МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И СЕТЕЙ

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

доцент, канд. техн. наук		Т. Н. Соловьева
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ

ЦИФРОВОЙ ВОЛЬТМЕТР

по дисциплине: МИКРОКОНТРОЛЛЕРНЫЕ СИСТЕМЫ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4143		А. М. Гридин
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА

студенту группы 4143 (номер) Гридину Артёму Максимовичу (фамилия, имя, отчество)

на тему Цифровой вольтметр

Цель проекта: разработка аппаратно-программного комплекса цифрового вольтметра на базе микроконтроллера серии ARM в системе автоматизированного проектирования Proteus.

Задачи, подлежащие решению: реализация возможности выбора диапазона измеряемого напряжения,
определение падения напряжения между двумя выводами вольтметра с учетом выбранного диапазона,
отображения измеренного напряжения на жидкокристаллическом дисплее.

Содержание пояснительной записки (основные разделы):
проектирование аппаратного обеспечения,
проектирование программного обеспечения,
тестирование и отладка аппаратно-программного комплекса.

Срок сдачи работы « 30 » ноября 2024

Руководитель

доц., канд. техн. наук		Т.Н. Соловьева
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

Задание принял к исполнению

студент группы № 4143		А. М. Гридин
	подпись, дата	инициалы, фамилия

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире, где электроника и автоматизация играют ключевую роль в различных отраслях, точные измерения электрических параметров становятся необходимостью для обеспечения надежности и эффективности работы устройств. Одним из основных инструментов для измерения электрического напряжения является вольтметр. С развитием технологий и широким распространением микроконтроллеров, появилась возможность создания цифровых вольтметров, которые обладают высокой точностью, удобством использования и возможностью интеграции с другими системами.

Целью данной курсовой работы является разработка аппаратно-программного комплекса цифрового вольтметра на базе микроконтроллера серии ARM в среде автоматизированного проектирования Proteus. Данный комплекс позволит измерять напряжение с использованием возможностей микроконтроллера и выводить результаты на жидкокристаллический дисплей.

Для достижения поставленной цели необходимо решить следующие задачи:

- разработать аппаратную часть устройства, включая выбор и подключение АЦП и дисплея;
- создать программное обеспечение для микроконтроллера, обеспечивающее считывание данных с АЦП и их отображение;
- провести тестирование и отладку комплекса для проверки его работоспособности и точности измерений.

Создание цифрового вольтметра не только способствует углублению знаний в области электроники и программирования, но и открывает возможности для дальнейших разработок в области автоматизации и контроля, поэтому данная тема является актуальной.

1 Проектирование аппаратного обеспечения

1.1 Обзор существующих решений

В настоящее время существует огромное количество решений, которые подойдут для любого случая. Вольтметры могут классифицироваться по таким факторам, как [1]:

- Принцип действия;
- Конструкция;
- Назначение.

По принципу действия вольтметры делят на:

- Электромеханические
- Электронные

Измерения прибором первого типа основываются на взаимодействии конструктивных механизмов (рамки, стрелки, металлические наконечники) и напряжения. В результате взаимодействия появляется электромагнитное поле, которое с помощью стрелки отображает полученный результат. Таким образом входное переменное напряжение преобразовывается в постоянное. Чем больше величина входного напряжения, тем больше отклоняется стрелочный указатель.

Цифровой вольтметр считается более точным по сравнению с аналоговыми приборами. Работа устройства основана на трансформировании входного напряжения в цифровой показатель, отображающийся на дисплее. При этом точность результата зависит от того, какой дискретностью обладает устройство. Технические и функциональные возможности универсальных цифровых вольтметров позволяют использовать прибор как на производстве, так и при проведении лабораторно-исследовательских и научных работ.

По назначению все измерительные устройства можно разделить на следующие типы [1]:

- Импульсные. Используются для замеров коротких импульсов.

- Селективные. Приборы этого типа применяют для избирательного выделения определенных гармоник в сложном сигнале.
- Вольтметры для работы в сетях переменного/постоянного тока.
- Фазочувствительные, или векторметры. Такие приборы применяют не только для определения комплексного напряжения, но и для измерения отдельных составляющих электрической цепи.
- Универсальные. Многофункциональный тип вольтметров, с помощью которого производят замеры ЭДС в разных видах сетей. Одним из преимуществ таких приборов считается небольшое потребление энергии при богатом наборе функциональных возможностей.

1.2 Выбор аппаратной реализации

В данном проекте для реализации цифрового термометра выбраны следующие компоненты: микроконтроллер STM32F103C8, встроенный с микроконтроллер аналого-цифровой преобразователь, жидкокристаллический дисплей LM032L и пара переключателей. Эта комбинация аппаратных средств позволяет точно измерять температуру и выводить результаты на дисплей.

1.3 Описание разработки схемы проекта

Разработка начинается с создания проекта в среде Proteus 8 Professional. Проект создается пустой, т.е. без указания каких семейств и конкретных моделей контроллера. Далее добавляем в панель компонентов все необходимые для работы компоненты (рисунок 1). Выбираются они в поисковом меню по нажатию на кнопку «P»

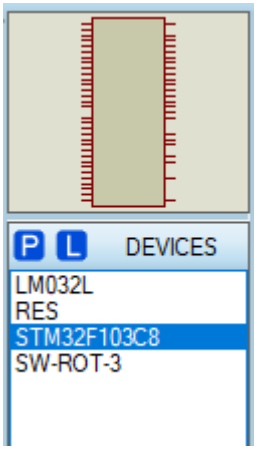

Рисунок 1 – Компоненты проекта

Теперь можно осуществить соединение микроконтроллера STM32 и LCD дисплея. Питание микроконтроллера обеспечивается с помощью земли на VSSA и напряжением на VDDA. На пины PB3-PB6 подключаем входы D4-D7. На пины PB7 и PB7 подключаем пины дисплея RS и E соответственно.

Питание дисплея обеспечивается заземлением VSS и подачей напряжения на VDD. Пин RW дисплея также заземляется, так как необходимо только выдавать данные на него.

Промежуточный результат подключений микроконтроллера и дисплея изображен на рисунке 2.

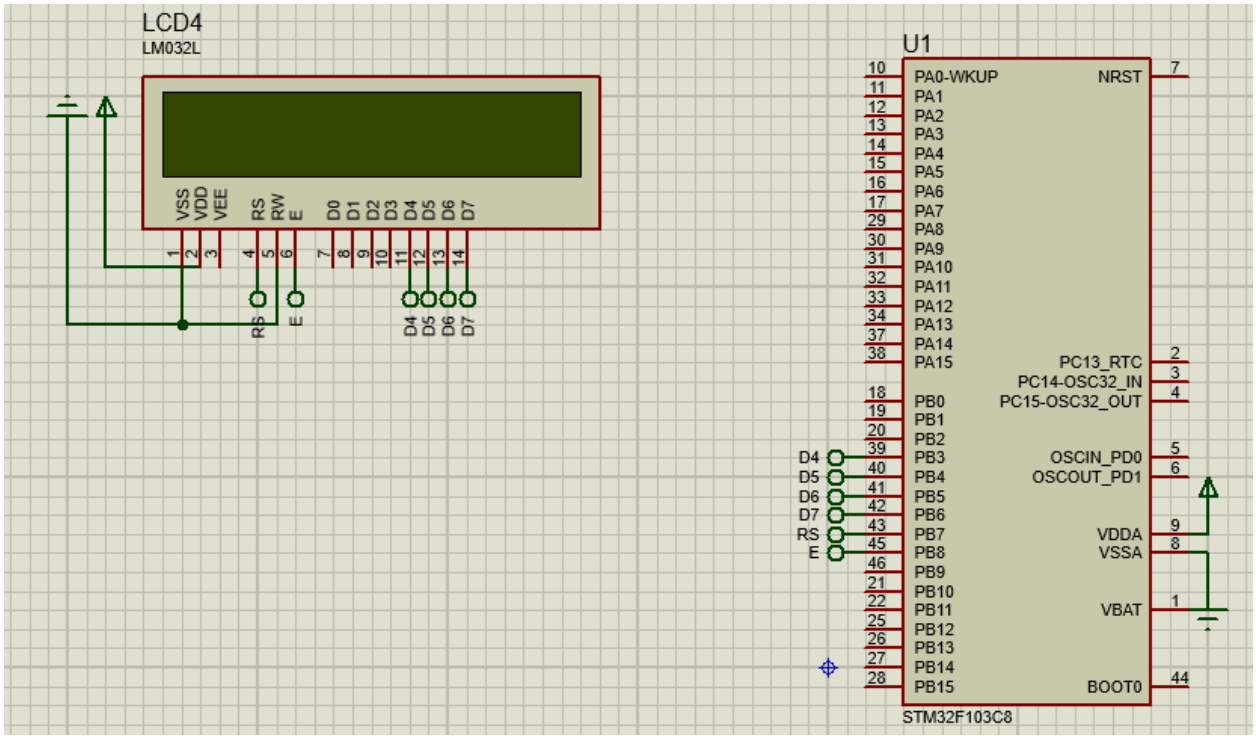

Рисунок 2 – Промежуточный результат подключений

Далее подключим ротаторы изменения режимов работы и отображения к пинам контроллера. Подаем к ротаторам напряжение, чтобы это напряжение приходило в назначенные пины. Для режимов работы это PB0-PB2, для режимов отображения это PA2-PA4 (рисунок 3).

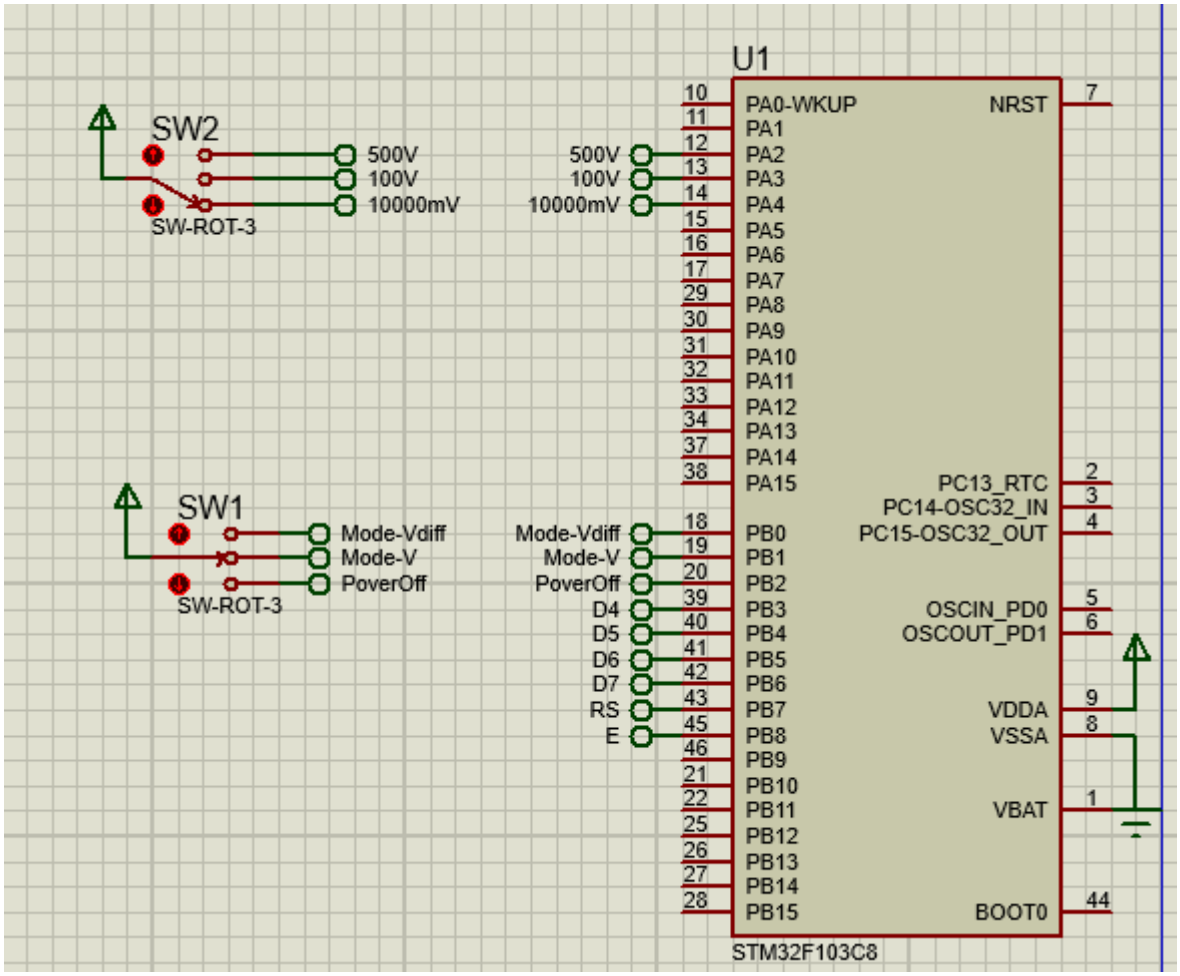

Рисунок 3 – Подключение ротаторов

Так как вольтметр должен измерять напряжение до 500 В, то нам понадобятся делители напряжения 1:100 на пины PA0 и PA1. Ток был выбран в 0,005 А, как безопасный для контроллера [2]. По закону Ома, R2= 5 В/0,005 А = 1000 Ом, а R1= 500 В/0,005 А – 1000 Ом = 99 кОм. Для тестирования проекта в будущем, добавим сейчас источники напряжения и тестеры напряжения (рисунок 4).

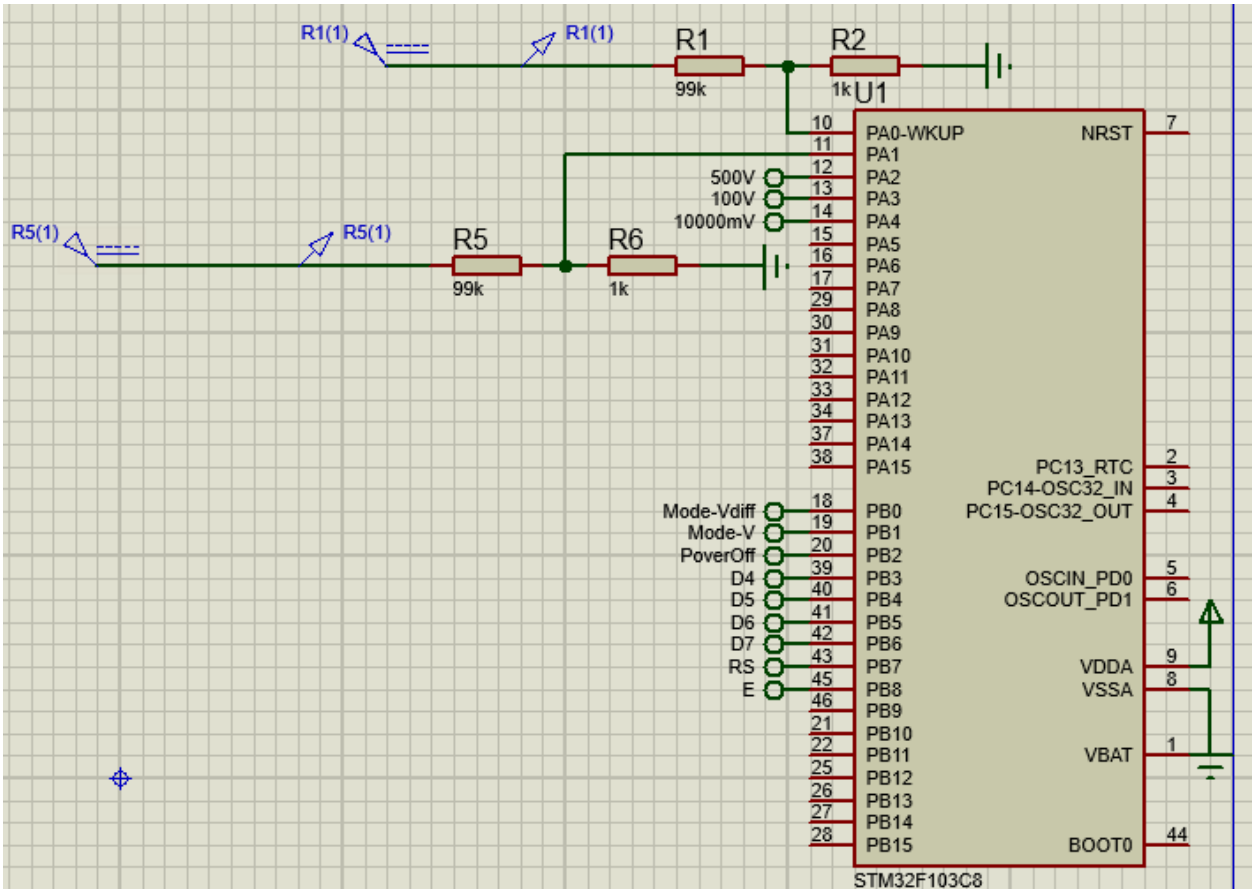

Рисунок 4 – Подключение делителей напряжения

Итоговая принципиальная схема проекта содержится в приложении А.

Подключения нужных пинов у компонентов приведены в таблицах 1-4.

Таблица 1 – Подключение микроконтроллера STM32F103C8

Пин:	Соединён с:
PA0	«Щуп» вольтметра с делителем напряжения
PA1	«Щуп» вольтметра с делителем напряжения
PA2	Режим отображения «500 В»
PA3	Режим отображения «100 В»
PA4	Режим отображения «10000 мВ»
PB0	Режим работы «Измерение падения напряжения»

PB1	Режим работы «Измерение напряжения»
PB2	Режим работы «Off»
PB3	D4 дисплея
PB4	D5 дисплея
PB5	D6 дисплея
PB6	D7 дисплея
PB7	RS
PB8	E
VDDA	+5V
VSSA	Земля

Таблица 2 – Подключение дисплея LM032L

Пин:	Соединён с:
VSS	Земля
VDD	+5V
RS	RS
RW	Земля
E	E
D4-D7	PB3-PB6 контроллера

Таблица 3 – Подключение ротатора SW1

Пин:	Соединён с:
Положение 1	Режим «Off»
Положение 2	Режим работы «Измерение напряжения»
Положение 3	Режим работы «Измерение падения напряжения»

Таблица 4 – Подключение ротатора SW2

Пин:	Соединён с:
Положение 1	Режим отображения «10000 мВ»
Положение 2	Режим отображения «100 В»
Положение 3	Режим отображения «500 В»

1.4 Характеристики элементов и их назначение

Микроконтроллер STM32F103C8 служит центральным элементом управления в проекте, обеспечивая связь между АЦП и дисплеем и обрабатывая данные в режиме реального времени. Поддержка прерываний позволяет своевременно считывать данные с АЦП. Характеристики [2]:

Ядро: ARM Cortex-M3

Тактовая частота: до 72 МГц

Память:

• Flash-память: 64 КБ

• SRAM: 20 КБ

Цифровые входы/выходы: до 37 GPIO (в зависимости от конфигурации)

Аналоговые входы: многоканальный 12-битный АЦП

Интерфейсы:

• USART (до 3)

• SPI (до 2)

• I2C (до 2)

• USB 2.0 Full Speed

Таймеры: несколько таймеров общего назначения и один таймер для управления ШИМ

Рабочее напряжение: 2.0 В - 5 В

Температурный диапазон: от -40°C до +85°C (коммерческий) или до +125°C (промышленный)

Корпус: LQFP-48

Жидкокристаллический дисплей LM032L используется для отображения текущих значений температуры. Характеристики [3]:

Формат 20x2: Дисплей поддерживает вывод до 20 символов на каждой из двух строк, что позволяет отображать достаточное количество данных.

4-битный режим передачи данных: Дисплей может работать в 4-битном режиме, что экономит порты микроконтроллера.

Стандартный набор команд: LM016L поддерживает стандартные команды LCD, такие как очистка дисплея, установка курсора и отправка данных.

2 Проектирование программного обеспечения

2.1 Описание назначения программы и процесса её разработки

Программа служит прошивкой для микроконтроллера. Для её создания необходимо назначить пины контроллера, настроить включение АЦП и написать саму программу. Для разработки прошивки была использована программа STM32CubeIDE.

Создадим новый проект, указав наш микроконтроллер. Перейдём к конфигурации пинов. Для работы АЦП во вкладке Analog >> ADC1 выберем IN0 и IN1 для взаимодействия с АЦП. Используем 2 канала с помощью параметра ADC_Regular_ConversionMode >> Number of Conversion >> 2. В Rank 1 и Rank 2 устанавливаем разные каналы. Для работы с ротаторами укажем нужные пины как GPIO_Input, а для работы с LCD дисплеем укажем нужные пины как GPIO_Output (рисунок 5). После настройки пинов, генерируем файл «main.c» с кодом и переходим на следующий этап разработки ПО.

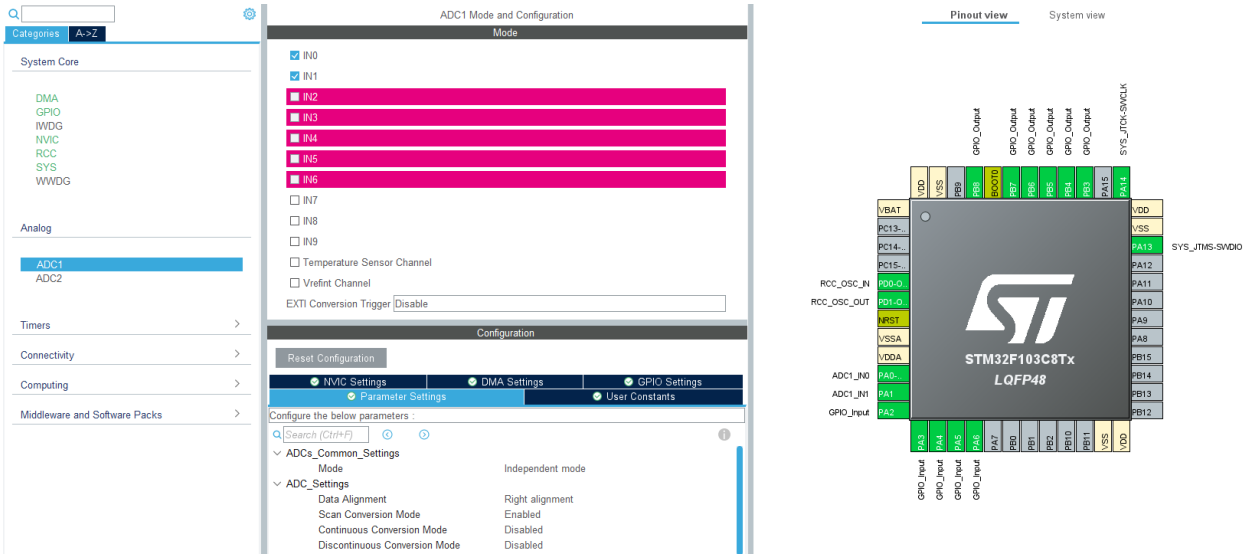

Рисунок 5 – Настройка пинов

В папке проекта создаем файлы «lcd.h» и «lcd.c», с помощью которых осуществляется работа с дисплеем.

В файле «main.c» первоначально были написаны блоки программы для взаимодействия с LCD дисплеем. Далее началась разработка обработки данных с АЦП и их проектирование на дисплей. На последнем этапе

разработки происходила оптимизация кода, его «чистка» и добавление комментариев.

Код программы находится в приложении Б.

2.2 Обобщённая структура программы

1. Включение содержимого файла «lcd.h».
2. Инициализация значений для обработки данных.
3. Определение функций.
4. Функция колбэка по прерыванию от АЦП.
5. Инициализация HAL, системного тактирования, GPIO, DMA и ADC1.
6. Инициализация LCD дисплея.
7. Запуск АЦП в двух каналах.
8. Преобразование значений АЦП в напряжение и вычисление падения напряжение.
9. Вывод на дисплей режима работы в зависимости от положения ротатора.
10. Вывод на дисплей диапазона измерений в зависимости от положения ротатора.
11. Вывод на дисплей результата измерений в зависимости от диапазона измерений.
12. Повторный запуск АЦП для получения новых значений
13. Повторение пунктов 8-12.

2.3 Описание отдельных алгоритмов

Алгоритм вывода измерения в диапазоне 10-100 В:

1. Проверяем напряжение на пине PA2, если оно есть – идёт дальше.
2. Если выбран режим «off», то экран будет очищен.
3. Иначе ставится курсор на позицию «0,11» и на дисплей выводится выбранный диапазон.

4. Если полученное напряжение выше 100, то выводится сообщение о достигнутом лимите режима.

4. Если полученное напряжение в диапазоне от 10 до 100 В, то ставится курсор в нужное положение, значение напряжения переводится из типа float в string (буфер для string уже инициализирован в начале программы), выводится значение на дисплей, переносится курсор для вывода обозначения и пустых символов до конца дисплея. Это делается, чтобы «подчищать» экран при смене режимов и диапазонов.

5. Если полученное напряжение меньше 10 В, то происходят те же шаги, что и в 4 пункте, только меняются позиции курсоров, обозначения и пустых символов.

Алгоритм функции Lcd_write_command:

1. Функция принимает два параметра. Lcd_HandleTypeDef * lcd: указатель на структуру, содержащую информацию о конфигурации LCD и uint8_t command: команда, которую необходимо отправить на дисплей.

2. Устанавливается состояние порта rs (Register Select) в режим команды. Это означает, что следующая отправляемая информация будет интерпретироваться как команда, а не данные для отображения.

3. Функция проверяет, установлен ли режим работы дисплея на 4-битный (LCD_4_BIT_MODE). Если да, то отправляется старшая nibble (4 бита) команды. Команда сдвигается вправо на 4 бита, чтобы выделить старшие 4 бита. Отправляется младшая nibble (4 бита) команды. Используется побитовая операция AND для извлечения младших 4 битов команды.

Если режим не 4-битный (т.е. 8-битный), то отправляется вся команда целиком как один байт.

3 Тестирование и отладка аппаратно-программного комплекса

Тестирование общей работоспособности устройства сопровождалось подключением источников напряжения к пинам микроконтроллера через делители напряжения. На рисунках 5 - 10 показаны различные результаты работы вольтметра при таких ситуациях, как: измерение напряжение и измерение падения напряжения в различных режимах отображения, оповещение о повышении режима отображения измерения, режим «off»

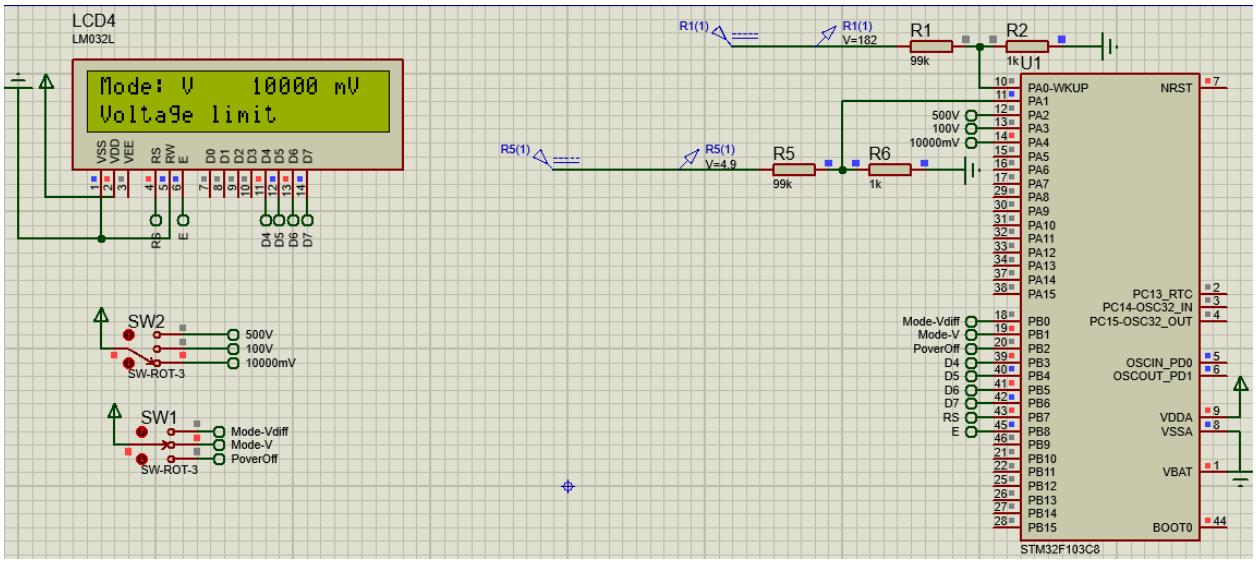

Рисунок 5 – Измерение 182 В в режиме «10000 мВ»

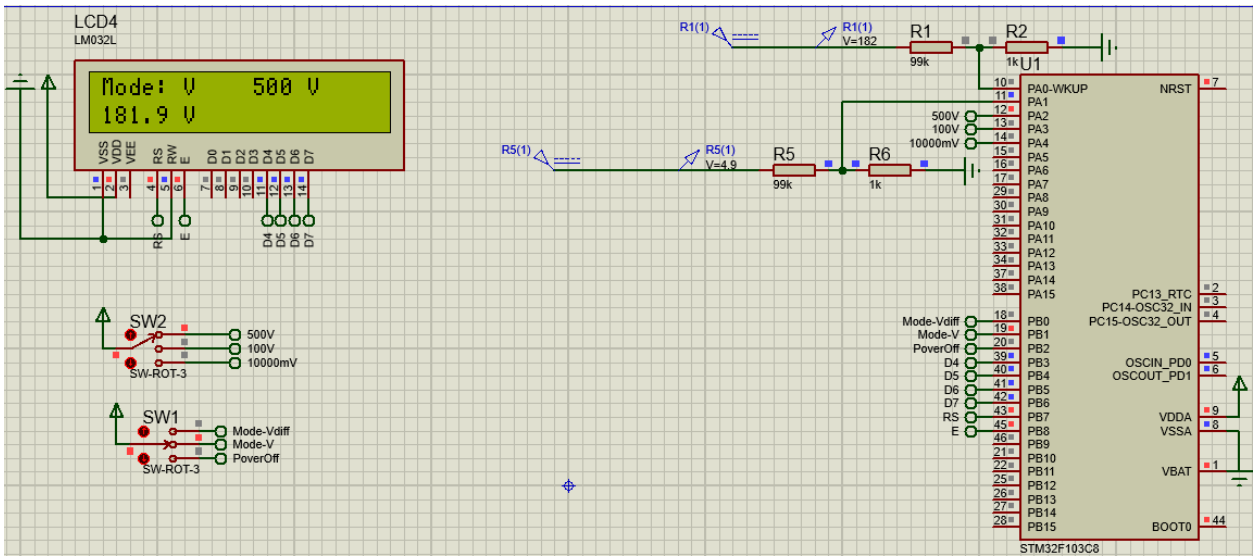

Рисунок 6 – Измерение 182 В в режиме «500 В»

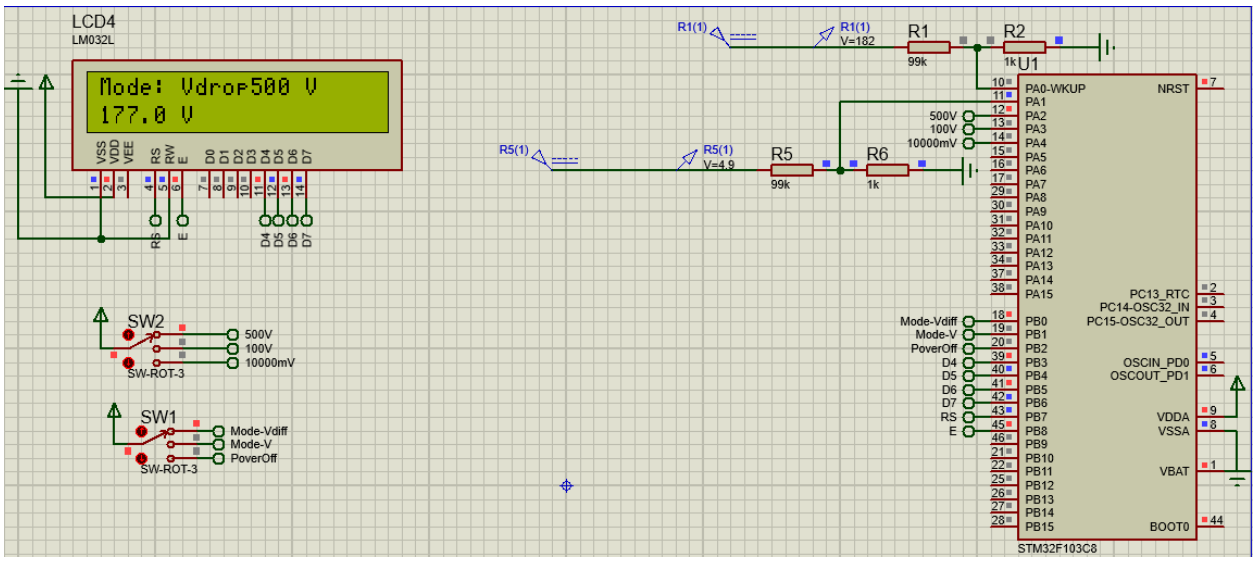

Рисунок 7 – Измерение падения напряжения в режиме «500 В»

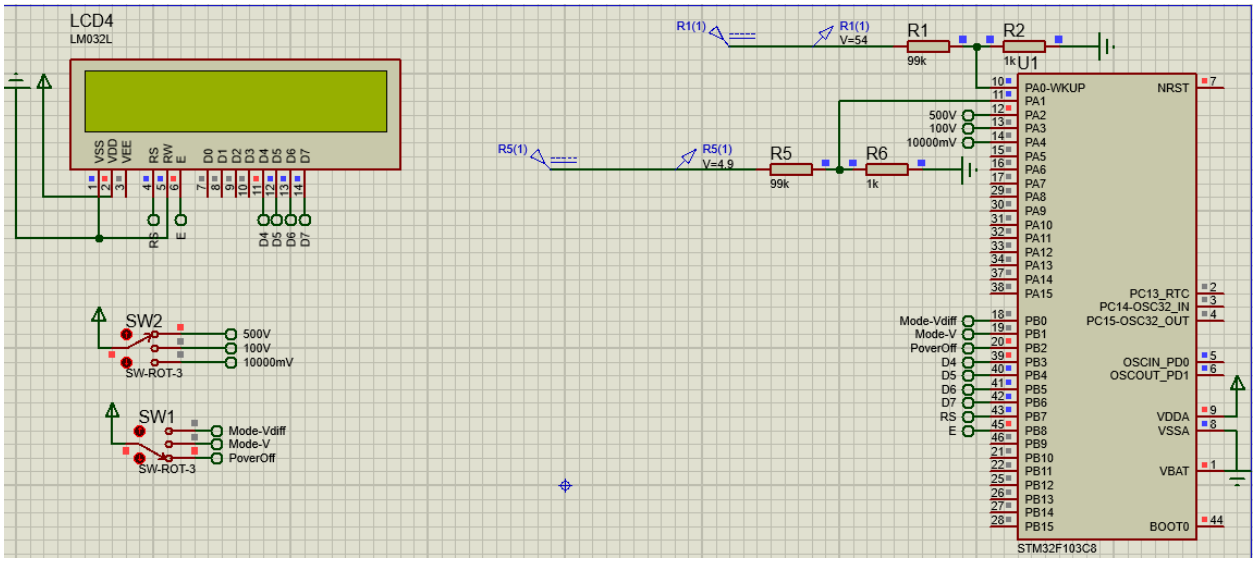

Рисунок 8 – Режим «Off»

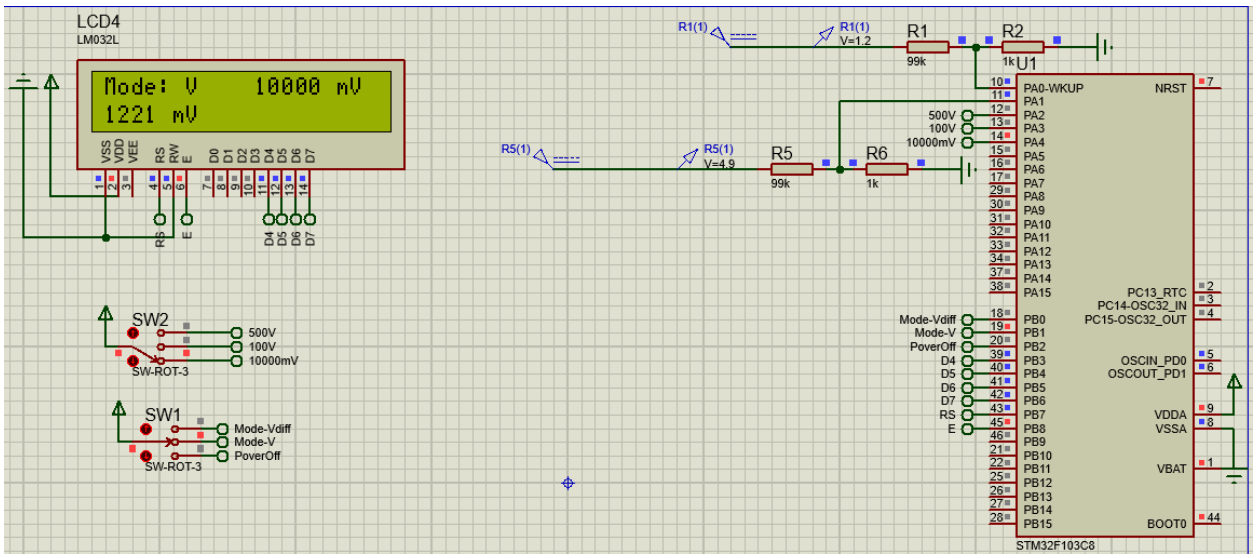

Рисунок 9 – Измерение 1.2 В в режиме «10000 мВ»

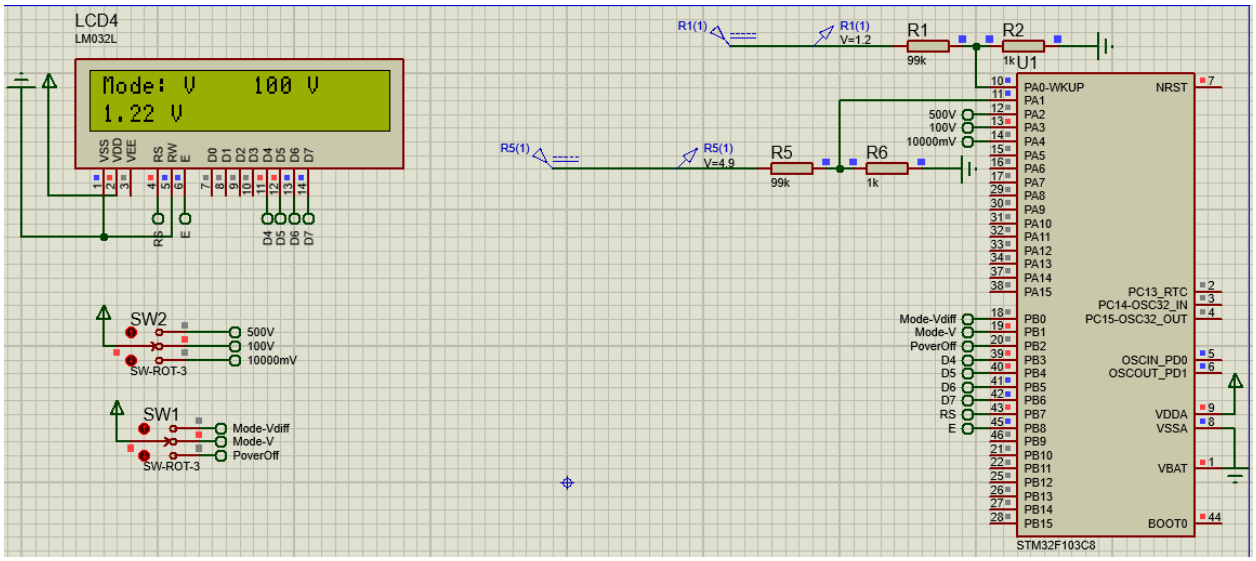

Рисунок 10 – Измерение 1.2 В в режиме «100 В»

ЗАКЛЮЧЕНИЕ

Результатом курсовой работы является разработанный аппаратно-программный комплекс цифрового вольтметра на базе микроконтроллера серии ARM, который представляет собой компактное устройство для точного измерения и отображения напряжения и его падения на участках цепи. Назначение данного устройства — обеспечение надежного мониторинга напряжения в режиме реального времени.

Устройство выполняет следующие функции:

Оцифровка данных: Аналого-цифровой преобразователь микроконтроллера преобразует аналоговый сигнал в цифровую форму, что делает возможной обработку данных микроконтроллером.

Измерение напряжения: микроконтроллер обрабатывает данные с АЦП, вычисляя напряжение, полученное на соответствующий пин.

Измерение падения напряжения: благодаря многоканальной работе АЦП, есть возможность получать значения на него с нескольких пинов, чтобы вычислять изменение напряжения на участках цепи.

Отображение результата: ЖК-дисплей LCD032L выводит измеренное значение напряжения в различных форматах, позволяя пользователю легко считывать показания в удобной форме.

Основные технические характеристики устройства:

- Диапазон измерения напряжения: от 0 В до 500 В.
- Точность измерений: ±122мВ.
- Режимы отображения напряжения: до 10000 мВ, до 100 В, до 500 В.
- Питание: для работы устройства достаточно напряжения 5V.
- Интерфейс отображения: LCD-дисплей с 2 строками по 20 символов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. «Вольтметры: виды и принцип работы», URL: https://www.souz-pribor.ru/articles/voltmetry-vidy-i-printsip-raboty/ (дата обращения 26.11.2024)

2. STM32F103C8T6 datasheet, URL: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf (дата обращения 26.11.2024)

3. LM032L datasheet, URL: https://www.qsl.net/la9sja/electronics/ea/1838001.pdf (дата обращения 26.11.2024)

ПРИЛОЖЕНИЕ А

Принципиальная схема проекта

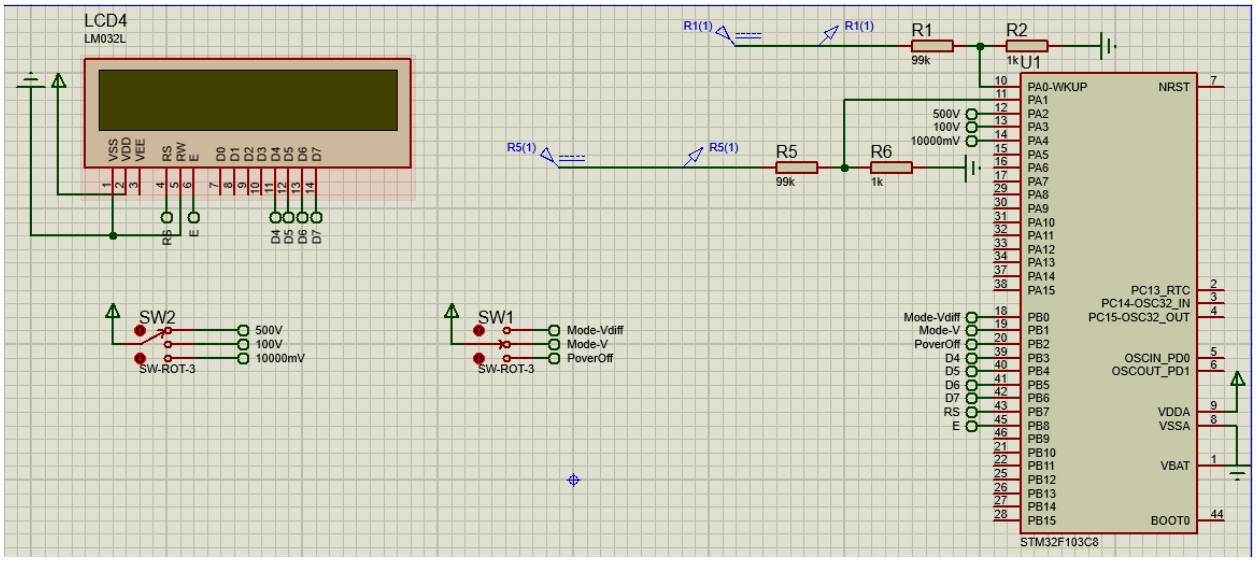

Рисунок 11 – Схема проекта

ПРИЛОЖЕНИЕ Б

Текст программы

Файл «main.c»

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "lcd.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

/* USER CODE BEGIN PV */
volatile uint16_t adc[2]={0,0}; // значения с АЦП с двух каналов
volatile float voltage[2]={0,0}; //расчитываемое напряжение
volatile uint8_t flag = 0; // флаг для колбэка
char buffer[12]; // буффер для числа
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
```

```c
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) // Колбэк по прерыванию от работы ADC
{
    if(hadc->Instance == ADC1) //проверка если прерывание от ADC1
    {
        flag = 1;
    }
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */
  Lcd_PortType ports[]={GPIOB,GPIOB,GPIOB,GPIOB};
  Lcd_PinType pins[]={GPIO_PIN_3,GPIO_PIN_4,GPIO_PIN_5,GPIO_PIN_6};
  Lcd_HandleTypeDef lcd;
  lcd=Lcd_create(ports,pins,GPIOB,GPIO_PIN_7,GPIOB,GPIO_PIN_8, LCD_4_BIT_MODE);
//инициализация ЛСД

  HAL_ADC_Start_DMA(&hadc1, (uint32_t*)&adc, 2); //запуск АЦП в двух каналах с помощью прямого доступа к памяти
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

        if(flag)
        {
              flag = 0; //
              voltage[0]=adc[0]/4096.0*500.0; //делим полученное измерение на кол-
во измерений (2^12) и умножаем на макс. значение вольтметра, получаем вольты
              voltage[1] = adc[1]/4096.0*500.0; // тоже самое для второго канала
(нужен только для режима измерения падения напряжения)


        if (HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0)==GPIO_PIN_SET){ //если выбран режим
измерения падения напряжения
              Lcd_cursor(&lcd,0,0);
              Lcd_string(&lcd, "Mode: Vdrop"); //вывод на lcd
              voltage[1]=voltage[1]-voltage[0]; // считаем разницу

        }
        if (HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1)==GPIO_PIN_SET){ // если выбран режим
измерения напряжения
              Lcd_cursor(&lcd,0,0);
              Lcd_string(&lcd, "Mode: V    ");//вывод на lcd

        }

        if (HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_2)==GPIO_PIN_SET){ // если выбран
диапазон 100-500В
              if (HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2)==GPIO_PIN_SET){ // проверка
если стоит режим выключенного прибора
                    Lcd_clear(&lcd); // значит держим экран чистым
              }
              else{
                    Lcd_cursor(&lcd,0,11);
                    Lcd_string(&lcd, "500 V   "); // вывод выбранного диапазона
                    if (voltage[1]>=499.9){ // если напряжение выше возможного
измерить
                          Lcd_string(&lcd, "Voltage limit"); // пишем, что
достигли лимита вольтметра
                    }
                    if (voltage[1]>10.0 & voltage[1]<100.0){ // если напряжение
ниже выбранного диапазона
                          Lcd_cursor(&lcd,1,0);
                          snprintf(buffer, sizeof(buffer), "%.1f", voltage[1]);
//из float в string
                          Lcd_string(&lcd, buffer); // выводим напряжение
                          Lcd_cursor(&lcd,1,4);
                          Lcd_string(&lcd, " V            "); // обозначение +
чистим экран от значов из других режимов
                    }
                    if (voltage[1]<10.0){ // если напряжение ниже выбранного
диапазона
                          Lcd_cursor(&lcd,1,0);
                          snprintf(buffer, sizeof(buffer), "%.1f", voltage[1]);
//из float в string
                          Lcd_string(&lcd, buffer); // выводим напряжение
                          Lcd_cursor(&lcd,1,3);
                          Lcd_string(&lcd, " V             "); // обозначение +
чистим экран от значов из других режимов
```

```
                        }
                        if (voltage[1]>=100.0){
                                snprintf(buffer, sizeof(buffer), "%.1f", voltage[1]);
//из float в string
                                Lcd_cursor(&lcd,1,0);
                                Lcd_string(&lcd, buffer); // выводим напряжение
                                Lcd_cursor(&lcd,1,5);
                                Lcd_string(&lcd, " V            "); // обозначение +
чистим экран от значов из других режимов
                        }
                }
        }
        if (HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_3)==GPIO_PIN_SET){ // если выбран
диапазон 10-100В
                if (HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2)==GPIO_PIN_SET){ // проверка
если стоит режим выключенного прибора
                        Lcd_clear(&lcd); // значит держим экран чистым
                }
                else{
                        Lcd_cursor(&lcd,0,11);
                        Lcd_string(&lcd, "100 V   "); // вывод выбранного диапазона
                        if (voltage[1]>=100.0){ // если напряжение выше выбранного
диапазона
                                Lcd_cursor(&lcd,1,0);
                                Lcd_string(&lcd, "Voltage limit"); // пишем, что
достигли лимита диапазона
                        }
                        if (voltage[1]<10.0){ // если напряжение ниже выбранного
диапазона
                                Lcd_cursor(&lcd,1,0);
                                snprintf(buffer, sizeof(buffer), "%.2f", voltage[1]);
//из float в string
                                Lcd_cursor(&lcd,1,0);
                                Lcd_string(&lcd, buffer); // выводим напряжение
                                Lcd_cursor(&lcd,1,4);
                                Lcd_string(&lcd, " V             "); // обозначение +
чистим экран от значов из других режимов
                        }
                        if (voltage[1]>10.0 & voltage[1]<100.0){ // если напряжение в
нашем выбранном диапазоне
                                snprintf(buffer, sizeof(buffer), "%.2f", voltage[1]);
//из float в string
                                Lcd_cursor(&lcd,1,0);
                                Lcd_string(&lcd, buffer); // выводим напряжение
                                Lcd_cursor(&lcd,1,5);
                                Lcd_string(&lcd, " V            "); // обозначение +
чистим экран от значов из других режимов
                        }
                }
        }
        if (HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_4)==GPIO_PIN_SET){ // если выбран
диапазон 0-10В
                if (HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2)==GPIO_PIN_SET){ // проверка
если стоит режим выключенного прибора
                        Lcd_clear(&lcd); // значит держим экран чистым
                }
                else{
                        Lcd_cursor(&lcd,0,11);
                        Lcd_string(&lcd, "10000 mV"); // вывод выбранного диапазона
                        if (voltage[1]>=10.0){ // если напряжение выше выбранного
диапазона
```

```
                        Lcd_cursor(&lcd,1,0);
                        Lcd_string(&lcd, "Voltage limit"); // пишем, что
достигли лимита диапазона
                    }
                    else{
                        voltage[1]=voltage[1]*1000.0; //переводим в мВ
                        snprintf(buffer, sizeof(buffer), "%4.0f", voltage[1]);
//из float в string
                        Lcd_cursor(&lcd,1,0);
                        Lcd_string(&lcd, buffer);// выводим напряжение
                        Lcd_cursor(&lcd,1,4);
                        Lcd_string(&lcd, " mV          "); // обозначение +
чистим экран от значов из других режимов
                    }
                }
        }
        HAL_ADC_Start_DMA(&hadc1, (uint32_t*)&adc, 2); //по новой измеряем значение
АЦП
        }

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
 }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 2;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = ADC_REGULAR_RANK_2;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */
```

```
  /* USER CODE END ADC1_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6
                          |GPIO_PIN_7|GPIO_PIN_8, GPIO_PIN_RESET);

  /*Configure GPIO pins : PA2 PA3 PA4 PA5
                           PA6 */
  GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                          |GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : PB3 PB4 PB5 PB6
                           PB7 PB8 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6
                          |GPIO_PIN_7|GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
```

```
/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Файл «lcd.h»

```
#ifndef LCD_H_
#define LCD_H_

#include "stm32f1xx_hal.h"
#include "string.h"
#include "stdio.h"
#include "main.h"

//Chose which LCD you are using
#define LCD20xN           // For 20xN LCDs

// For row start addresses
extern const uint8_t ROW_16[];
extern const uint8_t ROW_20[];

/************************************ Command register
************************************/
#define CLEAR_DISPLAY 0x01

#define RETURN_HOME 0x02

#define ENTRY_MODE_SET 0x04
```

```
#define OPT_S0x01                               // Shift entire display to right
#define OPT_INC 0x02                                  // Cursor increment

#define DISPLAY_ON_OFF_CONTROL 0x08
#define OPT_D0x04                               // Turn on display
#define OPT_C0x02                               // Turn on cursor
#define OPT_B       0x01                              // Turn on cursor blink

#define CURSOR_DISPLAY_SHIFT 0x10                        // Move and shift cursor
#define OPT_SC 0x08
#define OPT_RL 0x04

#define FUNCTION_SET 0x20
#define OPT_DL 0x10                              // Set interface data length
#define OPT_N 0x08                               // Set number of display lines
#define OPT_F 0x04                               // Set alternate font
#define SETCGRAM_ADDR 0x040
#define SET_DDRAM_ADDR 0x80                             // Set DDRAM address


/************************************ Helper macros
************************************/
#define DELAY(X) HAL_Delay(X)

/************************************ LCD defines
************************************/
#define LCD_NIB 4
#define LCD_BYTE 8
#define LCD_DATA_REG 1
#define LCD_COMMAND_REG 0

/************************************ LCD typedefs
************************************/
#define Lcd_PortType GPIO_TypeDef*
#define Lcd_PinType uint16_t

typedef enum {
      LCD_4_BIT_MODE,
      LCD_8_BIT_MODE
} Lcd_ModeTypeDef;


typedef struct {
      Lcd_PortType * data_port;
      Lcd_PinType * data_pin;

      Lcd_PortType rs_port;
      Lcd_PinType rs_pin;

      Lcd_PortType en_port;
      Lcd_PinType en_pin;

      Lcd_ModeTypeDef mode;

} Lcd_HandleTypeDef;

/************************************ Public functions Prototypes
************************************/
void Lcd_init(Lcd_HandleTypeDef * lcd);
void Lcd_int(Lcd_HandleTypeDef * lcd, int number);
void Lcd_string(Lcd_HandleTypeDef * lcd, char * string);
```

```
void Lcd_cursor(Lcd_HandleTypeDef * lcd, uint8_t row, uint8_t col);
Lcd_HandleTypeDef Lcd_create(
             Lcd_PortType port[], Lcd_PinType pin[],
             Lcd_PortType rs_port, Lcd_PinType rs_pin,
             Lcd_PortType en_port, Lcd_PinType en_pin, Lcd_ModeTypeDef mode);
void Lcd_define_char(Lcd_HandleTypeDef * lcd, uint8_t code, uint8_t bitmap[]);
void Lcd_clear(Lcd_HandleTypeDef * lcd);

#endif /* LCD_H_ */
```

Файл «lcd.c»

```
#include <lcd.h>
const uint8_t ROW_16[] = {0x00, 0x40, 0x10, 0x50};
const uint8_t ROW_20[] = {0x00, 0x40, 0x14, 0x54};
/************************************ Static declarations
************************************/

static void lcd_write_data(Lcd_HandleTypeDef * lcd, uint8_t data);
static void lcd_write_command(Lcd_HandleTypeDef * lcd, uint8_t command);
static void lcd_write(Lcd_HandleTypeDef * lcd, uint8_t data, uint8_t len);


/************************************ Function definitions
************************************/


/**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 * Create new Lcd_HandleTypeDef and initialize the Lcd pins
 *++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
Lcd_HandleTypeDef Lcd_create(
             Lcd_PortType port[], Lcd_PinType pin[],
             Lcd_PortType rs_port, Lcd_PinType rs_pin,
             Lcd_PortType en_port, Lcd_PinType en_pin, Lcd_ModeTypeDef mode)
{
      Lcd_HandleTypeDef lcd;

      lcd.mode = mode;

      lcd.en_pin = en_pin;
      lcd.en_port = en_port;

      lcd.rs_pin = rs_pin;
      lcd.rs_port = rs_port;

      lcd.data_pin = pin;
      lcd.data_port = port;

      Lcd_init(&lcd);

      return lcd;
}


/**++++++++++++++++++++++++++++++++++++
 * Initialize 20x4-lcd without cursor
 *++++++++++++++++++++++++++++++++++++*/
void Lcd_init(Lcd_HandleTypeDef * lcd)
{
      if(lcd->mode == LCD_4_BIT_MODE)
```

```
	{
			lcd_write_command(lcd, 0x33);
			lcd_write_command(lcd, 0x32);
			lcd_write_command(lcd, FUNCTION_SET | OPT_N); // Set LCD in 4-bit
mode
	}
	else
		lcd_write_command(lcd, FUNCTION_SET | OPT_DL | OPT_N);


	lcd_write_command(lcd, CLEAR_DISPLAY);			// Clear screen
	lcd_write_command(lcd, DISPLAY_ON_OFF_CONTROL | OPT_D);		// Lcd-on,
cursor-off, no-blink
	lcd_write_command(lcd, ENTRY_MODE_SET | OPT_INC);		// Increment cursor
}


/**+++++++++++++++++++++++++++++++++++++++++++
 * Write a number on the current position
 **+++++++++++++++++++++++++++++++++++++++++++*/
void Lcd_int(Lcd_HandleTypeDef * lcd, int number)
{
	char buffer[11];
	sprintf(buffer, "%d", number);

	Lcd_string(lcd, buffer);
}

/**+++++++++++++++++++++++++++++++++++++++++
 * Write a string on the current position
 *+++++++++++++++++++++++++++++++++++++++++*/
void Lcd_string(Lcd_HandleTypeDef * lcd, char * string)
{
	for(uint8_t i = 0; i < strlen(string); i++)
	{
		lcd_write_data(lcd, string[i]);
	}
}

/**+++++++++++++++++++++++++++++++++++++++++++*
 * Set the cursor position
 *+++++++++++++++++++++++++++++++++++++++++++*/
void Lcd_cursor(Lcd_HandleTypeDef * lcd, uint8_t row, uint8_t col)
{
	#ifdef LCD20xN
	lcd_write_command(lcd, SET_DDRAM_ADDR + ROW_20[row] + col);
	#endif

	#ifdef LCD16xN
	lcd_write_command(lcd, SET_DDRAM_ADDR + ROW_16[row] + col);
	#endif
}

/**++++++++++++++++++*
 * Clear the screen
 *++++++++++++++++++*/
void Lcd_clear(Lcd_HandleTypeDef * lcd) {
	lcd_write_command(lcd, CLEAR_DISPLAY);
}

void Lcd_define_char(Lcd_HandleTypeDef * lcd, uint8_t code, uint8_t bitmap[]){
```

```
	lcd_write_command(lcd, SETCGRAM_ADDR + (code << 3));
	for(uint8_t i=0;i<8;++i){
		lcd_write_data(lcd, bitmap[i]);
	}

}

/**++++++++++++++++++++++++++++++++++++++*
 * Write a byte to the command register
 *++++++++++++++++++++++++++++++++++++++*/
void lcd_write_command(Lcd_HandleTypeDef * lcd, uint8_t command)
{
	HAL_GPIO_WritePin(lcd->rs_port, lcd->rs_pin, LCD_COMMAND_REG);	// Write to
command register

	if(lcd->mode == LCD_4_BIT_MODE)
	{
		lcd_write(lcd, (command >> 4), LCD_NIB);
		lcd_write(lcd, command & 0x0F, LCD_NIB);
	}
	else
	{
		lcd_write(lcd, command, LCD_BYTE);
	}

}

/**+++++++++++++++++++++++++++++++++++*
 * Write a byte to the data register
 *+++++++++++++++++++++++++++++++++++*/
void lcd_write_data(Lcd_HandleTypeDef * lcd, uint8_t data)
{
	HAL_GPIO_WritePin(lcd->rs_port, lcd->rs_pin, LCD_DATA_REG);  // Write to data
register

	if(lcd->mode == LCD_4_BIT_MODE)
	{
		lcd_write(lcd, data >> 4, LCD_NIB);
		lcd_write(lcd, data & 0x0F, LCD_NIB);
	}
	else
	{
		lcd_write(lcd, data, LCD_BYTE);
	}

}

/**+++++++++++++++++++++++++++++++++++++++++++++++++++++*
 * Set len bits on the bus and toggle the enable line
 *+++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void lcd_write(Lcd_HandleTypeDef * lcd, uint8_t data, uint8_t len)
{
	for(uint8_t i = 0; i < len; i++)
	{
		HAL_GPIO_WritePin(lcd->data_port[i], lcd->data_pin[i], (data >> i) &
0x01);
	}

	HAL_GPIO_WritePin(lcd->en_port, lcd->en_pin, 1);
	DELAY(1);
```

```
	HAL_GPIO_WritePin(lcd->en_port, lcd->en_pin, 0);	// Data receive on
falling edge
}


//END
```