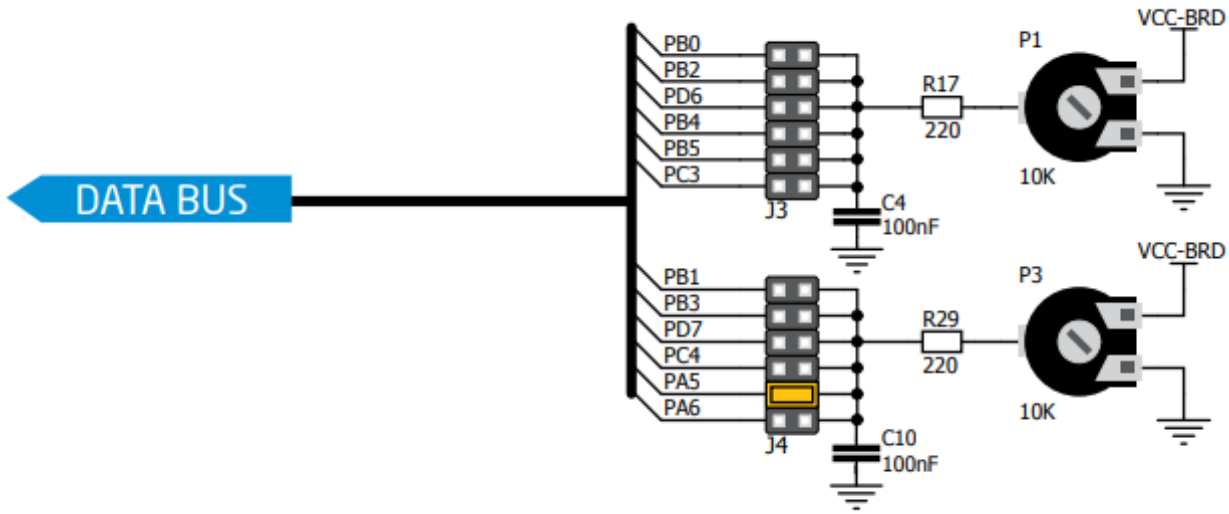

Рисунок 5 – Блок формирования тестового аналогового сигнала

Поскольку произвести тестирование прибора на всем диапазоне температур не представляется возможным, в качестве измерительного преобразователя температуры используется переменное сопротивление. При таком подходе величина температуры, отображаемая на индикаторе, зависит только от угла поворота ручки переменного сопротивления.

Выход тестового аналогового сигнала подключен к выводу PA5 микроконтроллера, который в свою очередь соединен с пятым каналом АЦП.

1.1.5 Генератор тактовых импульсов

Для работы микроконтроллера требуется источник тактовых сигналов. В данной курсовой работе микроконтроллер тактируется генератором тактовых импульсов с частотой 8 МГц.

На рисунке 6 представлена схема генератора тактовых импульсов [6, с.8].

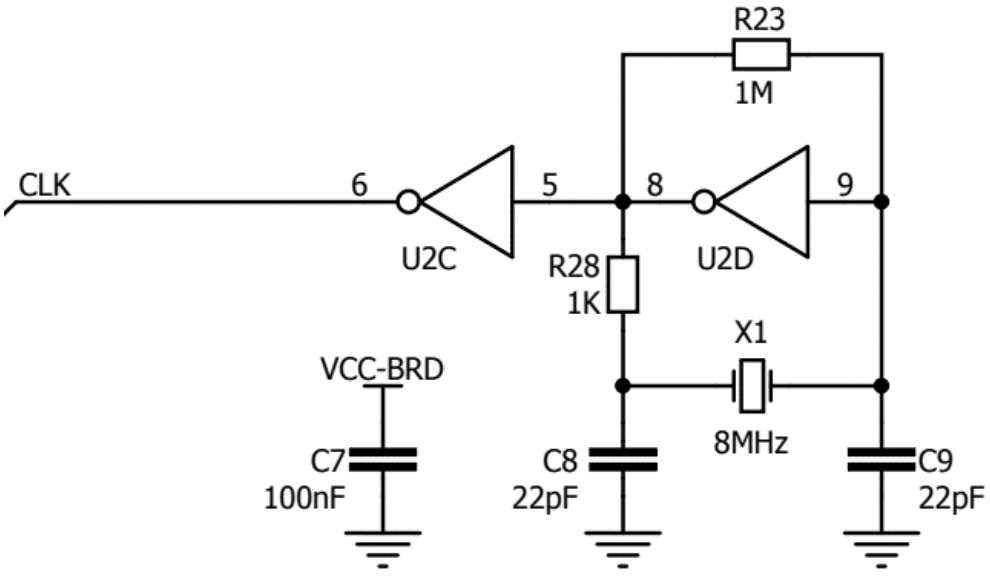

Рисунок 6 – Генератор тактовых прямоугольных импульсов

Выход генератора тактовых импульсов должен быть подключен к выводу XTAL1 микроконтроллера. Подключение генератора тактовых импульсов зависит от комбинации соответствующей группы перемычек. На рисунке Рисунок 7 представлена конфигурация перемычек тактового генератора.

Рисунок 7 – Конфигурация перемычек генератора тактовых импульсов

1.1.6 Блок связи по интерфейсу USB

На рисунке 8 представлена схема блока связи по интерфейсу USB [6, с.15].

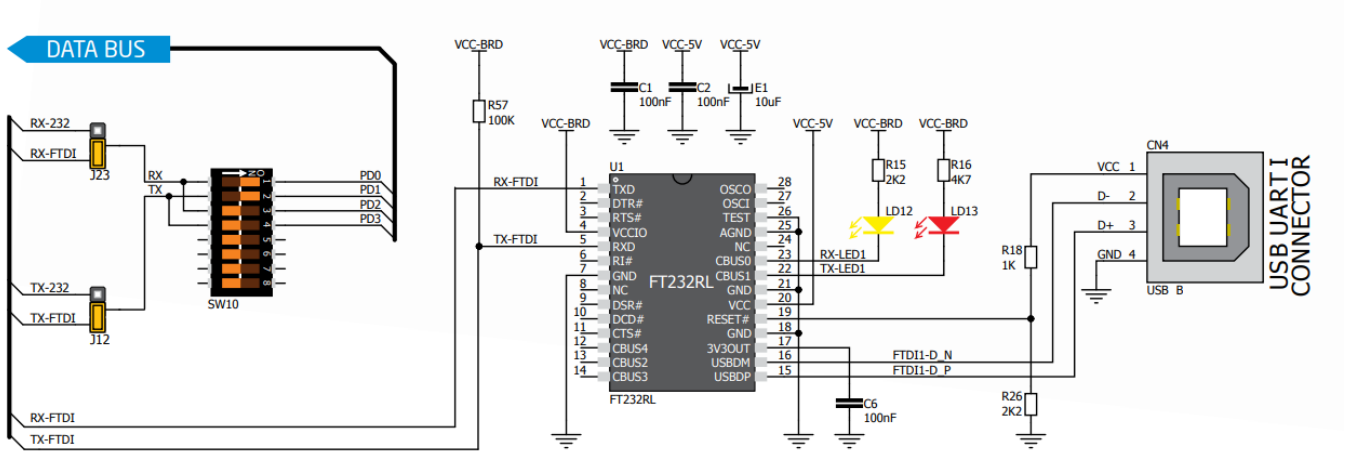

Рисунок 8 – Блок связи по интерфейсу USB

Микросхема U1 обеспечивает преобразование пакетов интерфейса USART, которые формируются соответствующим блоком микроконтроллера, в пакеты интерфейса USB.

Подключение микросхемы U1 к микроконтроллеру настраивается с помощью перемычек J12, J23 и группы переключателей SW10 в соответствии с рисунком 7.

1.1.7 Внутрисхемный программатор

Для загрузки программного обеспечения в микроконтроллер используется внутрисхемный программатор mikroProg.

Загрузка программного обеспечения в микроконтроллер производится с помощью программы AVR Flash [7].

На рисунке 9 представлен блок внутрисхемного программатора [6, с. 11].

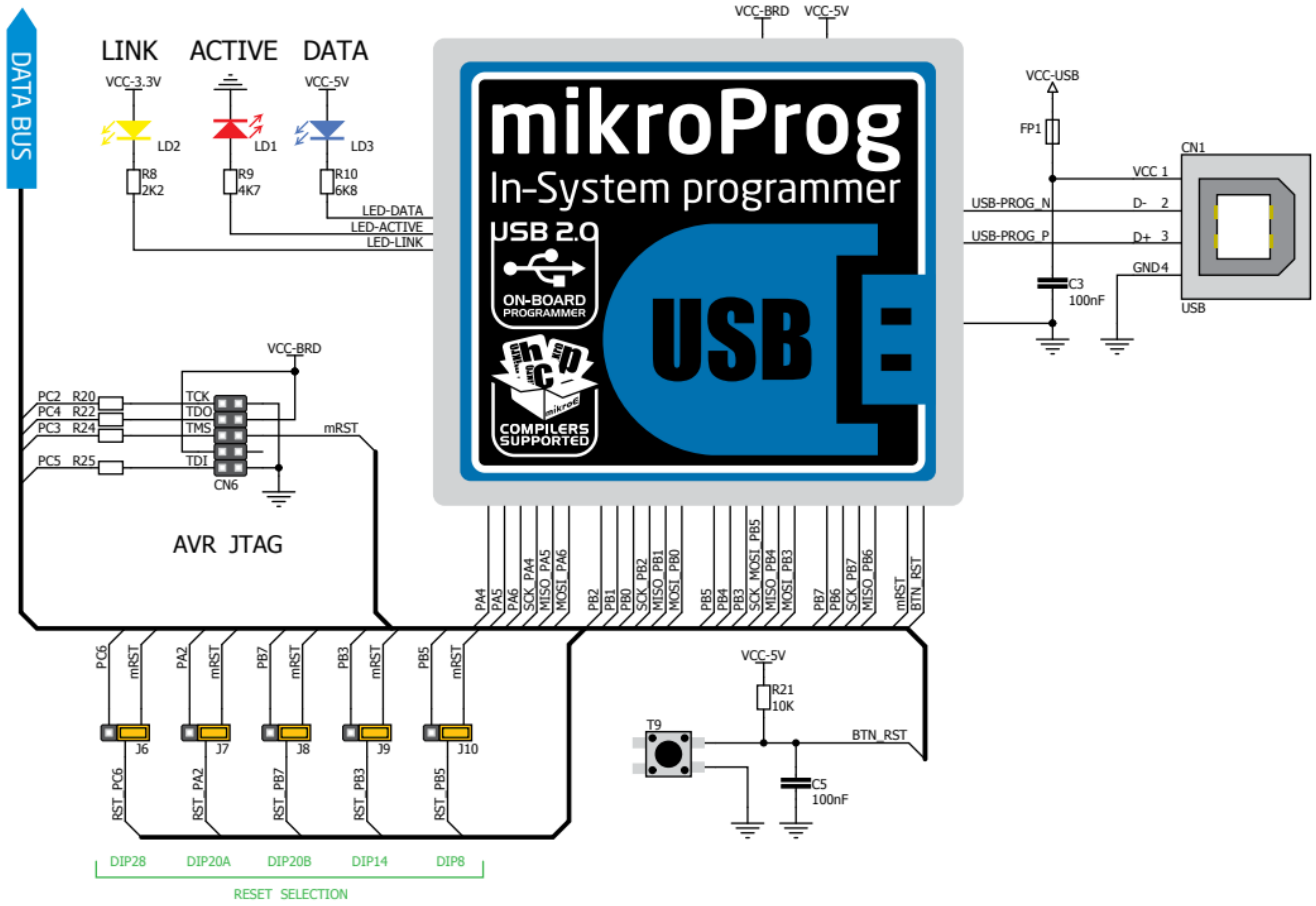

Рисунок 9 – Блок внутрисхемного программатора

1.1.8 Графический индикатор

Для отображения показаний прибора используется графический индикатор с разрешением 128x64, управляемый контроллером NT7108 [8].

На рисунке 10 представлен блок графического индикатора.

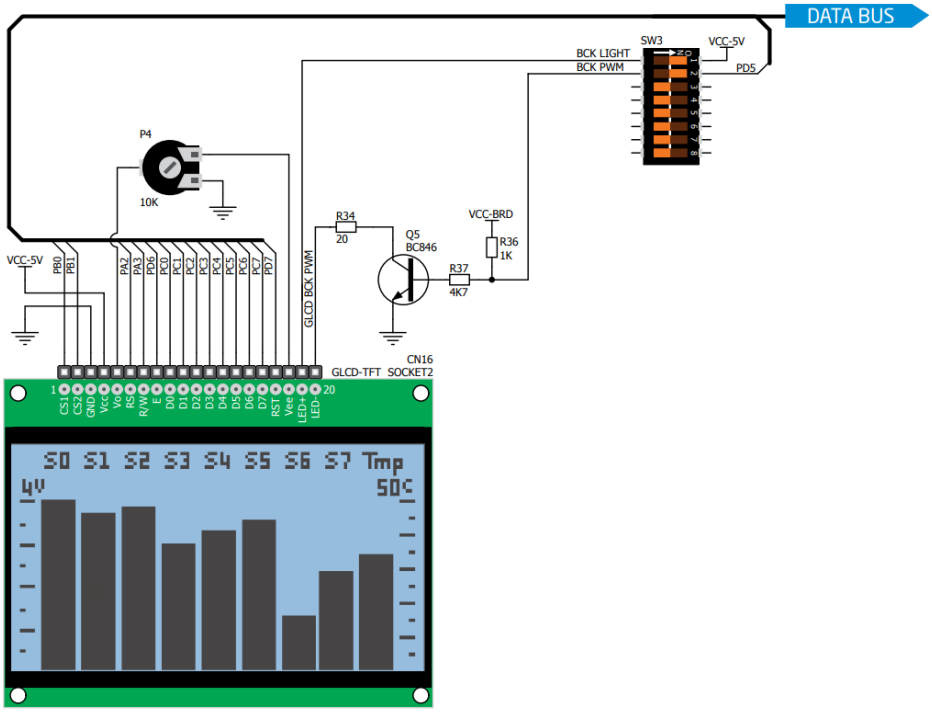

Рисунок 10 – Блок графического индикатора

В таблице 2 приведено описание управляющих выводов графического индикатора.

Управление контрастностью подсветки производится с помощью переменного сопротивления P4. Управление яркостью производится с помощью ШИМ сигнала, поступающего с вывода PD5 микроконтроллера. Если управление яркостью не требуется, следует перевести переключатель BCK_PWM в левое положение. В таком случае яркость индикатора будет максимальной.

Таблица 2 – Описание управляющих выводов индикатора

Вывод	Описание
D0 – D7	Данные
R/W	Выбор направления данных: – запись при R/W = 1; – чтение при R/W = 0.
CS1, CS2	Выбор контроллера
RS	Выбор типа передаваемых данных: – данные при RS = 1; – инструкции при RS = 0.
E	Сигнал, стробирующий передачу данных
RST	Сброс

Для записи данных в индикатор необходимо выбрать чип, который управляет видео памятью. Выставить логические уровни линий R/W, RS в соответствии с документацией. На линиях D0 – D7 выставить логические уровни данных. В таблице 3 приведен пример команды, которая управляет подсветкой индикатора

Таблица 3 – Команда управления питанием подсветки

RS	**R/W**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
0	0	0	0	1	1	1	1	1	D

Если логический уровень линии D0 равен 1, то включается питание подсветки. Иначе питание подсветки выключается.

Полный перечень команд приведен в технической документации [8]

1.1.9 Сенсорная панель управления

Графический индикатор поставляется вместе с резистивной панелью управления. На рисунке 11 представлен блок сенсорной панели управления.

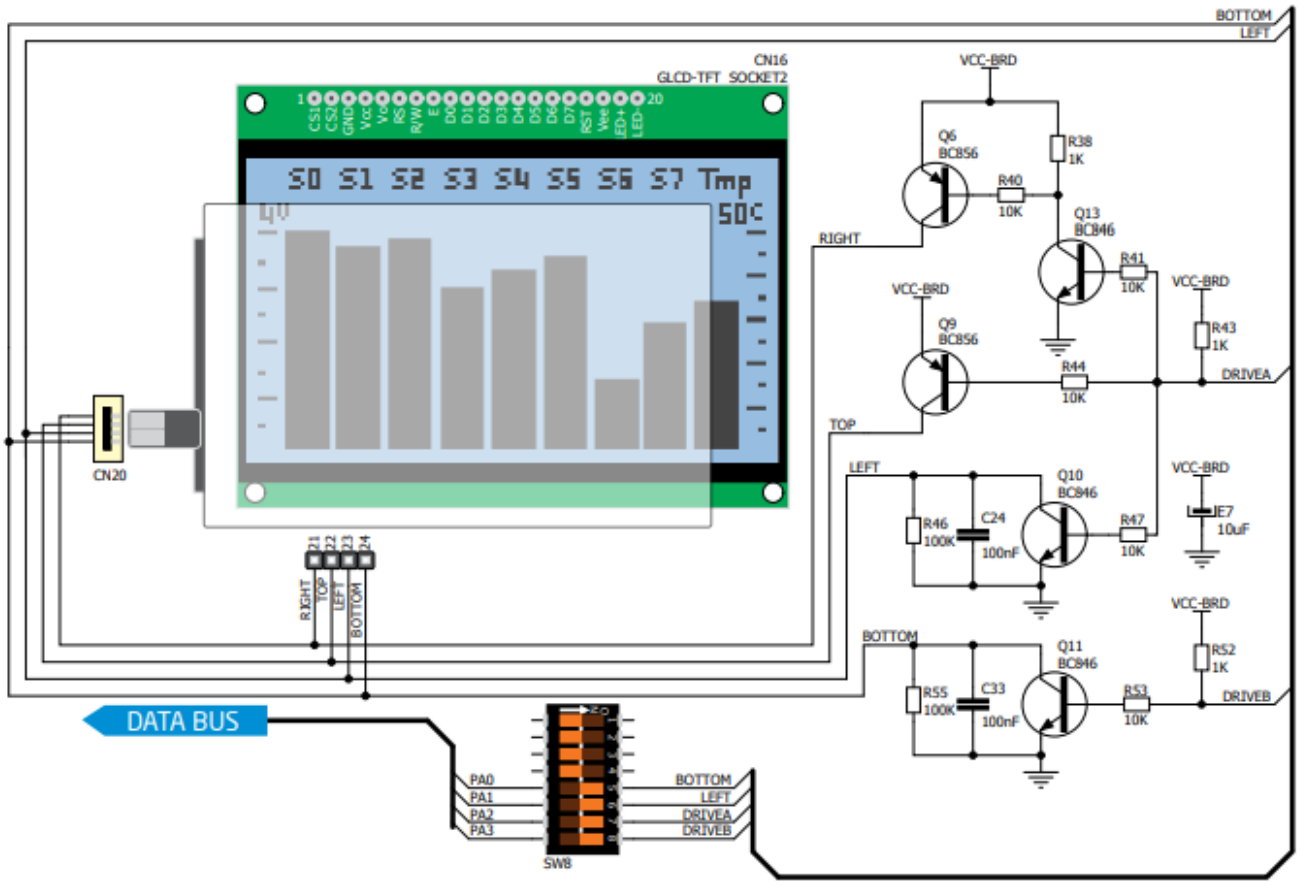

Рисунок 11 – Блок сенсорной панели управления

На рисунке 12 представлена структура резистивной сенсорной панели управления.

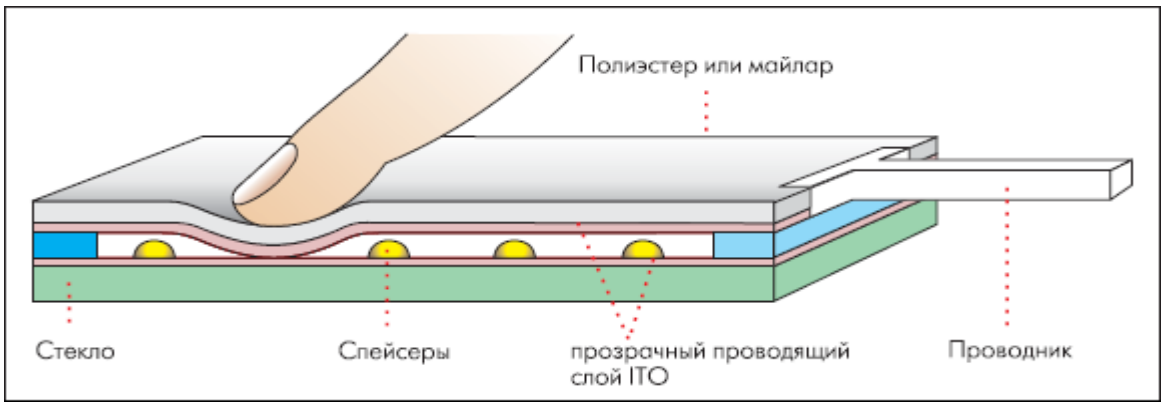

Рисунок 12 – Структура сенсорной панели управления

У резистивной панели есть два слоя, которые обладают нормированным сопротивлением электрическому току. В исходном состоянии эти слои не касаются друг друга. При физическом нажатии на панель происходит замыкание верхнего и нижнего слоев в точке нажатия. Таким образом образуется два делителя напряжения – один вдоль оси X, второй вдоль оси Y.

На рисунке 13 изображены делители напряжения, которые образуются при нажатии на сенсорную панель.

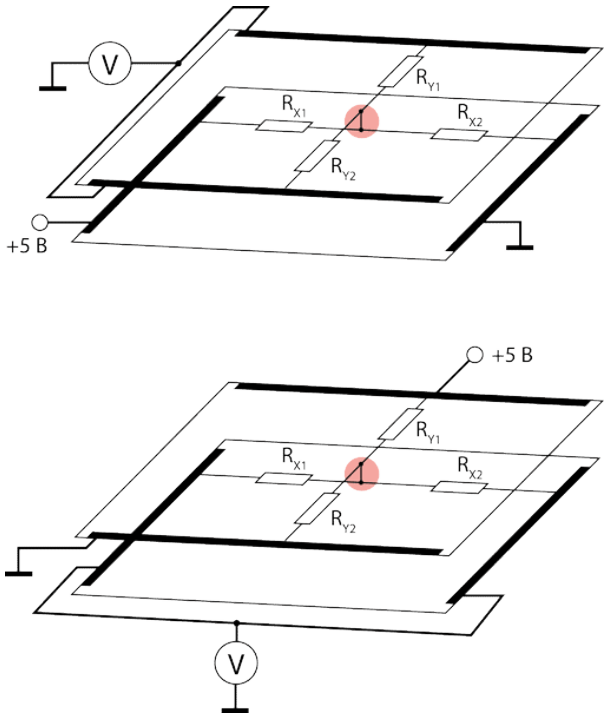

Рисунок 13 – Делители напряжения, образующиеся при нажатии на сенсорную панель управления

Чтобы считать абсциссу точки касания необходимо на нижнюю пластину подать напряжение питания. В таком случае потенциал верхней пластины станет равен потенциалу в точке соединения сопротивлений R_{x1} и R_{x2}. Следовательно, при измерении напряжения на соответствующем выводе панели с помощью АЦП, получим абсциссу точки нажатия. Считывание ординаты точки касания происходит аналогично.

На рисунке 14 представлен алгоритм считывания точки касания.

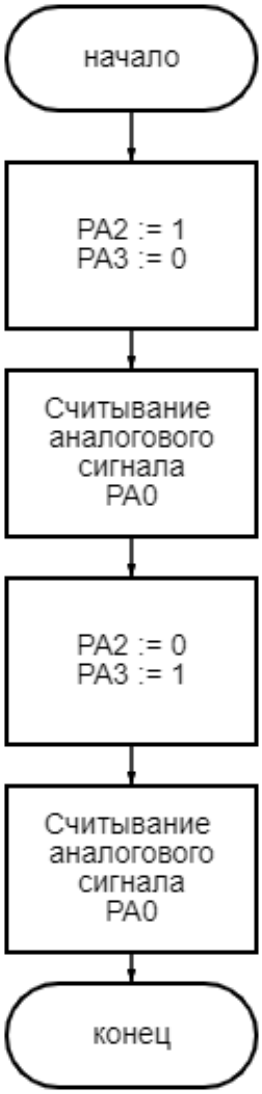

Рисунок 14 – Алгоритм считывания координат точки касания

1.2 Обобщенный алгоритм работы

На рисунке 15 представлен обобщенный алгоритм работы.

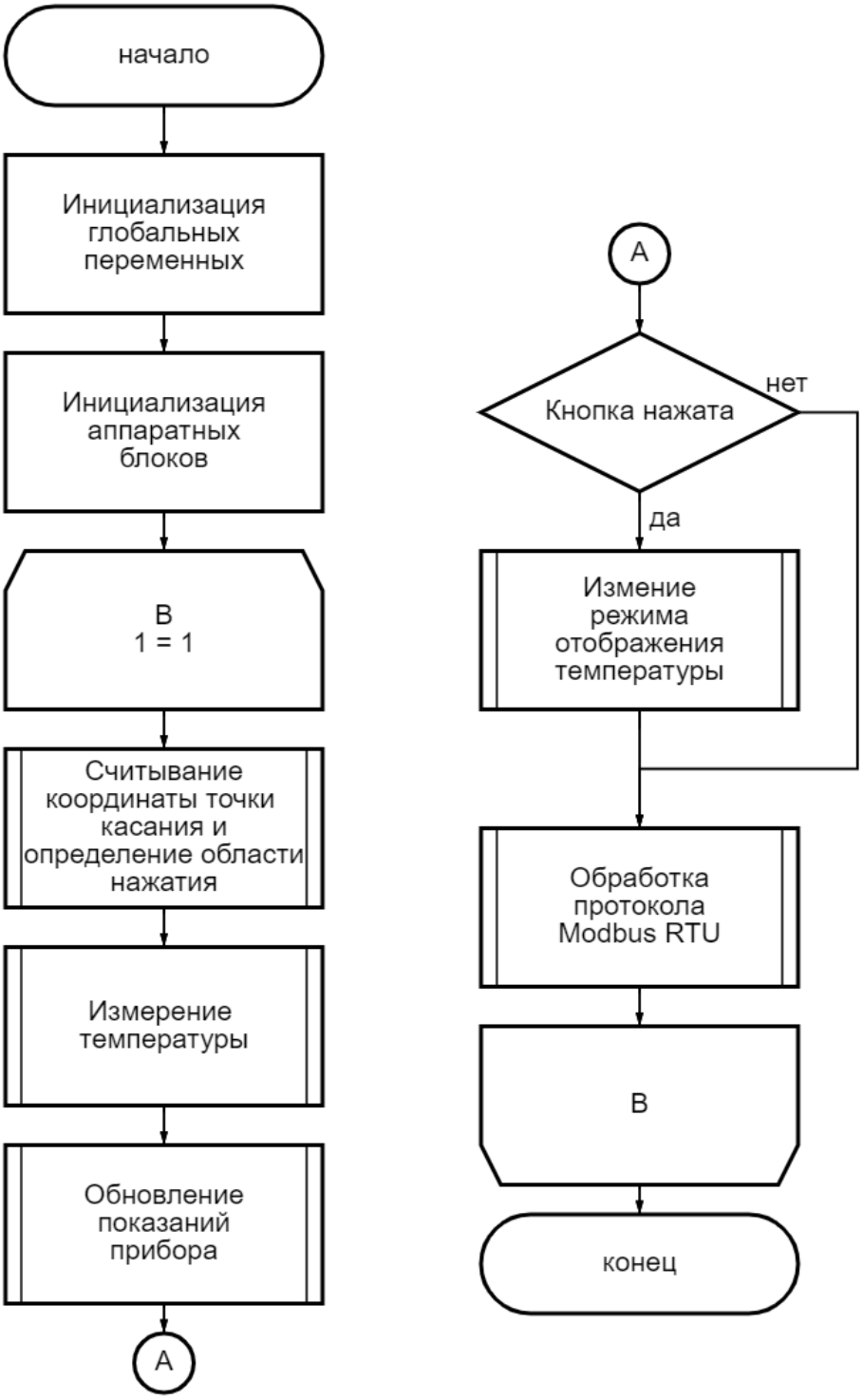

Рисунок 15 – Обобщенный алгоритм работы программы

При включении питания микроконтроллера инициализируются глобальные переменные и необходимые аппаратные блоки.

После инициализации аппаратных блоков производится отрисовка главного экрана графического интерфейса прибора.

Затем начинается бесконечный цикл, в котором происходит считывание сенсорной панели управления с последующей обработкой факта нажатия на виртуальную кнопку, измерение температуры, обновление показаний прибора и обработка протокола Modbus RTU. По умолчанию температура выводится в градусах Цельсия.

При нажатии на кнопку C на индикаторе будет отображаться температура в градусах Цельсия. При нажатии на кнопку K на индикаторе будет отображаться температура в Кельвинах. При нажатии на кнопку F будет отображаться температура в градусах Фаренгейта.

Измерение температуры происходит сразу в трех единицах измерения. Полученные значения температуры доступны при опросе устройства по протоколу Modbus RTU.

2 РАЗРАБОТКА АЛГОРИТМА ВЫЧИСЛЕНИЙ ИЗМЕРЯЕМОГО ПАРАМЕТРА

Для разработки алгоритма вычислений измеряемой температуры, необходимо определить номинальную статическую характеристику (НСХ) терморезистора.

НСХ терморезистора – функция, которая определяет зависимость сопротивления терморезистора от температуры.

В ГОСТ 6651-2009 [9, с. 7] приведена НСХ для платиновых термосопротивлений. НСХ для диапазона измерений от минус 200 до 0 °С описывается формулой (1):

$$R_t = R_0[1 + At + Bt^2 + C(t - 100°C)t^3], \quad (1)$$

где R_t – сопротивления терморезистора при температуре t, Ом;

R_0 – сопротивление терморезистора при температуре 0 °С, Ом;

t – температура, °С;

A – коэффициент, $°C^{-1}$;

B – коэффициент, $°C^{-2}$;

C – коэффициент, $°C^{-3}$.

Значение коэффициентов A, B, C зависят от типа платины, который в свою очередь характеризуется температурным коэффициентом сопротивления (ТКС) α [9, с. 6]. Коэффициент α вычисляется по формуле (2):

$$\alpha = \frac{R_{100} - R_0}{R_0 \cdot 100°C}, \quad (2)$$

где R_{100} – сопротивление при температуре 100 °С, Ом.

Для диапазона измерений от 0 до 850 °С НСХ определяется по формуле (3):

$$R_t = R_0[1 + At + Bt^2] \quad (3)$$

На рисунке 16 представлен график НСХ для платины с ТКС $3{,}85 \cdot 10^{-3}$ °C^{-1} и R_0 величиной 100 Ом.

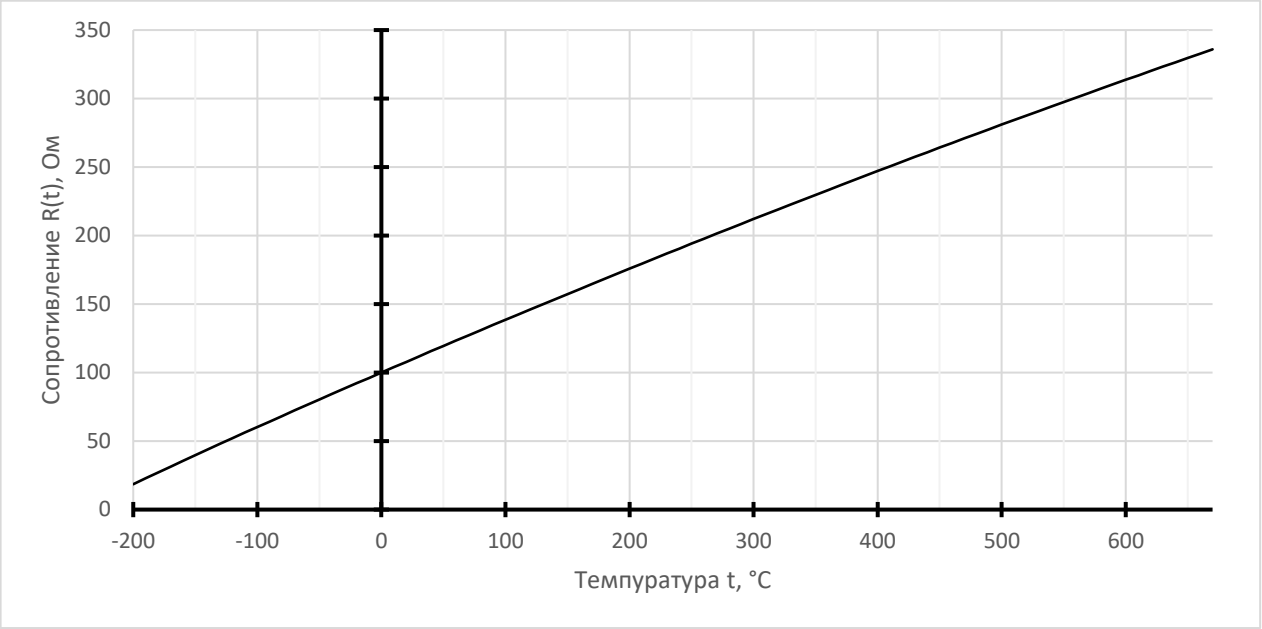

Рисунок 16 - Номинальная статическая характеристика платинового термосопротивления

Из графика видно, что характеристика не является абсолютно линейной. Тем не менее существуют аппаратные методы, позволяющие преобразовать подобную характеристику в линейную [10, с. 75-79]. При этом сигнал измерительного преобразователя нормируется так, чтобы минимальному измеряемому значению температуры $t_{мин.}$ соответствовало выходное напряжение $U_{вых.}$, равное нулю В, а максимальному значению температуры $t_{макс.}$ соответствовало выходное напряжения $U_{вых.}$, равное пяти В.

Таким образом преобразовать код АЦП в значение температуры можно по формуле (4):

$$t = k \cdot x + b, \tag{4}$$

где t – измеренное значение температуры, °C;

k, b – коэффициенты;

x – код, полученный с помощью АЦП.

Коэффициент k вычисляется по формуле (5):

$$k = \frac{t_{\text{макс.}} - t_{\text{мин.}}}{2^n - 1}, \tag{5}$$

где $t_{\text{макс.}}$ – верхний предел измерения, °C;

$t_{\text{мин.}}$ – нижний предел измерения, °C;

n – разрядность АЦП, бит;

Коэффициент b вычисляется по формуле (6):

$$b = t_{\text{мин.}} \tag{6}$$

Поскольку используемый микроконтроллер не имеет блока для вычислений с плавающей точкой, целесообразно производить преобразование кода АЦП в значение температуры без использования чисел с плавающей точкой.

В таком случае формула (4) преобразуется в формулу (7):

$$t = \frac{(k \cdot x + b) \cdot 2^m \cdot 10^r}{2^m}, \tag{7}$$

где m – масштабирующий коэффициент;

r – требуемая разрешающая способность.

В формуле (6) заранее вычисляется числитель, затем все коэффициенты округляются до целого. Деление заменяется на арифметический сдвиг вправо на m бит. Погрешность такого метода вычисляется по формуле (8):

$$\Delta_{\text{метода}} = \frac{1}{2^m} \tag{8}$$

При подстановке числовых значений в формулу 7, получим следующие формулы преобразования кода АЦП в температуру:

$$t_{\circ C} = \frac{512500 \cdot x - 131072000}{2^{16}}; \quad (9)$$

$$t_K = t_{\circ C} + 2731; \quad (10)$$

$$t_{\circ F} = \frac{922500 \cdot x - 214958080}{2^{16}}; \quad (11)$$

где $t_{\circ C}$ – значение температуры в градусах Цельсия;

t_K – значение температуры в Кельвинах;

$t_{\circ F}$ – значение температуры в градусах Фаренгейта.

Стоит заметить, что значения, полученные с помощью формул (9 – 11) увеличены в 10 раз из-за того, что хранение десятичного разделителя в памяти микроконтроллера не подразумевается.

Также из формулы (5) следует, что при проектировании реального устройства для обеспечения требуемой точности необходимо использовать внешнюю микросхему АЦП с разрядностью 14 бит или более.

3 ОПИСАНИЕ ФРАГМЕНТОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МИКРОКОНТРОЛЛЕРА

В данной разделе описываются фрагменты программы для микроконтроллера. Полный текст программы представлен в приложении А.

Разработка программного обеспечения производится с помощью специальной среды разработки Microchip Studio [11]. Начальный экран среды разработки представлен на рисунке 17.

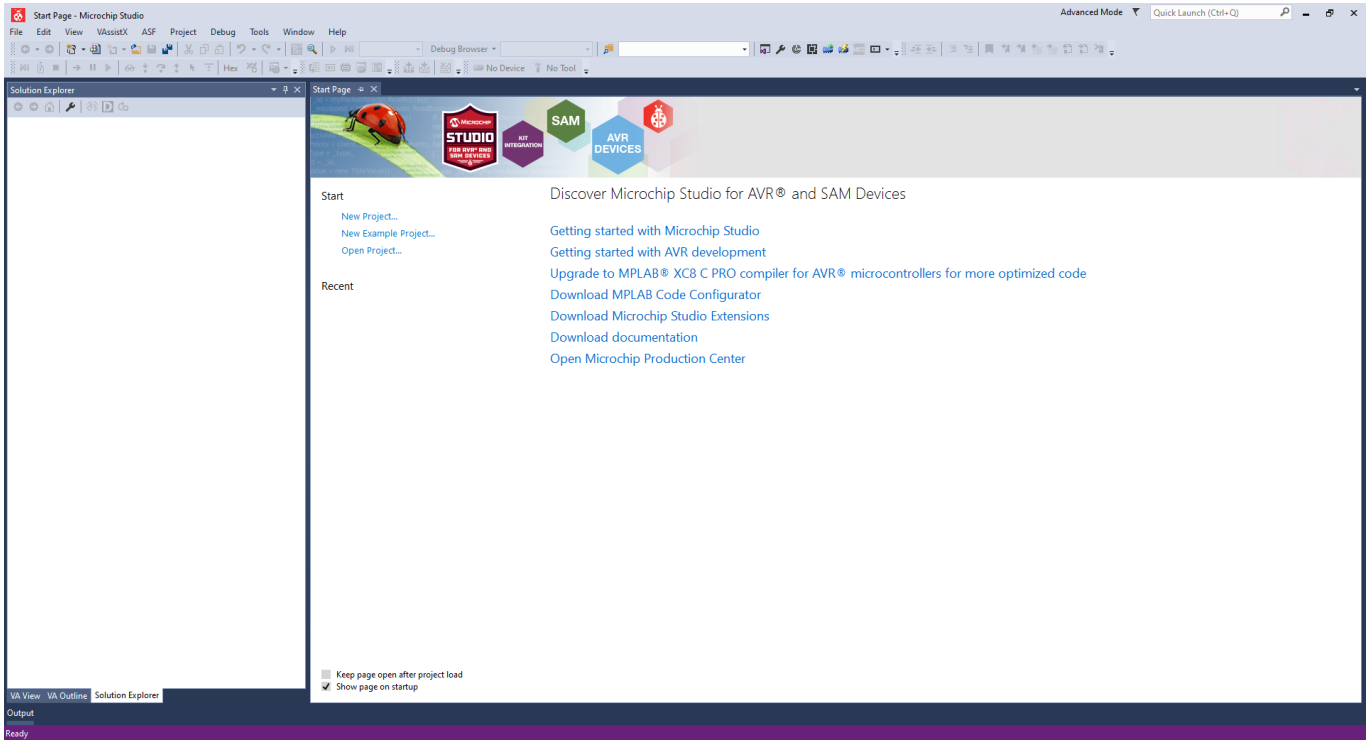

Рисунок 17 – Начальный экран среды разработки Microchip Studio

Для создания нового проекта необходимо навести курсор на ссылку «New Project» в области Start и нажать на левую кнопку мыши. В результате появится окно создания нового проекта, в котором необходимо выбрать тип проекта GCC C Executable Project, выбрать имя и расположение файлов проекта. Окно создания проекта представлено на рисунке 18 .

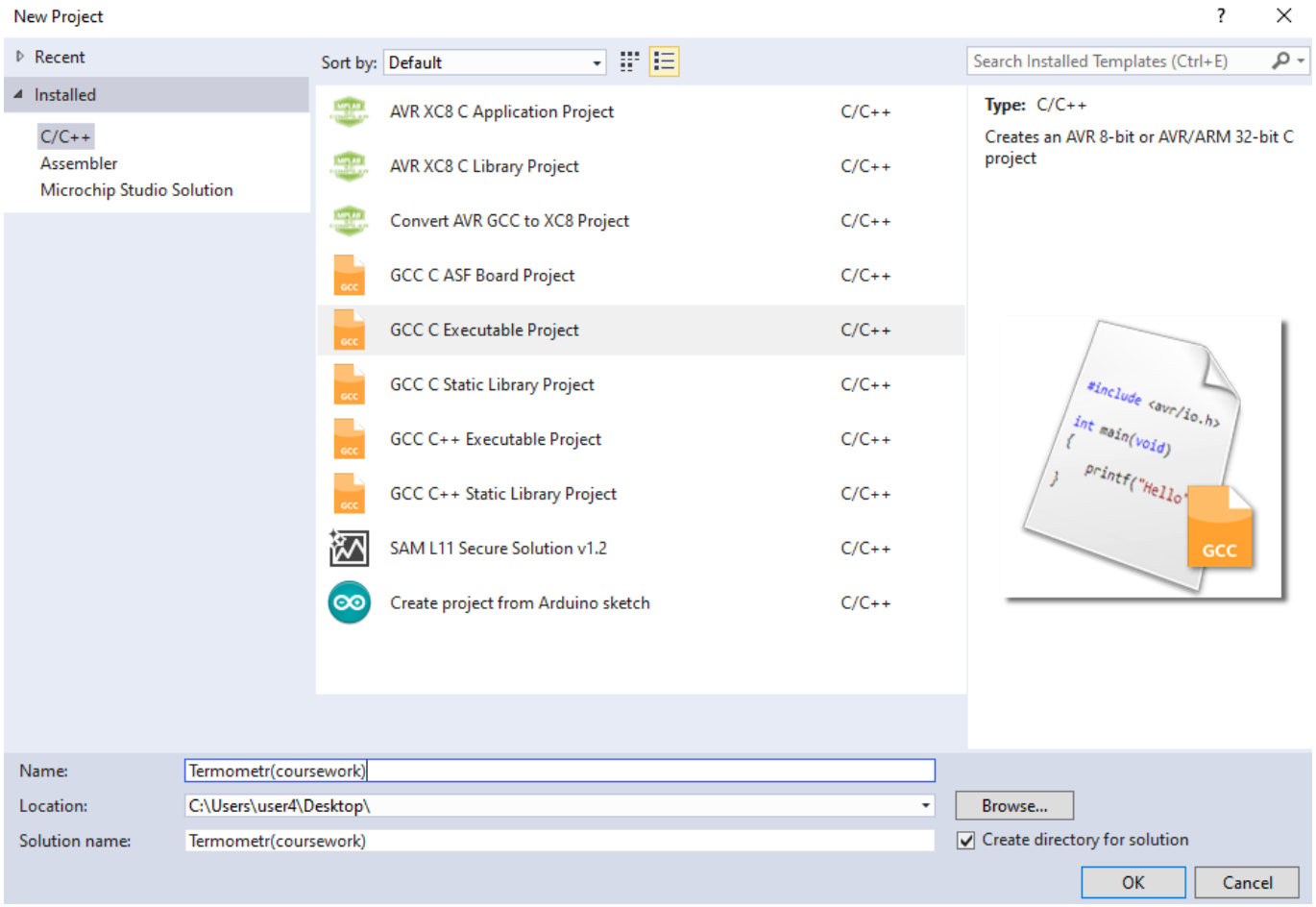

Рисунок 18 – Окно создания нового проекта

После нажатия на кнопку OK появляется окно выбора микроконтроллера, в котором необходимо выбрать микроконтроллер ATmega 32. Поиск микроконтроллера производится с помощью поисковой строки Search for device. Окно выбора микроконтроллера представлено на рисунке 19. После нажатия клавиши OK создание проекта завершится.

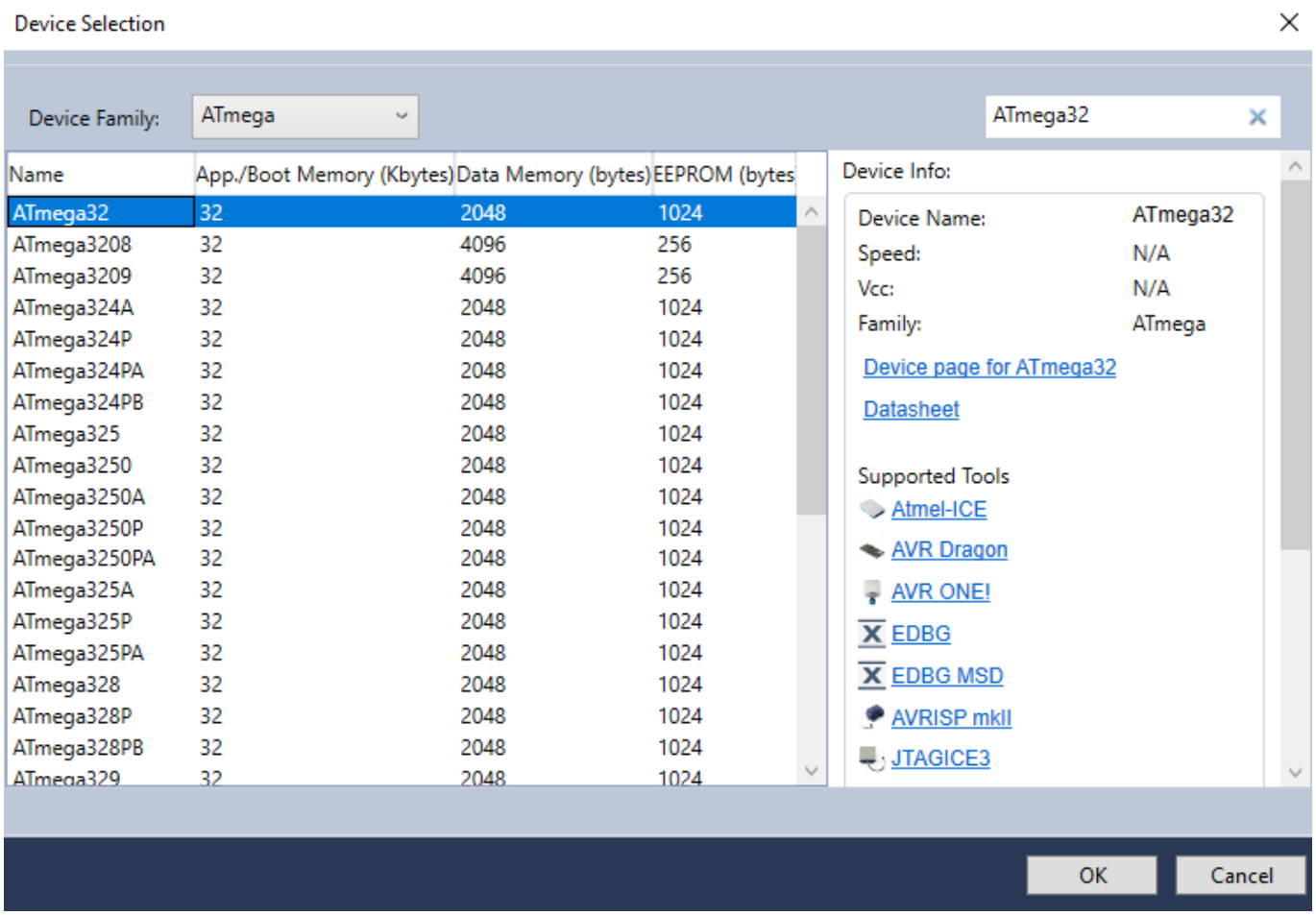

Рисунок 19 – Окно выбора микроконтроллера

После создания проекта появится рабочее пространство проекта. В рабочем пространстве есть следующие области:

– область кода;
– область навигации (Solution Explorer);
– область вывода (Output).

Область кода представляет собой пространство текстового редактора, в котором производится написание программного обеспечения.

В области навигации отображены все файлы проекта

В области вывода располагаются сообщения компилятора, сигнализирующие о статусе сборки проекта.

На рисунке 20 представлено рабочее пространство среды разработки.

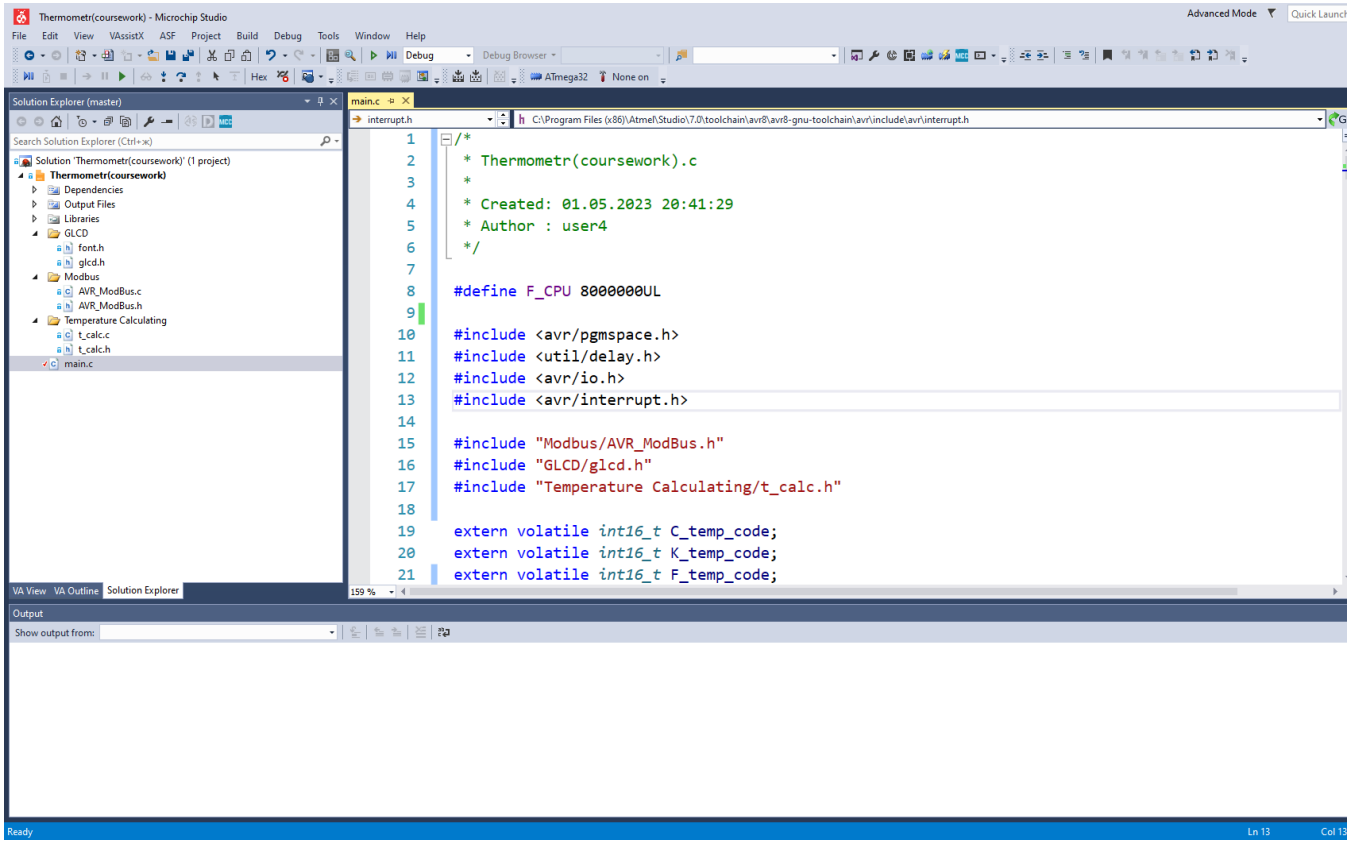

Рисунок 20 – Рабочее пространство среды разработки

Для сборки файлов проекта необходимо нажать Build -> Build solution. После этого в файлах в папке Debug появится файл с расширением .hex, который необходим для загрузки программного обеспечения в память микроконтроллера.

3.1 Определение и подключение библиотек

На рисунке 21 приведен листинг подключения библиотек.

```
#define F_CPU 8000000UL

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

#include "Modbus/AVR_ModBus.h"
#include "GLCD/glcd.h"
#include "Temperature Calculating/t_calc.h"
```

Рисунок 21 - Листинг подключения библиотек

Для всех библиотек необходимо предопределить константу F_CPU, которая хранит значение тактовой частоты микроконтроллера. Переопределение константы производится с помощью директивы препроцессора #define.

Библиотеки, имена которых заключены в фигурные скобки поставляются вместе со средой программирования Microchip Studio.

Библиотеки, имена которых заключены в кавычки являются внешними файлами, которые должны храниться в проекте.

В качестве основы для обработки протокола Modbus RTU используется библиотека AVR_ModBus [12]. Данная библиотека использует следующие аппаратные блоки:

– Модуль USART;
– Таймер-счётчик 0.

Для корректной работы библиотеки необходимо модифицировать заголовочный файл AVR_Modbus.h.

На рисунке 22 приведен листинг модификации заголовочного файла AVR_Modbus.h.

```
//ModBus
//ID оборудования
#define DEFAULT_SLAVE_ID 0x0A
//максимальный размер буфера принимаемых(Rx)/передаваемых(Tx) по UART данных, байт.
#define MAX_LENGHT_REC_BUF 25
#define MAX_LENGHT_TR_BUF 25
//количество регистров
#define QUANTITY_REG_0X 0 //количество дискретных выходов (DO) / r0x01, w0x05, w0x0F(15)
#define QUANTITY_REG_1X 0 //количество дискретных входов (DI) / r0x02
#define QUANTITY_REG_3X 6 //количество аналоговых входов (AI) / r0x04
#define QUANTITY_REG_4X 2 //количество аналоговых выходов (AO) r0x03, w0x06, w0x10(16)
```

Рисунок 22 - Листинг модификации заголовочного файла AVR_Modbus.h

Для работы с графическим индикатором используется библиотека glcd [13]. В файлах данной библиотеки определены функции отрисовки текста и примитивов на графическом индикаторе.

В файлах библиотеки t_calc определены функции подсчета температуры и подготовки результата измерения к выводу на индикатор и передаче по протоколу Modbus RTU.

3.2 Инициализация переменных

На рисунке 23 приведен листинг инициализации переменных

```
extern volatile int16_t C_temp_code;
extern volatile int16_t K_temp_code;
extern volatile int16_t F_temp_code;

#define base 20 //отступ от края дисплея

volatile uint8_t g_key = 0;             //номер нажатой клавиши
volatile uint8_t g_key_status = 0;      //Если бит 7 равен 1, значит клавиша нажата.
                                        //Если бит 6 равен 1, значит клавиша не отпущена

volatile uint8_t mode = 0; //Режим отображения температуры
/*
1. mode == 0 - температура в градусах Цельсия
2. mode == 1 - температура в Кельвинах
3. mode == 2 - температура в градусах Фаренгейта
*/

volatile uint8_t upd_flag = 0; //флаг, сигнализирующий об обновлении данных

volatile uint8_t TCI_counts = 0; //количество прерываний таймера 2

RegNum4x[0] = BAUD_RATE;        //скорость передачи по умолчанию
RegNum4x[1] = DEFAULT_SLAVE_ID; //адрес устройства по умолчанию
```

Рисунок 23 – Листинг объявления глобальных переменных

Переменные C_temp_code, K_temp_code и F_temp_code хранят значения, полученные с помощью формул (9-11). Эти переменные определены в файле t_calc.c.

Переменные g_key и g_key_status нужны для проверки факта нажатия на сенсорную кнопку.

Переменная TCI_counts нужна для формирования временных интервалов, соответствующих частоте опроса сенсорной панели и датчика температуры.

3.3 Инициализация аппаратных блоков

Функции, описываемые в данном подразделе, вызываются в начале функции main и соответствуют блоку «Инициализация аппаратных блоков» обобщенного алгоритма работы (рисунок 15).

3.3.1 Инициализация портов ввода-вывода

Каждая линия портов ввода-вывода может быть индивидуально настроена на ввод или вывод двоичной информации. Конфигурация направления информации портов ввода-вывода производится с помощью регистров DDRx, где x – буквенное обозначение порта ввода-вывода. Если бит номер n равен единице, значит линия номер n соответствующего порта ввода-вывода настроена на вывод. Иначе эта линия настроена на ввод.

Если линия порта ввода-вывода настроена на вывод, ее состояние можно изменить с помощью регистра PORTx.

Если линия порта ввода-вывода настроена на ввод, ее состояние отражается в регистре PINx.

На рисунке 24 приведен листинг функции инициализации портов ввода-вывода.

```
void IO_init(void)
{
        DDRA |=0b00001100;      //инициализация портов
        PORTA |=0b00000100;     //для сенсерной панели управления

        DDRB |=0b11100011;
        PORTB |=0b00000000;

        DDRD |=0b11000000;
        PORTD |=0b10000000;

        DDRC |=0b11111111;
}
```

Рисунок 24 – Листинг функции инициализации портов ввода-вывода

3.3.2 Инициализация таймера-счётчика 2

На рисунке 25 представлена блок схема таймера-счётчика.

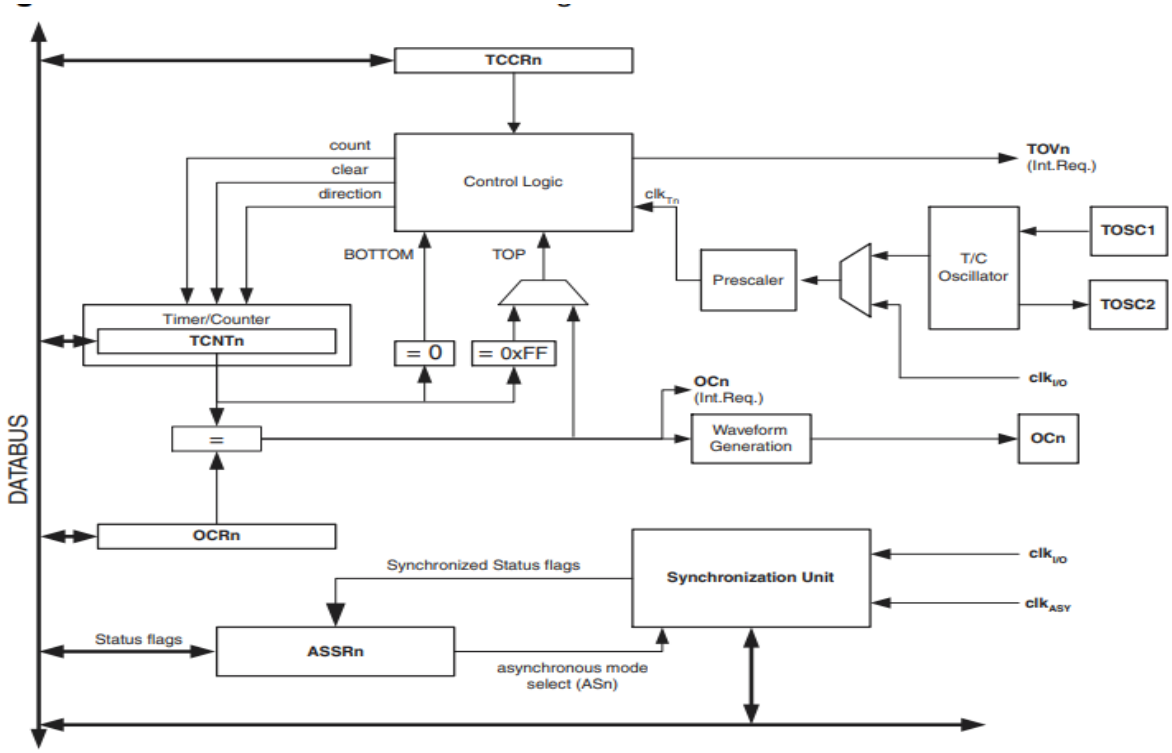

Рисунок 25 – Структурная схема таймера счетчика

Таймер-счётчик считает импульсы, поступающие с предделителя. Количество посчитанных импульсов хранится в регистре TCNTx, где x – номер таймера счетчика. Регистр OCRx хранит значение сравнения. В режиме «сброс при совпадении» (CTC), если значения TCNTx и OCRx совпадают, то следующий импульс установит бит OCFx в регистре TIFR в единицу и, если в регистре TIMSK установлен бит OCIEx, то произойдет прерывание.

Для настройки таймера-счётчика 2 необходимо изменить значения регистров OCR2, TCCR2, TIMSK. На рисунке 26 представлены регистры таймера-счётчика 2 [4, с. 122-128].

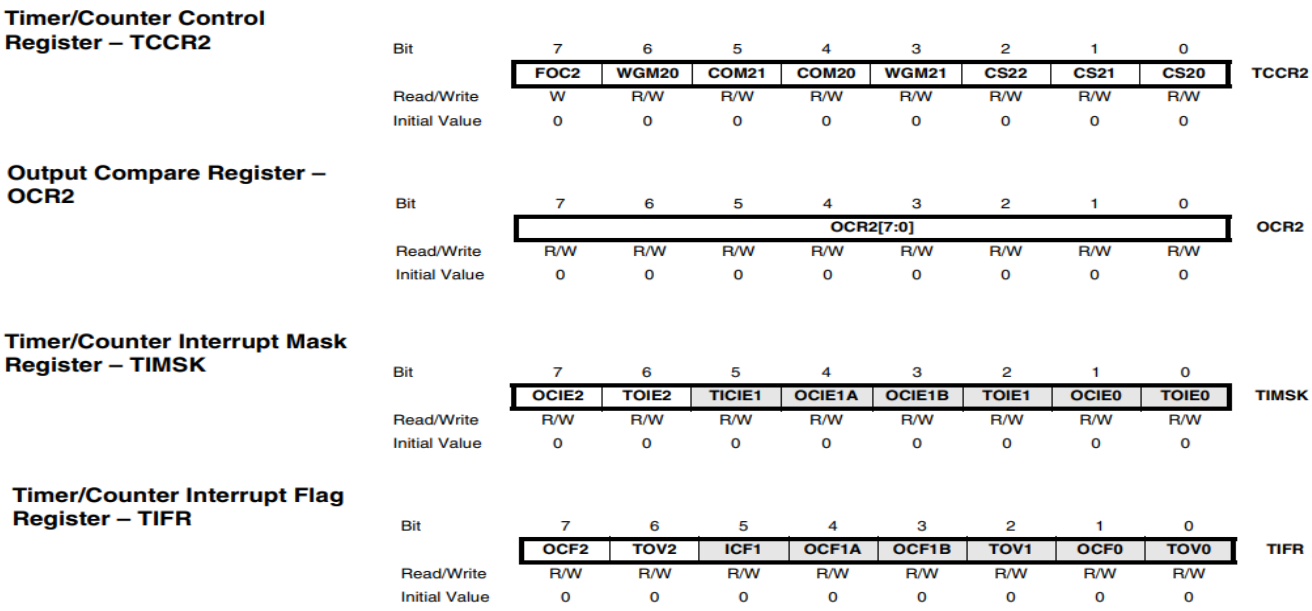

Рисунок 26 Регистры управления таймера-счётчика 2

Выбор источника тактового сигнала определятся значениями разрядов CS21, CS22 в регистре TCCR2. В таблице 4 приведены все возможные комбинации значений в этих разрядах.

Режим работы таймера-счётчика определяется значениями разрядов WGM20, WGM21 в регистре TCCR2. В таблице 5 приведены все возможные комбинации значений в этих разрядах.

Таблица 4 – Значения разрядов CS22, CS21, CS20

CS22	CS21	CS20	Источник тактового сигнала
0	0	0	f_{tc} = 0 (таймер остановлен)
0	0	1	$f_{tc} = f_{clk}$
0	1	0	$f_{tc} = f_{clk}/8$
0	1	1	$f_{tc} = f_{clk}/32$
1	0	0	$f_{tc} = f_{clk}/64$
1	0	1	$f_{tc} = f_{clk}/128$
1	1	0	$f_{tc} = f_{clk}/256$
1	1	1	$f_{tc} = f_{clk}/1024$

Таблица 5 – Значения разрядов WGM20, WGM21 регистра TCCR2

WGM21	WGM20	Режим работы таймера
0	0	Нормальный
0	1	ШИМ с корректной фазой
1	0	Сброс при совпадении
1	1	Быстрый ШИМ

Расчет интервала срабатывания прерывания таймера определяется по формуле (12):

$$T = \frac{d \cdot (OCR + 1)}{f_{osc}}, \tag{12}$$

где T – интервал срабатывания таймера, с;

d – величина предделителя;

OCR – значение регистра сравнения;

f_{osc} – тактовая частота микроконтроллера, Гц.

Подобрав величину предделителя так, чтобы при требуемом периоде срабатывания таймера, значение OCR было целым, получим искомые значения предделителя и OCR. На рисунке 27 представлен листинг функции инициализации таймера-счётчика 2.

```
void timer_init(void)
{
        //настройка на срабатывание T/C2 с интервалом 8 мс
        TCNT2=0b00000000;       //очистка T/C2
        OCR2=249;               //запись константы 249 в регистр сравнения T/C2
        TCCR2=0b00001110;       //режим "сброс при совпадении" и делитель на 256
        TIMSK|=0b10000000;      //разрешение прерывания по совпадению от T/C2
        TIFR &=0b11000000;      //очистка флагов прерываний T/C2
}
```

Рисунок 27 – Листинг функции инициализации таймера-счётчика 2

3.3.3 Инициализация внутреннего АЦП

На рисунке 28 представлена структурная схема внутреннего АЦП [4, с.200].

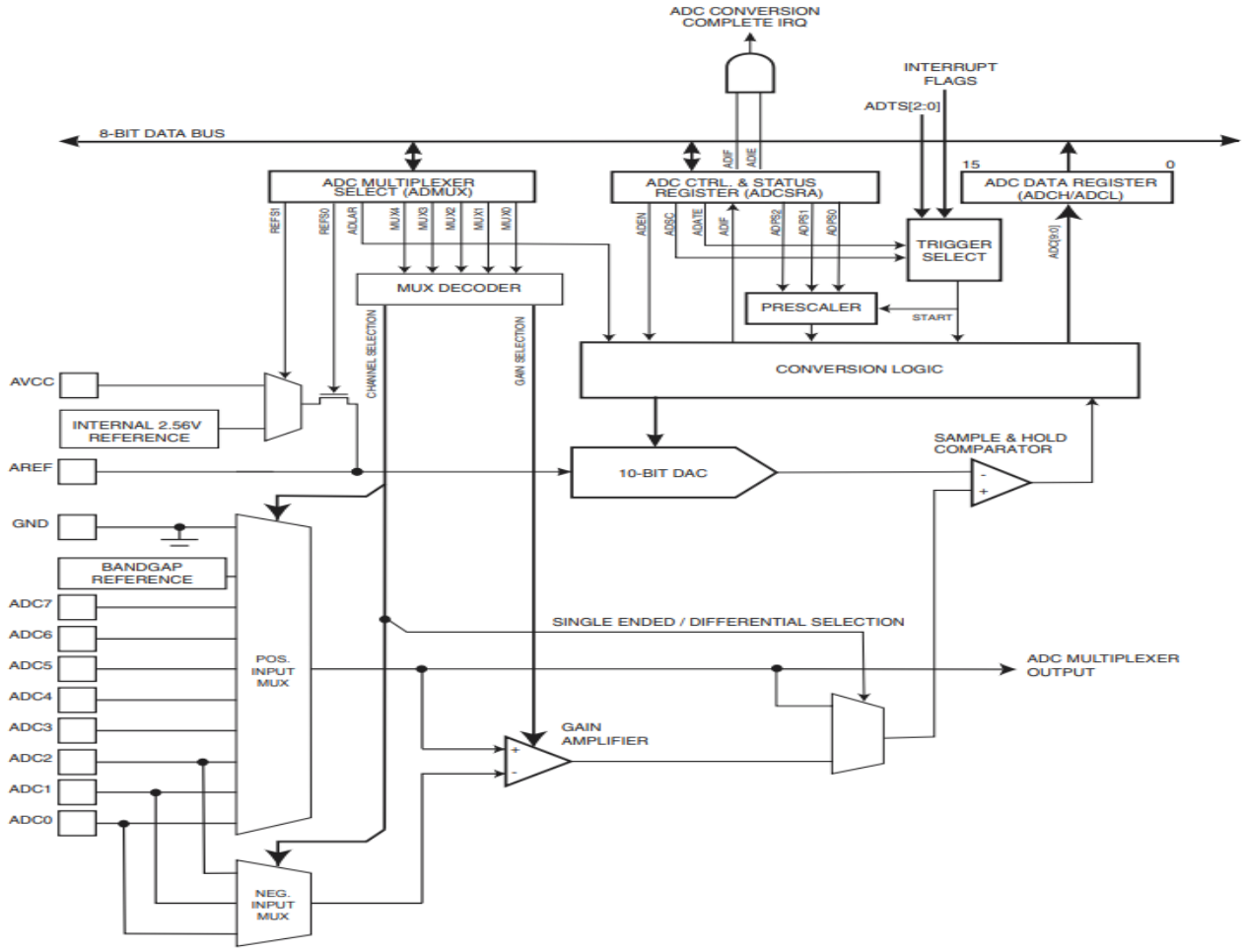

Рисунок 28 – Структурная схема внутреннего АЦП

На рисунке 29 представлены регистры управления внутреннего АЦП.

ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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	

ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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 29 – Регистры управления внутреннего АЦП

Разряды MUX0 – MUX4 регистра ADMUX настраиваются для выбора канала АЦП, при чем запись логических единиц в разряды MUX3, MUX4 переводят АЦП в режим дифференциального измерения напряжения.

Разряд ADEN в регистре ADCSRA включает АЦП. Разряд ADSC начинает аналогово-цифровое преобразование. Разряды ADPS0, ADPS1, ADPS2 изменяют величину делителя частоты, что изменяет на частоту дискретизации АЦП.

На рисунке 30 представлен листинг функции инициализации внутреннего АЦП.

```
void ADC_init(void)
{
        ADMUX = 0b00000000;
        ADCSRA = 0b10000111; //ADEN | предделитель АЦП 128
};
```

Рисунок 30 – Листинг функции инициализации внутреннего АЦП

3.3.4 Инициализация графического индикатора и отрисовка основного интерфейса прибора

На рисунке 31 представлена функция, реализующая рекомендуемую последовательность инициализации

```
void glcd_init(void)
{
        _delay_ms(100);
        glcd_off();             //выключаем дисплей
        _delay_ms(100);
        glcd_on();              //включаем после задержки
        glcd_clear();           //очистка дисплея
}
```

Рисунок 31 – Листинг функции инициализации индикатора

Интерфейс включает в себя следующие элементы:

– области виртуальных кнопок и их неизменные надписи;

– внешний контур;

– контур текстового поля, в котором отображаются показания прибора;

– надпись, отражающая показания прибора.

Для отрисовки интерфейса прибора нужна функция отрисовки прямоугольника со скругленными углами, которая не поставляется вместе с библиотекой для графического индикатора. На рисунке 32 представлена функции отрисовки прямоугольника со скругленными углами.

```
void draw_round_rectangle(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t radius)
{
        int16_t tSwitch = 3 - 2 * radius;
        uint8_t width, height, x, y;
        width = x2-x1;
        height = y2-y1;
        x = 0;
        y = radius;

        h_line(x1 + radius, y1, width - 2 * radius, 0, 1);
        h_line(x1 + radius, y2, width - 2 * radius, 0, 1);
        v_line(x1, y1 + radius, height - 2 * radius, 0, 1);
        v_line(x2, y1 + radius, height - 2 * radius, 0, 1);
         while (x <= y)
         {
                 point_at(x1 + radius - x, y1 + radius - y, 1);
                 point_at (x1 + radius - y, y1 + radius - x, 1);

                 point_at(x1 + width - radius + x, y1 + radius - y, 1);
                 point_at (x1 + width - radius + y, y1 + radius - x, 1);

                 point_at(x1 + radius - x, y1 + height - radius + y, 1);
                 point_at (x1 + radius - y, y1 + height - radius + x, 1);

                 point_at(x1 + width - radius + x, y1 + height - radius + y, 1);
                 point_at (x1 + width - radius + y, y1 + height - radius + x, 1);

                if (tSwitch < 0)
                {
                        tSwitch += 4 * x + 6;
                }
                else
                {
                        tSwitch += 4 * (x-y) + 10;
                        --y;
                }
                ++x;
         }
}
```

Рисунок 32 - Листинг функции отрисовки прямоугольника со скругленными углами

На рисунке 33 представлена функция отрисовки интерфейса прибора.

```
void draw_main_screen()
{
        draw_round_rectangle(0,0,126,62, 8);    //отрисовка внешнего контура
        // вывод границ виртуальных "кнопок"
        draw_round_rectangle(5,35,40,59, 8);
        draw_round_rectangle(86,35,121,59,8);
        draw_round_rectangle(45,35,81,59,8);
        // вывод текстовой заставки
        draw_round_rectangle(15,5,113,17, 6);
        glcd_puts("T=",base,1,0,1,1);
        glcd_puts("°C", base + 8 *9, 1, 0, 1, 1);
        // Размещаем буквы в центр виртуальных "кнопок"
        glcd_puts("C",16,5,0,2,1);
        glcd_puts("K",57,5,0,2,1);
        glcd_puts("F",97,5,0,2,1);
}
```

Рисунок 33 – Листинг функции отрисовки интерфейса прибора

На рисунке 34 изображение графического интерфейса прибора при включении.

Рисунок 34 – Изображение графического интерфейса прибора при включении

3.4 Обработчик прерываний таймера-счетчика 2

В момент, когда таймер-счётчик вызывает прерывание, ядро микроконтроллера запоминает контекст выполняемой функции и переходит к выполнению специальной функции, которая называется обработчиком прерывания.

На рисунке 35 представлена функция обработчика прерывания таймера-счётчика 2.

```
ISR(TIMER2_COMP_vect)
{
        ++TCI_counts;

        if(TCI_counts == 10)
        {
                TCI_counts = 0;


                upd_flag = 1;
        }
}
```

Рисунок 35 – Листинг обработчика прерывания таймера-счётчика 2

После выполнения обработчика прерывания, контекст восстанавливается и ядро микроконтроллера продолжает выполнение основной программы с того места, в котором основная программа прервалась.

Переменная upd_flag сигнализирует основной программе о том, что был отсчитан интервал, равный восьми миллисекундам. Затем в бесконечном цикле программы переменная upd_flag сравнивается с единицей. При равенстве выполняется вычисление координаты точки касания, определения области нажатия, измерение температуры и отображение показаний на индикаторе. Затем переменная upd_flag становится равной нулю.

На рисунке 36 представлена часть бесконечного цикла, которая описывалась выше.

```
while (1)
{
            if(upd_flag)
            {
                    scan_key();
                    get_temp();
                    display_result();
                    upd_flag = 0;
            }
```

Рисунок 36 – Листинг обработки переменной-флага upd_flag

3.5 Функция определения точки касания и области нажатия на сенсорной панели управления

В соответствии с алгоритмом, представленном на рисунке 14, программа для микроконтроллера каждые 8 миллисекунд производит считывание координаты точки касания.

После определения координаты точки касания программа определяет коллизию между полученной точкой и областью виртуальной кнопки. О том, что точка является частью области виртуальной кнопки, сигнализируют переменные g_key_status и g_key.

На рисунке 37 представлена функция обработки виртуальных кнопок.

```
void scan_key(void)
{
        PORTA=0b00000100;
        _delay_ms(2);                   // Задержка !!! (паразитная емкость должна зарядиться)
        ADMUX=0b00000000;               // Выбираем нулевой канал АЦП
        ADCSRA|=(1<<ADSC);              // Запуск АЦП
        while ((ADCSRA&(1<<ADSC))!=0);  // Ожидаем конца преобразования АЦП
        volatile uint16_t x_coordinate=ADC;

        PORTA=0b00001000;
        _delay_ms(2);
        ADMUX=0b00000001;               // Смена канала
        ADCSRA|=(1<<ADSC);              // Запуск АЦП
        while ((ADCSRA&(1<<ADSC))!=0);  // Ожидаем конца преобразования АЦП
        volatile uint16_t y_coordinate=ADC;
        if (((y_coordinate>190)&&(y_coordinate<370))&&((x_coordinate>80)&&(x_coordinate<260))&&(g_key_status==0))
        { //Регистрация нажатия кнопки "C"
                g_key_status=0b11000000;
                g_key=0;
        }
        if (((y_coordinate>190)&&(y_coordinate<370))&&((x_coordinate>330)&&(x_coordinate<490))&&(g_key_status==0))
        { //Регистрация нажатия кнопки "K"
                g_key_status=0b11000000;
                g_key=1;
        }
        if (((y_coordinate>190)&&(y_coordinate<370))&&((x_coordinate>565)&&(x_coordinate<730))&&(g_key_status==0))
        { //Регистрация нажатия кнопки "F"
                g_key_status=0b11000000;
                g_key=2;
        }
        if (((y_coordinate<20)&&(x_coordinate<20))&&(g_key_status&0b01000000)!=0)
        { //если нет замыкания между слоями панели после нажатия
                g_key_status=g_key_status&0b10000000; //значит виртуальная кнопка отпущена
        }

}
```

Рисунок 37 - Листинг функции обработки нажатия виртуальных кнопок

Определение условий, которые ограничивающих области виртуальных кнопок, производится опытным путем. Для этого необходимо видоизменить функцию, представленную на рисунке 37 так, чтобы при отрисованных областях кнопок вместо температуры выводились координаты точки касания.

Таким образом для определения условий, ограничивающих области виртуальных кнопок, необходимо произвести четыре касания на границе областей и записать их координаты, а именно:

– в самой верхней границе области кнопки;
– в самой нижней границе области кнопки;
– в самой левой границе области кнопки;
– в самой правой границе области кнопки.

Пример такой функции для определения координат точки касания представлен на рисунке 38.

```
//      функция  отображения координат точки касания
void disp_x_y (void){
        PORTA=0b00000100;
        _delay_ms(2);                   // Задержка !!! (паразитная емкость должна зарядиться)
        ADMUX=0b00000000;               // Выбор нулевого канала АЦП
        ADCSRA|=(1<<ADSC);              // Запуск АЦП
        while ((ADCSRA&(1<<ADSC))!=0);  // Ожидаем конца преобразования АЦП
// Отображаем координату на графическом дисплее
        temp_code_ADC=ADC;
        glcd_putchar((temp_code_ADC%10+48),60,1,0,1); //младшая цифра
        temp_code_ADC/=10;
        glcd_putchar((temp_code_ADC%10+48),51,1,0,1);
        temp_code_ADC/=10;
        glcd_putchar((temp_code_ADC%10+48),42,1,0,1);
        temp_code_ADC/=10;
        glcd_putchar((temp_code_ADC%10+48),33,1,0,1); //старшая цифра
//////
        PORTA=0b00001000;
        _delay_ms(2);
        ADMUX=0b00000001;
        ADCSRA|=(1<<ADSC);
        while ((ADCSRA&(1<<ADSC))!=0);
// Отображаем координату на графическом дисплее
        temp_code_ADC=ADC;
        glcd_putchar((temp_code_ADC%10+48),60,2,0,1); //младшая цифра
        temp_code_ADC/=10;
        glcd_putchar((temp_code_ADC%10+48),51,2,0,1);
        temp_code_ADC/=10;
        glcd_putchar((temp_code_ADC%10+48),42,2,0,1);
        temp_code_ADC/=10;
        glcd_putchar((temp_code_ADC%10+48),33,2,0,1); //старшая цифра
}
```

Рисунок 38 – Пример функции, отображающей координаты точки касания на индикаторе

Далее в главной цикле программы обрабатываются переменные g_key и g_key_status или, что то же самое, обрабатывается факт нажатия виртуальной кнопки. Обработка заключается в смене переменной mode, состояние которой отражает текущий режим отображения температуры.

На рисунке 39 представлен фрагмент главного цикла программы, в котором обрабатывается факт нажатия виртуальной кнопки.

```
if ((g_key_status &0b10000000) != 0)
{//Если клавиша нажата
        g_key_status &= 0b01000000; //сбросить бит, отвечающий за факт нажатия
        switch (g_key)
        {//поменять режим в зависимости от нажатой кнопки
                case 0:
                {
                        mode = 0;
                        glcd_puts("°C",base+8*9,1,0,1,1);
                        break;
                }
                case 1:
                {
                        mode = 1;
                        glcd_puts(" K", base+8*9,1,0,1,1);
                        break;
                }
                case 2:
                {
                        mode = 2;
                        glcd_puts("°F",base+8*9,1,0,1,1);
                        break;
                }
        }
}
```

Рисунок 39 – Листинг фрагмента программы, обрабатывающего факт нажатия виртуальной кнопки

3.6 Функция измерения температуры

В соответствии с формулами (9-11) производится вычисление температуры в трех единицах измерения. На рисунке 40 представлена функция измерения температуры, которая определена в файле t_calc.c.

```
//инициализация структур данных для хранения температуры
volatile union temperature_t temp_C = {0};
volatile union temperature_t temp_K = {0};
volatile union temperature_t temp_F = {0};

//переменные для вывода на графический индикатор
volatile int16_t C_temp_code = 0;
volatile int16_t K_temp_code = 0;
volatile int16_t F_temp_code = 0;

void get_temp(void)
{
        ADMUX=0b00000101;
        ADCSRA|=(1<<ADSC);              // запуск АЦП;
        while ((ADCSRA&(1<<ADSC))!=0); //ожидание конца преобразования АЦП
        C_temp_code = ((int32_t)ADCfactorK_C * ADC + (int32_t)ADCfactorB_C) >> 16;
        temp_C.f_cell = (float)C_temp_code * 0.1f;
        for (uint8_t i = 0; i < 2; ++i)
        {//байты хранятся в памяти мк в порядке "от младшего к старшему"
                RegNum3x[i] = temp_C.buff[1 - i];
        }


        K_temp_code = C_temp_code + 2731; //переменные X_temp_value хранят значение в 10 раз больше фактического
        temp_K.f_cell = (float)K_temp_code * 0.1f;
        for (uint8_t i = 0; i < 2; ++i)
        {
                RegNum3x[i+2] = temp_K.buff[1 - i];
        }

        F_temp_code = ((int32_t)ADCfactorK_F * ADC + (int32_t)ADCfactorB_F) >> 16;
        temp_F.f_cell = (float)F_temp_code * 0.1f;

        for (uint8_t i = 0; i < 2; ++i)
        {
                RegNum3x[i + 4] = temp_F.buff[1 - i];
        }
}
```

Рисунок 40 – Листинг функции измерения температуры

Измеренное значение температуры хранится в двух группах переменных. Переменные с типом uint16_t хранят значения, полученные с помощью формул (9-11). Данная группа переменных используется для вывода показаний прибора на графический индикатор.

Переменные с типом union temperature_t являются объединением. Объединение представляет собой структуру данных, которая имеет несколько представлений. На рисунке 41 представлено определение переменных типа union temperature_t.

```
union temperature_t
{
        float f_cell;                   //Часть объединения, в которое нужно записать
                                        //вещественное значение температуры для передачи
                                        //по протоколу ModBus RTU
        uint16_t buff[2];               //Это поле нужно для передачи всего значения с помощью
                                        //обработчика Modbus RTU
};
```

Рисунок 41 – Листинг определения переменной типа union temperature_t

Объединение позволяет расположить число с плавающей точкой в адресном пространстве протокола Modbus RTU в форме массива из двух переменных типа uint16_t.

Измерение начинается с выбора пятого канала АЦП и записи логической единицы в поле ADSC. После получения кода АЦП производится вычисление температуры в соответствии с формулами (9-11). Как упоминалось ранее, деление заменяется на побитовый сдвиг вправо на 16 бит.

Далее в объединение temp_C через поле f_cell производится запись измеренного значения температуры в формате с плавающей точкой после деления на 10. Полученное значение через поля buff[0] и buff[1] записываются в область памяти, выделенную под регистры аналоговых входов. Данные регистры определены в библиотеке AVR_Modbus.

3.7 Функция отображения показаний прибора

В зависимости от режима отображения температуры (переменная mode) в переменную code_temp записывается соответствующее значение температуры из переменной X_temp_code, где X принимает значения C, K, F.

Далее производится проверка знака. Если число отрицательное, то выводится знак минус и переменная code_temp меняет знак. Иначе вместо знака минус выводится пробел.

Для проведения двочино-десятичного преобразования выделится массив из 4-х переменных типа uint8_t с именем nums[]. Преобразование производится путем циклической записи остатка от деления переменной code_temp на 10 с последующим делением переменной на 10. Хранение цифр числа производится в порядке от старшего разряда к младшему.

На рисунке 42 представлена первая часть функции отображения показаний прибора, описанная выше.

```
void display_result(void)
{
	int16_t code_temp = 0;

	switch(mode)
	{
		case 0:
		{
			code_temp = C_temp_code;
			break;
		}
		case 1:
		{
			code_temp = K_temp_code;
			break;
		}
		case 2:
		{
			code_temp = F_temp_code;
			break;
		}
	}

	if (code_temp < 0)
	{
		glcd_putchar('-', base+8*3, 1, 1, 1);
		code_temp = -code_temp;
	}
	else
	{
		glcd_putchar(' ', base+8*3, 1, 1, 1);
	}

	uint8_t nums[4] = {0,};
	for (int8_t i = 3; i > -1; --i) //преобразуем число в массив. в ячейке [0] лежит старший разряд
	{
		nums[i] = code_temp % 10 + 48;
		code_temp = code_temp / 10;
	}
```

Рисунок 42 – Листинг первой части функции отображения показаний прибора

После двоично-десятичного преобразования необходимо отфильтровать незначащие старшие разряды. Для этого вводится две переменные – cursor и count для отслеживания положения виртуального курсора и количества незначащих разрядов.

Далее до первого ненулевого символа или пока количество незначащих разрядов меньше двух проверяются символы в массиве nums[]. Если проверяемый символ является нулевым, то переменная count увеличивается на единицу. После завершения фильтрации в цикле выводятся все символы. Вместо незначащих разрядов выводятся пробелы в конце строки.

На рисунке 43 представлена вторая часть функции отображения показаний прибора, описанная выше.

```
        uint8_t cursor = 4;//текущая координата виртуального курсора
        uint8_t count = 0; //количество незначащих разрядов
        while(count < 2)
        {
                if (nums[count]== 48)
                {
                        ++count; //увеличиваем счетчик до первого ненулевого значения или до двух
                }
                else
                {
                        break;
                }
        }

        for (uint8_t i = count; i < 3; ++i)
        {
                glcd_putchar(nums[i], base+8*(cursor++), 1, 1, 1);
        }

        glcd_putchar('.', base + 8 * (cursor++), 1,0,1); //разделительная точка
        glcd_putchar(nums[3], base + 8*(cursor++), 1, 0, 1);
        for (uint8_t i = 0; i < count; ++i)
        {
                glcd_putchar(' ', base + 8*(cursor++), 1,0,1); //заполняем пробелами оставшиеся символы, если они есть
        }
}
```

Рисунок 43 – Листинг второй части функции отображения показаний прибора

3.8 Обработка протокола Modbus RTU. Модификация библиотеки AVR_Modbus

Протокол Modbus является самым распространенным промышленным протоколом, который фактически является стандартом для промышленной автоматики.

На физическом уровне данный протокол использует следующие интерфейсы:

– RS-232, RS-422, RS-485;

– сети TCP/IP.

На логическом уровне существует 3 вида реализации протокола:

– Modbus TCP;

– Modbus RTU;

– Modbus ASCII.

При использовании протокола Modbus TCP данные передаются в пакетах протокола TCP/IP. Проверка целостности пакетов вынесена на уровень TCP/IP.

При использовании протокола Modbus RTU двоичные данные передаются в пакетах. Начало и конец пакета определяется по интервалам времени.

При использовании протокола Modbus ASCII данные передаются с помощью пакетов в hex формате. Начало пакета определяется символом «:». Конец пакета определяет символ возврата каретки.

В данной курсовой работе выбран протокол Modbus RTU.

Структура пакета Modbus RTU имеет следующий вид:

- адрес устройства (один байт, допустимые значения от нуля до 247);
- PDU (protocol data unit, до 253 байт) – основная часть пакета;
- CRC16 (два байта) – контрольная сумма.

Основная часть пакет пакета состоит из кода функции и данных, которые необходимо считать или записать. Все данные устройства хранятся в специальных регистрах. В таблице 6 представлены виды регистров Modbus RTU и функции, которые используются для работы с регистрами.

Таблица 6 – Адресное пространство протокола Modbus

Вид регистра	**Размер регистра, бит**	**Диапазон адресов регистров**	**Функции**
Дискретные выходы (DO)	8	От 1 до 9999	01 – чтение группы регистров; 05 – запись одного регистра; 15 – запись группы регистров;
Дискретные входы (DI)	8	От 10001 до 19999	02 – чтение группы регистров.
Аналоговые входы (AI)	16	От 30001 до 39999	04 – чтение групп регистров.
Регистры хранения (AO)	16	От 40001 до 49999	03 – чтение группы регистров; 06 – запись одного регистра; 16 – запись группы регистров.

В таблице 6 адреса регистров указаны вместе со смещением. Смещение регистра зависит от его типа. Таким образом в пакете Modbus RTU указывается адрес регистра от 0000_{16} до $27EE_{16}$. При этом смещение регистра зависит от номера функции, который указан в соответствующем поле основной части пакета.

Для реализации требуемого функционала используется два регистра AO для хранения настроек адреса и скорости передачи данных и шесть регистров AI для хранения температуры в формате с плавающей точкой по стандарту IEEE 754.

В таблице 7 представлено адресное пространство устройства.

Таблица 7 – Описание регистров устройства

Адрес регистра	Тип регистра	Описание регистра	Тип данных
0000	AI	Температура, °C (старшие два байта).	Число с плавающей точкой IEEE 754.
0001	AI	Температура, °C (младшие два байта).	
0002	AI	Температура, K (старшие два байта).	
0003	AI	Температура, K (младшие два байта).	
0004	AI	Температура, °F (старшие два байта).	
0005	AI	Температура, °F (младшие два байта).	
0000	AO	Скорость передачи данных, бод.	Целое число без знака (два байта).
0001	AO	Адрес устройства.	

Пример пакета-запроса Modbus RTU представлен в таблице 8.

Таблица 8 – Пример пакета, в котором отражен запрос на чтение температуры

Адрес	Функция	Адрес регистра		Количество регистров	
0A	04	00	00	00	06

В библиотеке AVR_Modbus все регистры хранятся в массивах RegNum0x (DO), RegNum1x (DI), RegNum3x (AI), RegNum4x (AO).

По при включении питания массив RegNum3x инициализируется нулевыми значениями. Массив RegNum4x инициализируется значениями скорости передачи и адреса по умолчанию (9600 бод и 0x0A соответственно).

Устройство поддерживает следующий набор скоростей передачи данных:

– 9600;

– 14400;

– 19200;

– 38400;

– 57600.

Для модификации библиотеки необходимо ввести новые переменные в файле AVR_Modbus.c. Листинг инициализации этих переменных приведен на рисунке 44.

```
volatile unsigned char Slave_ID = DEFAULT_SLAVE_ID;

volatile unsigned char Change_Parametrs_Is_Recieved = 0;//Если равна 1, значит пришел пакет с новыми настройками
                                                       //скорости передачи устройства
volatile unsigned char Baud_Divider = BAUD_DIVIDER;     //текущий делитель для регистров UBRR
```

Рисунок 44 – Листинг инициализации дополнительных переменных

Переменная Slave_ID нужна для отслеживания текущего адреса устройства. Значение при инициализации – 0x0A. Переменные Change_Parametrs_Is_Recieved и Baud_Divider необходимы для обработки события смены скорости передачи данных. Изменять скорость передачи при получении соответствующего пакета данных нельзя, поскольку в таком случае ведущее устройство не сможет получить ответ.

Baud_Divider – переменная, значение которой должно быть записано в регистры UBRRH и UBRRL. Данные регистры задают скорость передачи модуля UART. Переменная Baud_Divider вычисляется по формуле 13 [4, с.141]:

$$bd = \frac{f_{osc}}{16 \cdot v} - 1, \tag{13}$$

где bd – значение переменной Baud_Divider;

f_{osc} – тактовая частота микроконтроллера;

v – скорость передачи данных.

Основная часть модификации библиотеки заключается в изменении функции Func06 и обработчика вектора прерывания USART_UDRE_vect. Листинг измененных функций приведен на рисунках 45, 46.

```
char Func06(void)
{
//проверка корректного адреса в запросе
if(!(reg_adress<=QUANTITY_REG_4X)) return ErrorMessage(0x02); //Адрес данных, указанный в запросе, недоступен
//проверка корректных данных в запросе

//формируем ответ, возвращая полученное сообщение
if((reg_adress == 0) && ((value == 9600) || (value == 14400)
|| (value == 19200) || (value == 38400) || (value == 57600)))//при необходимости поставить контроль значений
{

        RegNum4x[reg_adress] = value;
        Change_Parametrs_Is_Recieved = 1;
        cmTrBuf0[1] = cmRcBuf0[1];
        cmTrBuf0[2] = cmRcBuf0[2];
        cmTrBuf0[3] = cmRcBuf0[3];
        cmTrBuf0[4] = cmRcBuf0[4];
        cmTrBuf0[5] = cmRcBuf0[5];
        cmTrBuf0[6] = cmRcBuf0[6];
        cmTrBuf0[7] = cmRcBuf0[7];
        Baud_Divider = F_CPU/(RegNum4x[reg_adress] * 16.0) - 1;

        return 8;
} //end if()
else if ((reg_adress == 1) && (value < 248))
{
        RegNum4x[reg_adress] = value;
        cmTrBuf0[1] = cmRcBuf0[1];
        cmTrBuf0[2] = cmRcBuf0[2];
        cmTrBuf0[3] = cmRcBuf0[3];
        cmTrBuf0[4] = cmRcBuf0[4];
        cmTrBuf0[5] = cmRcBuf0[5];
        cmTrBuf0[6] = cmRcBuf0[6];
        cmTrBuf0[7] = cmRcBuf0[7];
        Slave_ID = RegNum4x[reg_adress];
        return 8;
}
else
{
        return ErrorMessage(0x03);//Значение, содержащееся в поле данных запроса, является недопустимой величиной
}
return 0;
} //end Func06(void)
```

Рисунок 45 – Листинг измененной функции Func06

Если принятое значение является допустимым, то это значение записывается в соответствующие регистры. Если была получена команда на изменение скорости передаче данных, то переменная-флаг Change_Parametrs_Is_Recieved становится равна единице. Значение переменной Baud_Divider пересчитывается по формуле 13. В данном случае вместо *v* используется принятое значение скорости передачи. После этого ведущему устройству отправляется копия полученного пакета.

В случае, когда полученные значения являются недопустимыми, в качестве ответа ведущему устройству отправляется сообщение об ошибке.

```
ISR(USART_UDRE_vect)
{
if (TrCount<cNumTrByte0)
{
    UDR=cmTrBuf0[TrCount];
    TrCount++;
} //end if
else
{
    StopTrans();
    TrCount=0;
        if (Change_Parametrs_Is_Recieved == 1)
        {
                Change_Parametrs_Is_Recieved = 0;
                UBRRHi = Hi(Baud_Divider);
                UBRRLow = Low(Baud_Divider);
        }

} //end else
}//end ISR(USART_UDRE_vect)
```

Рисунок 46 – Листинг измененного вектора обработчика прерывания USART_UDRE_vect

В момент, когда модуль USART отправил весь байт и буфер пуст, вызывается прерывание по вектору USART_UDRE_vect. Выполнение программы зависит от того сколько осталось байт отправить.

В случае, когда номер текущего передаваемого байта меньше общего количества байт, в буфер модуля USART помещается следующий байт и номер текущего байта увеличивается на единицу.

Иначе передача данных прекращается. Далее если пришла команда на изменение текущей скорости передачи данных, в регистры UBRRH и UBBRL записываются соответствующие значения, определяющие скорость передачи данных.

4 ОТЛАДКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В данном разделе описывается процесс загрузки и отладки программного обеспечения для микроконтроллера.

Для загрузки и отладки используется следующий набор программного обеспечения:

- программа-загрузчик AVR-Flash [7];
- терминальная программа Termite [14].

4.1 Загрузка программного обеспечения

Для загрузки программного обеспечения в память микроконтроллера используется программа-загрузчик AVR-Flash вместе с внутрисхемным программатором mikroProg. На рисунке 47 представлено окно программы AVR-Flash с правильными настройками программирования.

Для загрузки программы необходимо выбрать микроконтроллер из выпадающего списка в области Device, произвести настройку тактовой частоты в области Device frequency, и битов конфигурации в области FUSE Bits.

Далее необходимо нажать на кнопку Load в области CODE и в появившемся окне указать путь к файлу встраиваемого программного обеспечения в формате .hex. Данный файл находится в папке Debug исходного проекта.

После указания пути к файлу необходимо нажать кнопку Write. На данном этапе загрузка программного обеспечения в память микроконтроллера завершается.

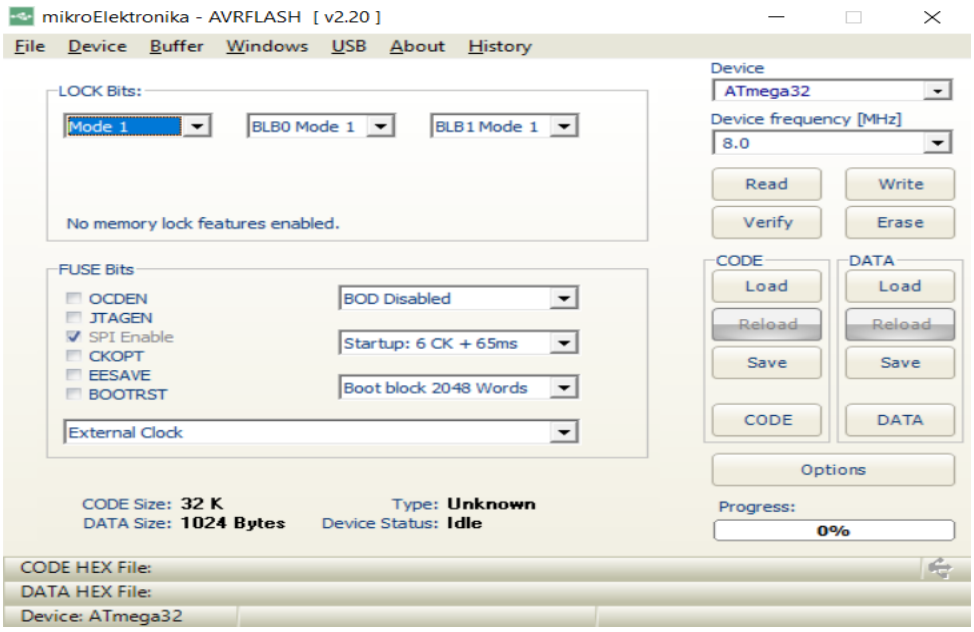

Рисунок 47 – Окно программы AVR-Flash с правильными настройками программирования

4.2 Отладка программного обеспечения

При включении питания контроллера по умолчанию температура выводится на индикатор в градусах Цельсия. При нажатии на кнопки C, K или F температура будет выводится в градусах Цельсия, Кельвинах или градусах Фаренгейта соответственно. На рисунках 48 – 50 представлены изображения графического интерфейса прибора при нажатии на соответствующие кнопки.

Рисунок 48 – Изображение на графическом индикаторе после нажатия кнопки C

Рисунок 49 – Изображение на графическом индикаторе после нажатия кнопки K

Рисунок 50 – Изображение на графическом индикаторе после нажатия кнопки F

Для отладки протокола Modbus RTU используется терминальная программа Thermite. На рисунке 51 представлено окно программы Thermite.

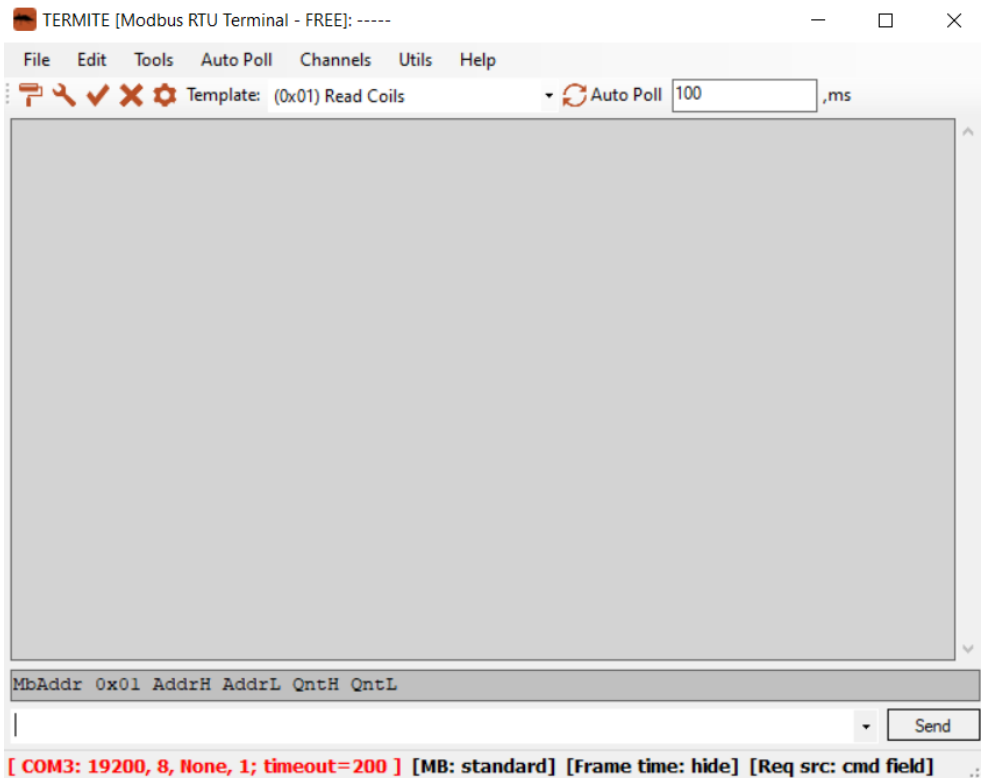

Рисунок 51 – Окно программы Thermite

Настройка COM-порта производится во вкладке Tools меню COM Port. В данном меню необходимо выбрать порт и задать настройки скорости передачи. По умолчанию скорость передачи равна 9600 бод. После настройки COM-порта терминал готов к работе. В строку необходимо ввести пакет Modbus RTU без контрольной суммы. На рисунках 52 – 55 представлены примеры считывания и изменения различных параметров устройства.

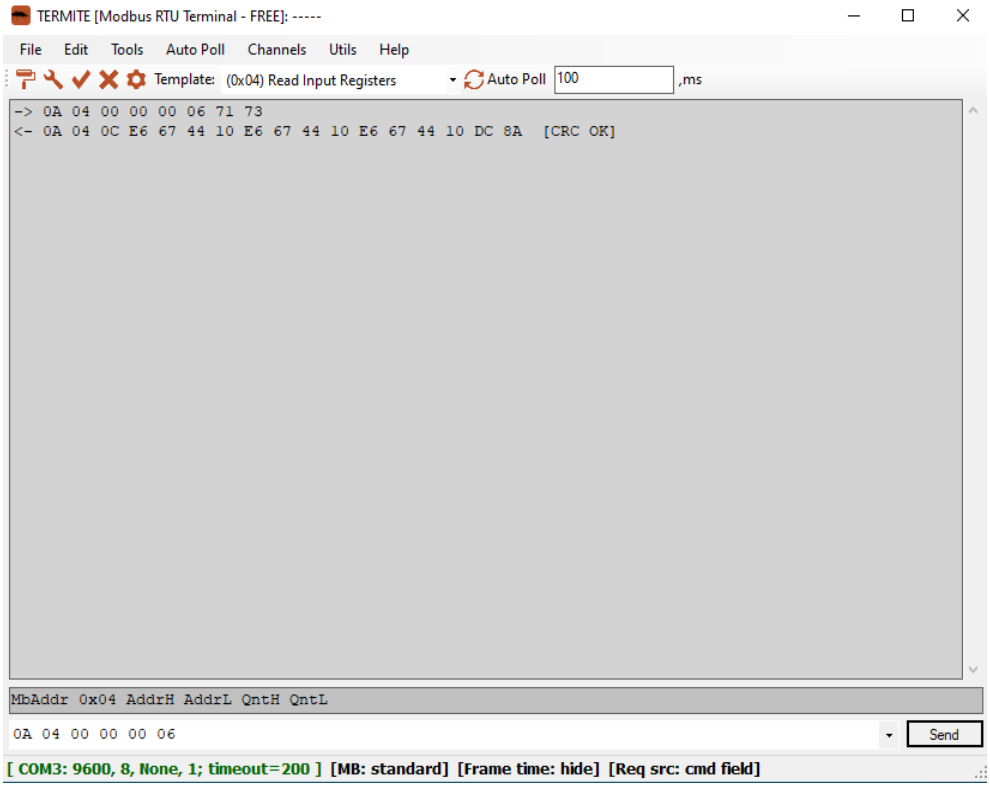

Рисунок 52 – Пример считывания температуры в трех единицах измерения одновременно

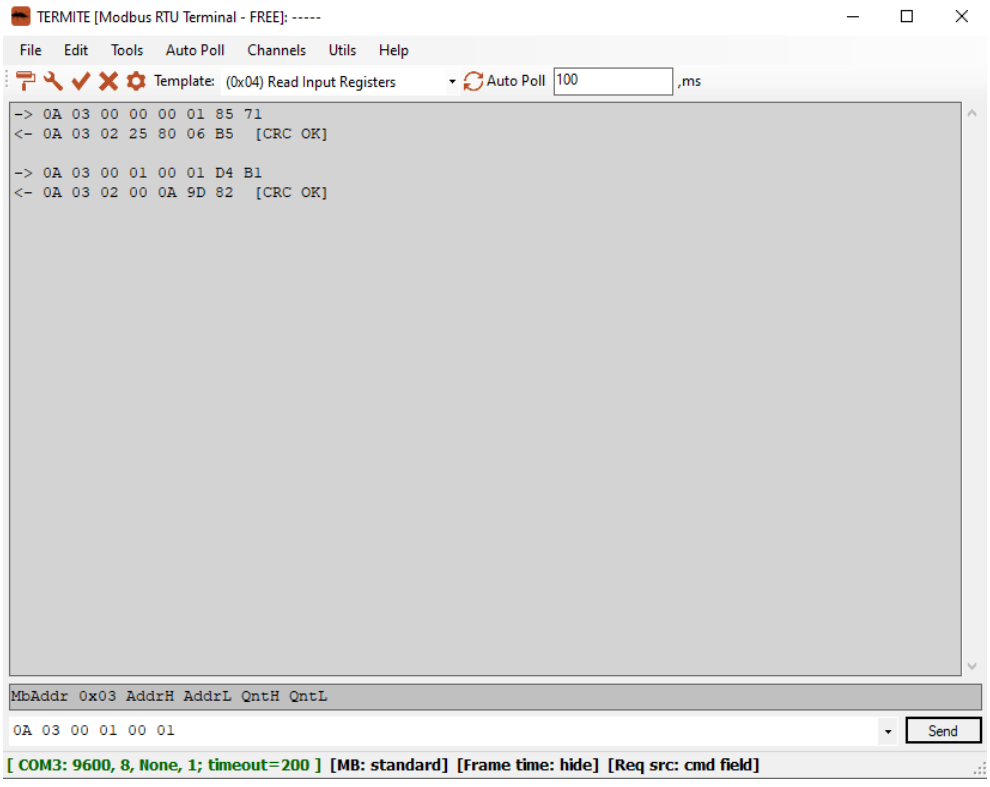

Рисунок 53 – Пример считывания текущей скорости передачи данных и адреса соответственно

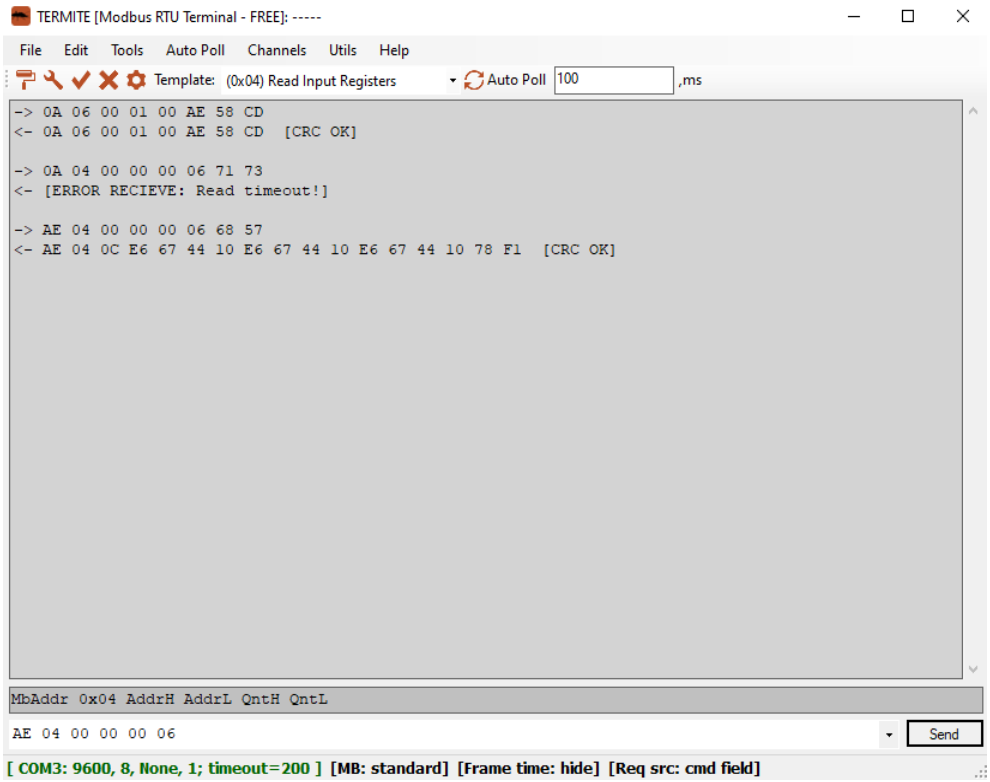

Рисунок 54 – Пример изменения адреса устройства

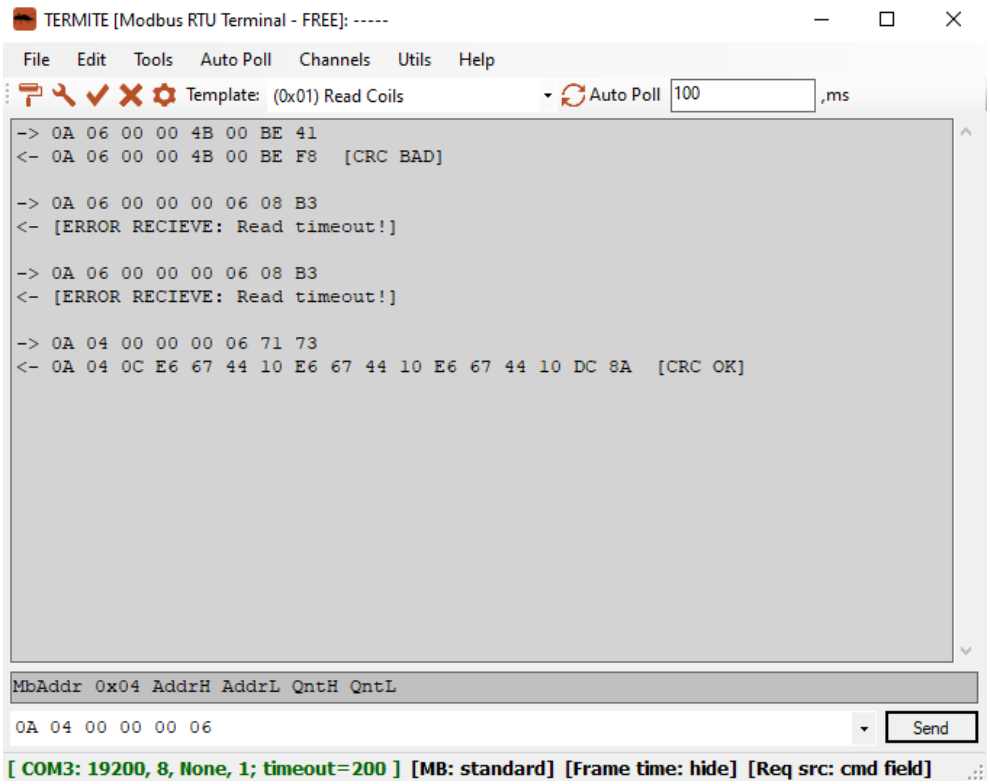

Рисунок 55 – Пример изменение параметра скорости передачи данных

В таблице 9 представлена расшифровка нескольких пакетов с запросом температуры при различной величине тестового аналогового сигнала.

Таблица 9 – Тестовая последовательность пакетов Modbus RTU

Значение температуре на индикаторе, °C	Полученный пакет Modbus RTU	Расшифрованное значение температуры, °C
-200,0	0A 04 04 C3 48 00 00 16 FD	-200
-151,6	0A 04 04 C3 17 99 9A 3F 27	-156,6
-39,0	0A 04 04 C2 1C 00 00 3A BD	-39,0
-0,6	0A 04 04 BF 19 99 9A 6C 5F	-0,6
29,9	0A 04 04 41 EF 33 33 68 30	29,9
77,6	0A 04 04 42 9B 33 33 36 70	77,6
109,6	0A 04 04 42 DB 33 33 E2 71	109,6
151,9	0A 04 04 43 17 E6 67 4E EF	151,900009
564,8	0A 04 04 44 0D 33 33 92 90	564,8

Таблица 9 показывает, что при передаче данных по протоколу Modbus RTU заявленная точность измерения сохраняется вопреки тому, что возникает погрешность, связанная с представлением числа с плавающей точкой. Тем не менее, вносимая погрешность существенно меньше, чем погрешность прибора.

ЗАКЛЮЧЕНИЕ

В ходе данной работы было разработано программное обеспечение для интеллектуального датчика температуры на базе платинового термопреобразователя сопротивления вследствие выполнения следующего перечня работ:

- разработаны структурная схема и обобщенный алгоритм работы устройства;
- разработан алгоритм вычисления измеряемого параметра;
- разработано программное обеспечение для микроконтроллера ATmega 32;
- произведена отладка программного обеспечения для микроконтроллера.

По итогам тестирования программного обеспечения для микроконтроллера выявлено, что программное обеспечение соответствует всем пунктам технического задания.

По итогам выполнения курсовой работы получены навыки разработки и отладки программного обеспечения для микроконтроллеров с ядром AVR, сопряженные с изучением технической документации и стандартов. Помимо этого, освоены методы вычисления измеряемого параметра с линейной выходной характеристикой и методы отладки устройств, использующих промышленные протоколы передачи данных.

Полученные компетенции являются востребованными для разработчика современных промышленных приборов и систем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ДТСxx5М.RS термосопротивления с цифровым интерфейсом RS-485 [интернет ресурс]. Сайт компании Овен. – Режим доступа: https://owen.ru/product/dtsxx5m_rs, свободный. (дата обращения 19.05.2023)

2. Функции и предназначение АСУ ТП [интернет ресурс]. Сайт компании Альянс Автоматика. – Режим доступа: https://a-automation.ru/asu-tp/, свободный. (дата обращения 19.05.2023)

3. Описание платформы EasyAVR v7 Development System фирмы «MikroElektronika» [интернет ресурс]. Сайт фирмы MicroElektronika d.o.o. – Режим доступа: https://www.mikroe.com/easyavr, свободный. (дата обращения 19.05.2023)

4. Техническая документация на микроконтроллер ATmega 32 [интернет ресурс]. Сайт фирмы Microchip Tehnology Inc. – Режим доступа: https://www.microchip.com/en-us/product/ATmega32, свободный. (дата обращения 19.05.2023)

5. Статья «Как общаются машины: протокол Modbus» [интернет ресурс]. Блог фирмы Advantech IIOT. – Режим доступа https://habr.com/ru/companies/advantech/articles/450234/, свободный. (дата обращения 19.05.2023)

6. Руководство пользователя платформы EasyAVR v7 Development System фирмы «MikroElektronika» [интернет ресурс]. Сайт фирмы MicroElektronika d.o.o. – Режим доступа: https://download.mikroe.com/documents/full-featured-boards/easy/easyavr-v7/easyavr-v7-manual-v101.pdf, свободный. (дата обращения 19.05.2023)

7. Руководство пользователя программы AVRFlash [электронный ресурс] – 361 с. Сайт фирмы MikroElektronika d.o.o. – Режим доступа: https://download.mikroe.com/documents/full-featured-boards/easy/easyavr-v6/avrflash-manual-v101.pdf, свободный. (дата обращения 19.05.2023)

8. Техническая документация на контроллер графического индикатора NT7108 [интернет ресурс]. Сайт фирмы MikroElektronika d.o.o. – Режим доступа: http://download.mikroe.com/documents/datasheets/lcd-driver-nt7108c-datasheet.pdf, свободный. (дата обращения 19.05.2023)

9. ГОСТ 6551 – 2009. Государственная система обеспечения единства измерений. Термопреобразователи сопротивления из платины, меди, никеля. Общие требования и методы испытаний. – Взамен ГОСТ 6651 – 94; введ. 01.01.2011. Межгосударственный Совет по стандартизации, метрологии и сертификации. Москва: Стандартинформ, 2019.

10. Н.М. Сафьянников, О.И. Буренева, А.Н. Алипов. Информационно-измерительные преобразователи киберфизических систем: учебное пособие. – Санкт-Петербург: Лань, 2020. – 234 с.

11. Руководство пользователя среды разработки Microchip Studio [интернет ресурс] – 360 с. Сайт фирмы Microchip. – Режим доступа: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/UserGuides/Microchip-Studio-UserGuide-DS50002718.pdf, свободный. (дата обращения 19.05.2023)

12. Библиотека с открытым исходным кодом AVR_Modbus [интернет ресурс]. Сервис размещения репозиториев с открытым исходным кодом GitHub. – Режим доступа: https://github.com/AntukhSerg/AVR_ModBus, свободный. (дата обращения 19.05.2023)

13. Библиотека с открытым исходным кодом glcd [интернет ресурс]. Сайт osamasLab. – Режим доступа: https://www.sites.google.com/site/osamaslab/Home/projects-list/glcd-library#_Toc27406744, свободный. (дата обращения 19.05.2023)

14. Руководство пользователя терминальной программы Thermite [интернет ресурс]. – 18 с. Сайт фирмы S2-Team. – Режим доступа: http://s2-team.ru/wrkrs/prods/modbus-tools/termite/, свободный. (дата обращения 19.05.2023)

ПРИЛОЖЕНИЕ А

Текст программы

Файл main.c

```
#define F_CPU 8000000UL
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "Modbus/AVR_ModBus.h"
#include "GLCD/glcd.h"
#include "Temperature Calculating/t_calc.h"
extern volatile int16_t C_temp_code;
extern volatile int16_t K_temp_code;
extern volatile int16_t F_temp_code;
#define base 20 //отступ от края дисплея
volatile uint8_t g_key = 0;             //номер нажатой клавиши
volatile uint8_t g_key_status = 0;      //Если бит 7 равен 1, значит клавиша нажата.
                                        //Если бит 6 равен 1, значит клавиша не отпущена
volatile uint8_t mode = 0; //Режим отображения температуры
/*
1. mode == 0 - температура в градусах Цельсия
2. mode == 1 - температура в Кельвинах
3. mode == 2 - температура в градусах Фаренгейта
*/
volatile uint8_t upd_flag = 0; //флаг, сигнализирующий об обновлении данных
volatile uint8_t TCI_counts = 0; //количество прерываний таймера 2
RegNum4x[0] = BAUD_RATE;       //скорость передачи по умолчанию
RegNum4x[1] = DEFAULT_SLAVE_ID; //адрес устройства по умолчанию
void glcd_init(void);
void draw_loading_screen();
void draw_main_screen(void);
```

```
void display_result(void);
void IO_init(void);
void timer_init(void);
void ADC_init(void);
void scan_key(void);
void draw_round_rectangle(uint8_t x1, uint8_t x2, uint8_t y1, uint8_t y2, uint8_t radius);
int main(void){
asm("cli");
        IO_init();
        timer_init();
        ADC_init();
        InitModBus();
        glcd_init();
        draw_main_screen();
        asm("sei");
    while (1) {
        if(upd_flag){
                scan_key();
                get_temp();
                display_result();
                upd_flag = 0;
        }
        if ((g_key_status &0b10000000) != 0){//Если клавиша нажата
                g_key_status &= 0b01000000; //сбросить бит, отвечающий за факт нажатия
                switch (g_key){//поменять режим в зависимости от нажатой кнопки
                        case 0:{
                                mode = 0;
                                glcd_puts("°C",base+8*9,1,0,1,1);
                                break;
                        }
                        case 1:{
                                mode = 1;
                                glcd_puts(" K", base+8*9,1,0,1,1);
                                break;
                        }
```

```
                        case 2:{
                                mode = 2;
                                glcd_puts("°F",base+8*9,1,0,1,1);
                                break;
                        }
                }
        }
        CheckModBus();
   }
}
void IO_init(void){
        DDRA |=0b00001100;          //инициализация портов
        PORTA |=0b00000100;         //для сенсерной панели управления
        DDRB |=0b11100011;
        PORTB |=0b00000000;
        DDRD |=0b11000000;
        PORTD |=0b10000000;
        DDRC |=0b11111111;
}
void timer_init(void){
        //настройка на срабатывание T/C2 с интервалом 8 мс
        TCNT2=0b00000000;           //очистка T/C2
        OCR2=249;                   //запись константы 249 в регистр сравнения T/C2
        TCCR2=0b00001110;           //режим "сброс при совпадении" и делитель на 256
        TIMSK|=0b10000000;          //разрешение прерывания по совпадению от T/C2
        TIFR &=0b11000000;          //очистка флагов прерываний T/C2
}
void ADC_init(void){
        ADMUX = 0b00000000;
        ADCSRA = 0b10000111; //ADEN | предделитель АЦП 128
};
//функция, реализующая рекомендованную последовательность инициализации дисплея
void glcd_init(void){
        _delay_ms(100);
        glcd_off();                 //выключаем дисплей
```

```
        _delay_ms(100);
        glcd_on();              //включаем после задержки
        glcd_clear();           //очистка дисплея
}
//функция отрисовки заставки
void draw_main_screen(){
        draw_round_rectangle(0,0,126,62, 8);        //отрисовка внешнего контура
        // вывод границ виртуальных "кнопок"
        draw_round_rectangle(5,35,40,59, 8);
        draw_round_rectangle(86,35,121,59,8);
        draw_round_rectangle(45,35,81,59,8);
        // вывод текстовой заставки
        draw_round_rectangle(15,5,113,17, 6);
        glcd_puts("T=",base,1,0,1,1);
        glcd_puts("°C", base + 8 *9, 1, 0, 1, 1);
        // Размещаем буквы в центр виртуальных "кнопок"
        glcd_puts("C",16,5,0,2,1);
        glcd_puts("K",57,5,0,2,1);
        glcd_puts("F",97,5,0,2,1);
}
void display_result(void){
        int16_t code_temp = 0;
        switch(mode){
                case 0:{
                        code_temp = C_temp_code;
                        break;
                }
                case 1: {
                        code_temp = K_temp_code;
                        break;
                }
                case 2:{
                        code_temp = F_temp_code;
                        break;
                }
```

```
		}
		if (code_temp < 0){
			glcd_putchar('-', base+8*3, 1, 1, 1);
			code_temp = -code_temp;
		}
		else { glcd_putchar(' ', base+8*3, 1, 1, 1); }
		uint8_t nums[4] = {0,};
		for (int8_t i = 3; i > -1; --i) { //преобразуем число в массив. в ячейке [0] лежит старший разряд
			nums[i] = code_temp % 10 + 48;
			code_temp = code_temp / 10;
		}
		uint8_t cursor = 4;//текущая координата виртуального курсора
		uint8_t count = 0; //количество незначащих разрядов
		while(count < 2){
			if (nums[count]== 48) { ++count; }
			else{ break; }//увеличиваем счетчик до первого ненулевого значения или до двух
		}
		for (uint8_t i = count; i < 3; ++i){ glcd_putchar(nums[i], base+8*(cursor++), 1, 1, 1);}
		glcd_putchar('.', base + 8 * (cursor++), 1,0,1); //разделительная точка
		glcd_putchar(nums[3], base + 8*(cursor++), 1, 0, 1);
		for (uint8_t i = 0; i < count; ++i){glcd_putchar(' ', base + 8*(cursor++), 1,0,1);}//заполняем пробелами оставшиеся символы, если они есть
	}
	void scan_key(void){
		PORTA=0b00000100;
		_delay_ms(2);			// Задержка !!! (паразитная емкость должна зарядиться)
		ADMUX=0b00000000;		// Выбираем нулевой канал АЦП
		ADCSRA|=(1<<ADSC);		// Запуск АЦП
		while ((ADCSRA&(1<<ADSC))!=0);	// Ожидаем конца преобразования АЦП
		volatile uint16_t x_coordinate=ADC;
		PORTA=0b00001000;
		_delay_ms(2);
		ADMUX=0b00000001;		// Смена канала
```

```
		ADCSRA|=(1<<ADSC);		// Запуск АЦП
		while ((ADCSRA&(1<<ADSC))!=0);		// Ожидаем конца преобразования АЦП
		volatile uint16_t y_coordinate=ADC;
		if	(((y_coordinate>190)&&(y_coordinate<370))&&((x_coordinate>80)&&(x_coordinate<260))&&(g_key_status==0)){ //Регистрация нажатия кнопки "C"
			g_key_status=0b11000000;
			g_key=0;
		}
		if	(((y_coordinate>190)&&(y_coordinate<370))&&((x_coordinate>330)&&(x_coordinate<490))&&(g_key_status==0)){ //Регистрация нажатия кнопки "K"
			g_key_status=0b11000000;
			g_key=1;
		}
		if	(((y_coordinate>190)&&(y_coordinate<370))&&((x_coordinate>565)&&(x_coordinate<730))&&(g_key_status==0)){ //Регистрация нажатия кнопки "F"
			g_key_status=0b11000000;
			g_key=2;
		}
		if (((y_coordinate<20)&&(x_coordinate<20))&&(g_key_status&0b01000000)!=0){ //если нет замыкания между слоями панели после нажатия
			g_key_status=g_key_status&0b10000000; //значит виртуальная кнопка отпущена
		}
	}
	ISR(TIMER2_COMP_vect){
		++TCI_counts;
		if(TCI_counts == 10){
			TCI_counts = 0;
			upd_flag = 1;
		}
	}
	void draw_round_rectangle(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t radius){
		int16_t tSwitch = 3 - 2 * radius;
		uint8_t width, height, x, y;
		width = x2-x1;
		height = y2-y1;
```

```
        x = 0;
        y = radius;
        h_line(x1 + radius, y1, width - 2 * radius, 0, 1);
        h_line(x1 + radius, y2, width - 2 * radius, 0, 1);
        v_line(x1, y1 + radius, height - 2 * radius, 0, 1);
        v_line(x2, y1 + radius, height - 2 * radius, 0, 1);
         while (x <= y) {
                point_at(x1 + radius - x, y1 + radius - y, 1);
                point_at (x1 + radius - y, y1 + radius - x, 1); //левый верхний угол
                 point_at(x1 + width - radius + x, y1 + radius - y, 1);
                 point_at (x1 + width - radius + y, y1 + radius - x, 1); //правый верхний угол
                 point_at(x1 + radius - x, y1 + height - radius + y, 1);
                 point_at (x1 + radius - y, y1 + height - radius + x, 1); //левый нижний угол
                 point_at(x1 + width - radius + x, y1 + height - radius + y, 1);
                 point_at (x1 + width - radius + y, y1 + height - radius + x, 1); //правый нижний
угол
                if (tSwitch < 0){tSwitch += 4 * x + 6;}
                else{
                        tSwitch += 4 * (x-y) + 10;
                        --y;
                }
                ++x;
         }
}
```

Файл t_calc.h

```
#ifndef T_CALC_H_
#define T_CALC_H_
#include <avr/io.h>
#include <avr/interrupt.h>
#define ADCfactorK_C 512500LL          //коэффициенты для целочисленного
вычисления
```

```
#define ADCfactorB_C -131072000LL      //значения температуры в градусах цельсия
#define ADCfactorK_F 922500LL          //Коэффициенты для целочисленного вычисления
#define ADCfactorB_F -214958080LL      //значения температуры в градусах Фаренгейта
union temperature_t{
        float f_cell;  //Часть объединения, в которое нужно записать
                       //вещественное значение температуры для передачи
                                       //по протоколу ModBus RTU
        uint16_t buff[2]; //Это поле нужно для передачи всего значения с помощью
                          //обработчика Modbus RTU
};
//инициализация структур данных для хранения температуры
extern volatile union temperature_t temp_C;
extern volatile union temperature_t temp_K;
extern volatile union temperature_t temp_F;
//переменные для вывода на графический индикатор
extern volatile int16_t C_temp_code;
extern volatile int16_t K_temp_code;
extern volatile int16_t F_temp_code;
void get_temp(void); //Прототип функции измерения температуры
#endif /* T_CALC_H_ */
```

Файл t_calc.c

```
#include "t_calc.h"
#include "AVR_ModBus.h"
//инициализация структур данных для хранения температуры
volatile union temperature_t temp_C = {0};
volatile union temperature_t temp_K = {0};
```

```
volatile union temperature_t temp_F = {0};
//переменные для вывода на графический индикатор
volatile int16_t C_temp_code = 0;
volatile int16_t K_temp_code = 0;
volatile int16_t F_temp_code = 0;
void get_temp(void){
	ADMUX=0b00000101;
	ADCSRA|=(1<<ADSC);		// запуск АЦП;
	while ((ADCSRA&(1<<ADSC))!=0);//ожидание конца преобразования АЦП
	C_temp_code = ((int32_t)ADCfactorK_C * ADC + (int32_t)ADCfactorB_C) >> 16;
	temp_C.f_cell = (float)C_temp_code * 0.1f;
	for (uint8_t i = 0; i < 2; ++i){//байты хранятся в памяти мк в порядке "от младшего к старшему"
		RegNum3x[i] = temp_C.buff[1 - i];
	}
	K_temp_code = C_temp_code + 2731; //переменные X_temp_value хранят значение в 10 раз больше фактического
	temp_K.f_cell = (float)K_temp_code * 0.1f;
	for (uint8_t i = 0; i < 2; ++i)
	{
		RegNum3x[i+2] = temp_K.buff[1 - i];
	}
	F_temp_code = ((int32_t)ADCfactorK_F * ADC + (int32_t)ADCfactorB_F) >> 16;
	temp_F.f_cell = (float)F_temp_code * 0.1f;
	for (uint8_t i = 0; i < 2; ++i)
	{
		RegNum3x[i + 4] = temp_F.buff[1 - i];
```

```
        }
}
```

Использованные функции из файла glcd.h

```
void glcd_on(){
#ifdef CS_ACTIVE_LOW
  CONTROLPORT2 &= ~CCS1;      //Activate both chips
  CONTROLPORT2 &= ~CCS2;
#else
  CONTROLPORT2 |= CCS1; //Activate both chips
  CONTROLPORT2 |= CCS2;
#endif
  CONTROLPORT &= ~DI;    //DI low --> command
  CONTROLPORT &= ~RW;  //RW low --> write
  DATAPORT = 0x3F;   //ON command
  trigger();
}
void glcd_off(){
#ifdef CS_ACTIVE_LOW
  CONTROLPORT2 &= ~CCS1;      //Activate both chips
  CONTROLPORT2 &= ~CCS2;
#else
  CONTROLPORT2 |= CCS1; //Activate both chips
  CONTROLPORT2 |= CCS2;
#endif
        CONTROLPORT &= ~DI;      //DI low --> command
        CONTROLPORT &= ~RW;     //RW low --> write
        DATAPORT = 0x3E; //OFF command
        trigger();
}
```

```
void point_at(unsigned int x,unsigned int y,byte color){
   byte pattern = 0;
   goto_xy(x,(int)(y/8));
   switch (color){
      case 0:        //Light spot
         pattern = ~(1<<(y%8)) & glcd_read(x);
      break;
      case 1:        //Dark spot
         pattern = (1<<(y%8)) | glcd_read(x);
      break;
   }
   goto_xy(x,(int)(y/8));
   glcd_write(pattern);
}
void h_line(unsigned int x,unsigned int y,byte l,byte s,byte c){
   int i;
   for(i=x; i<(l+x); i += (byte)(s+1))
      point_at(i,y,c);
}
void v_line(unsigned int x,unsigned int y,signed int l,byte s,byte c){
   unsigned int i;
   for(i=y; i<(y+l); i += (byte)(s+1))
      point_at(x,i,c);
}
void glcd_putchar(byte c,int x,int y,byte l,byte sz){
   if(l == 1){
        //if(c == 129) c = 250;
        switch(c){
        case 129:
                c = 250;
```

```
                break;
            case 144:
                c = 251;
                break;
            case 152:
                c = 252;
                break;
            case 142:
                c = 253;
                break;
            case 141:
                c = 254;
                break;
            }
           if((c >= 193) && (prevLet >= 193) && (pgm_read_byte(&(map[prevLet-193][5]))) && (pgm_read_byte(&(map[c-193][4]))) ){
              putItSz(pgm_read_byte(&(map[prevLet-193][stat+1])),prevX,prevY,sz);
              stat = 2;
           }else stat = 0;
           if(c >= 193) putItSz(pgm_read_byte(&(map[c-193][stat])),x,y,sz);
           else putItSz(c,x,y,sz);
           prevLet = c;
           prevX = x;
           prevY = y;
        }else putItSz(c,x,y,sz);
    }
    void glcd_puts(char *c,int x,int y,unsigned char l,byte sz,signed char space){
      char i = 0;
      char special = 0;
      while((i<strlen(c)) && l==0){
```

```
  glcd_putchar(*(c+i),x,y,0,sz);
  x += (8+space)*sz;
  if(x>128-8*sz){
    x=0;
    y += sz;
  }
  i++;
}
while((i<strlen(c)) && l==1){
  if((*(c+i) == 225) && (*(c+i+1) == 199))
    special = 249;
  else if((*(c+i) == 225) && (*(c+i+1) == 195))
    special = 231;
  else if((*(c+i) == 225) && (*(c+i+1) == 194))
    special = 232;
  else if((*(c+i) == 225) && (*(c+i+1) == 197))
    special = 233;
  if(special){
    glcd_putchar(special,x-8*sz,y,1,sz);
    i+=2;
    x -= 8*sz;
    special = 0;
  }
  else{
            glcd_putchar(*(c+i),x-8*sz,y,l,sz);
            if(*(c+i) == 32){                  //If space (i.e. new word)
                  x -= (8+space)*sz;                        //Space between words
            }
            else{ x -= 8*sz;}
            i++;
```

```
      }
      if(x < 8*sz+1){
        x=128-8*sz;
        y += sz;
      }
    }
    prevLet = 193;
  }
```

Модификации библиотеки AVR_Maodbus

```
#define QUANTITY_REG_0X 0 //количество дискретных выходов (DO)
#define QUANTITY_REG_1X 0 //количество дискретных входов (DI)
#define QUANTITY_REG_3X 6 //количество аналоговых входов (AI)
#define QUANTITY_REG_4X 2 //количество аналоговых выходов (AO)
volatile unsigned char Slave_ID = DEFAULT_SLAVE_ID;
volatile unsigned char Change_Parametrs_Is_Recieved = 0;//Если равна 1, значит пришел пакет с новыми настройками скорости передачи устройства
volatile unsigned char Baud_Divider = BAUD_DIVIDER;//текущий делитель для регистров UBRR

char Func06(void){
//проверка корректного адреса в запросе
if(!(reg_adress<=QUANTITY_REG_4X)) return ErrorMessage(0x02); //Адрес данных, указанный в запросе, недоступен
//проверка корректных данных в запросе
//формируем ответ, возвращая полученное сообщение
if((reg_adress == 0) && ((value == 9600) || (value == 14400)
|| (value == 19200) || (value == 38400) || (value == 57600)))//при необходимости поставить контроль значений{
      RegNum4x[reg_adress] = value;
```

```
            Change_Parametrs_Is_Recieved = 1;
            cmTrBuf0[1] = cmRcBuf0[1];
            cmTrBuf0[2] = cmRcBuf0[2];
            cmTrBuf0[3] = cmRcBuf0[3];
            cmTrBuf0[4] = cmRcBuf0[4];
            cmTrBuf0[5] = cmRcBuf0[5];
            cmTrBuf0[6] = cmRcBuf0[6];
            cmTrBuf0[7] = cmRcBuf0[7];
            Baud_Divider = F_CPU/(RegNum4x[reg_adress] * 16.0) - 1;
            return 8;
        } //end if()
        else if ((reg_adress == 1) && (value < 248)){
            RegNum4x[reg_adress] = value;
            cmTrBuf0[1] = cmRcBuf0[1];
            cmTrBuf0[2] = cmRcBuf0[2];
            cmTrBuf0[3] = cmRcBuf0[3];
            cmTrBuf0[4] = cmRcBuf0[4];
            cmTrBuf0[5] = cmRcBuf0[5];
            cmTrBuf0[6] = cmRcBuf0[6];
            cmTrBuf0[7] = cmRcBuf0[7];
            Slave_ID = RegNum4x[reg_adress];
            return 8;
        }
        else{
            return ErrorMessage(0x03);//Значение, содержащееся в поле данных за-
проса, является недопустимой величиной
        }
        return 0;
        } //end Func06(void)
```