МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(Самарский университет)

Институт информатики и кибернетики

Кафедра лазерных и биотехнических систем

Пояснительная записка к курсовому проекту

’ИЗМЕРИТЕЛЬ ПОСТОЯННОГО ТОКА ”

Выполнил студент группы 6364-120304D: _____ Рожновская Д.О.

Руководитель проекта: _____ Корнилин Д.В.

Работа защищена с оценкой: _____

Самара 2023

ЗАДАНИЕ

Разработать измеритель постоянного тока со следующими параметрами:

– Диапазон измеряемых токов: 1мкА - 0.1А;

– Максимальная погрешность: 0.5%;

– Индикация: цифровая с необходимым количеством разрядов;

– Передача данных: по интерфейсу CAN с фиксированной скоростью.

РЕФЕРАТ

Пояснительная записка: 23 страниц, 8 рисунков, 3 источника, 1 приложение.

ИЗМЕРИТЕЛЬ ПОСТОЯННОГО ТОКА, МИКРОКОНТРОЛЛЕР, CAN, STM32, АЛГОРИТМ, ПРОГРАММА, ОПЕРАЦИОННЫЙ УСИЛИТЕЛЬ, ИНСТРУМЕНТАЛЬНЫЙ УСИЛИТЕЛЬ, ЦИФРОВАЯ ИНДИКАЦИЯ

В курсовом проекте разработаны структурная и принципиальная схемы измерителя постоянного тока, осуществлен выбор микроконтроллера с шиной CAN, подобраны элементы для блока питания и датчика тока. В качестве индикатора выбран IPS дисплей. Разработан алгоритм анализа данных и программа на языке Си, реализующая его.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработка измерителя постоянного тока является важной задачей в области электротехники и электроники. Такой прибор необходим для точного измерения постоянного тока в различных электрических цепях и системах.

Измерители постоянного тока используются в различных областях, включая промышленность, автомобильную отрасль, энергетику и телекоммуникации. Они помогают обеспечить безопасность и надежность работы систем, а также повышают эффективность использования электроэнергии. Важность измерителей постоянного тока заключается в том, что они позволяют контролировать и оптимизировать работу систем, что в свою очередь повышает качество продукции и уменьшает затраты на производство.

В данном курсовом проекте рассматривается способ создания устройства на базе микроконтроллера, который сможет обеспечить высокую скорость передачи данных, что позволит быстро и точно измерять ток. В процессе был подобран необходимый в задании микроконтроллер с шиной CAN, а также написана управляющая программа на языке Си.

1 РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ УСТРОЙСТВА

Структурная схема устройства представлена на рисунке 1.

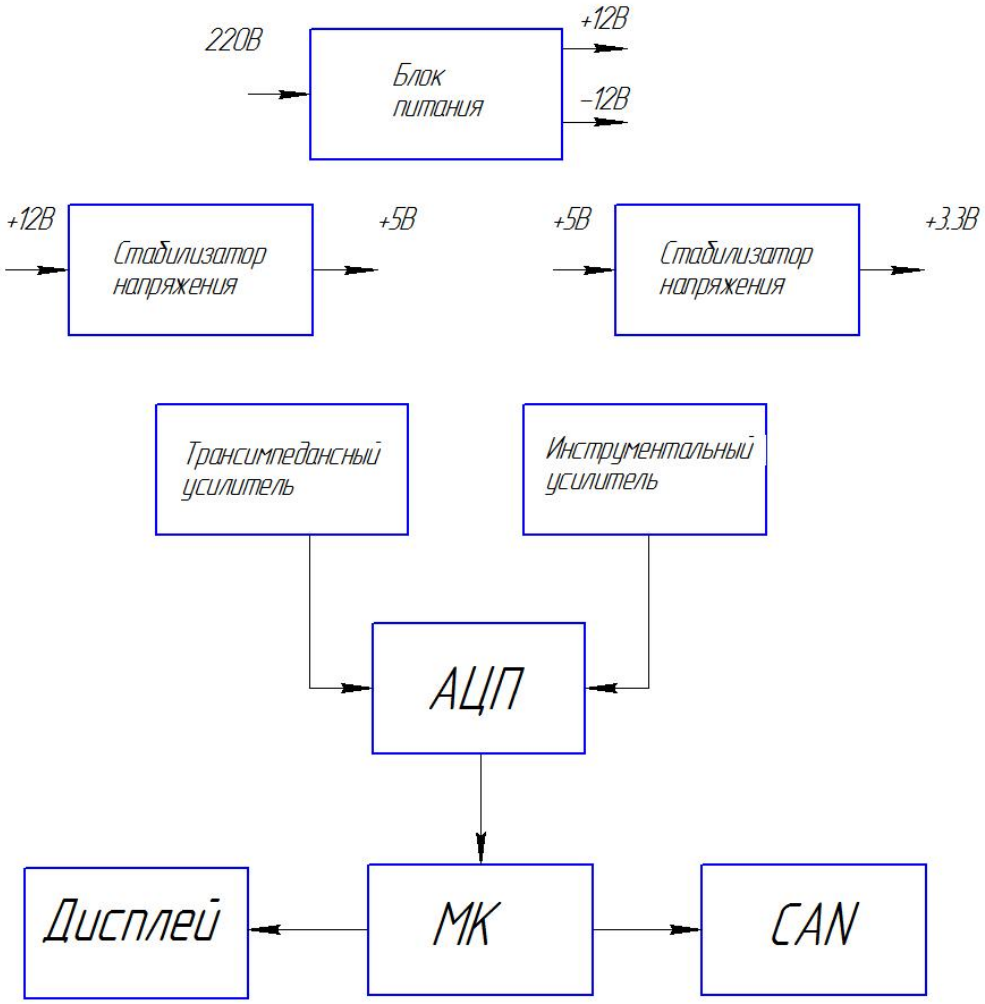

Рисунок 1 – Структурная схема устройства

Стоит отметить, что измерять ток цифровые устройства не умеют, поэтому, ток преобразуют в напряжение, чтобы АЦП мог оцифровать его. Принцип работы устройства заключается в следующем. АЦП имеет два канала. На один канал подключен выход инструментального уселителя, усиливающего напряжение на низкоомном шунте. Данный канал используется для измерения токов в диапазоне 1мА-100мА. Для измерения токов в диапазоне 1мкА-1мА используется схема трансимпедансного усилителя [4], изображенная на рисунке 2.

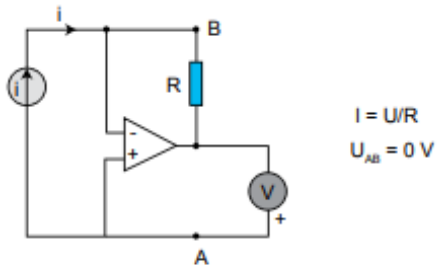

Figure 3—The transimpedance amplifier is a way to automatically adjust the counter voltage. An operational amplifier will set its output in order to have a nearly null voltage offset between its two inputs: U_{AB} = 0, which is exactly what we are looking for.

Рисунок 2 – Трансимпедансный усилитель

Переключение между каналами осуществляется программно. Измеренное значение напряжения пересчитывается в ток, и выводится на IPS дисплей. Так же, результаты могут быть переданы по интерфейсу CAN с фиксированной скоростью.

Блок питания формирует напряжение +12В и -12В из 220В для питания операционных усилителей. Посредством использования стабилизаторов напряжения из 12В получаем напряжения в 3.3В и 5В, необходимые для питания микроконтроллера и других элементов схемы.

2 РАЗРАБОТКА ПРИНЦИПИАЛЬНОЙ СХЕМЫ УСТРОЙСТВА

Электрическая принципиальная схема представлена в приложении.

2.1 Разработка аналоговой части

Основным требованием для ОУ в схеме трансимпендасного усилителя является маые входные токи - они должны быть меньше, чем минимальная разрешенная погрешность измерения. Данному требованию удовлетворяет AD8603 от Analog Devices. Его основные особенности представлены на рисунке 3.

FEATURES

Easy to use
- **Gain set with one external resistor (Gain range 1 to 10,000)**
- **Wide power supply range (±2.3 V to ±18 V)**
- **Higher performance than 3 op amp IA designs**
- **Available in 8-lead DIP and SOIC packaging**
- **Low power, 1.3 mA max supply current**

Excellent dc performance (B grade)
- **50 µV max, input offset voltage**
- **0.6 µV/°C max, input offset drift**
- **1.0 nA max, input bias current**
- **100 dB min common-mode rejection ratio (G = 10)**

Low noise
- **9 nV/√Hz @ 1 kHz, input voltage noise**
- **0.28 µV p-p noise (0.1 Hz to 10 Hz)**

Excellent ac specifications
- **120 kHz bandwidth (G = 100)**
- **15 µs settling time to 0.01%**

Рисунок 3 – Особенности AD8603

2.2 Выбор микроконтроллера

Серия stm32 f1 Ядро arm cortex-m3 Ширина шины данных 32-бит Тактовая частота 24 мгц Количество входов/выходов 37 Объем памяти программ 64 кбайт(64k x 8) Тип памяти программ flash Объем RAM 8k x 8 Наличие АЦП/ЦАП ацп 10x12b/цап 2x12b Встроенные интерфейсы i2c, irda, lin, spi, uart Встроенная периферия dma, pdr, por, pvd, pwm, tempsensor, wdt Напряжение

питания 2…3.6 в Рабочая температура -40…+85с Корпус lqfp-48(7x7) Вес, г 1.4

2.3 Выбор трансивера для CAN

MCP2551-I/SN [3]является высокоскоростным приемопередатчиком CAN, стойким к ошибкам устройством, которое служит в качестве интерфейса между контроллером CAN протокола и физической шиной. MCP2551-I/SN создает возможности дифференциальной передачи и приема для CAN контроллера и полностью совместим со стандартом ISO-11898. Приемопередатчик рекомендован для использования в системах с напряжением питания 12 В и 24 В. - Скорость передачи данных до 1Мбит/с - Управление выборкой данных для уменьшения влияния электромагнитных помех - Сброс по включению питания и снижению напряжения питания - Сброс MCP2551-I/SN не влияет на текущий обмен данными на шине CAN - Низкое энергопотребление в режиме ожидания - Защита от электрических импульсов - Защита от кратковременного подключения к цепям питания системы - Подключение до 112 узлов на одну CAN шину

2.4 Выбор индикатора

В качестве индикатора был выбран Дисплей TFT IPS 80x160 0.96”SPI RGB Полноцветный дисплей на IPS матрице с контроллером ST7735S

Технические характеристики: Разрешение: 80x160

Количество цветов: 65000

Угол обзора: > 160°

Напряжение питания: 3.3 В

Потребление: 0.04 ватта

Драйвер: ST7735S

Интерфейс: SPI

Размер дисплея: 21.7 x 10.8 мм.

Габаритные размеры: 30 x 24 x 4.1 мм

3 РАЗРАБОТКА ПРОГРАММЫ

3.1 Разработка алгоритма

Проанализируем задание, учитывая ранее описанное. Необходимо получать данные от АЦП ADS1115 по интерфейсу I2C. Так же данные передаются по интерфейсу CAN.

Для работы программы необходимо для начала разработать алгоритм. Алгоритм нашего устройства представлен на рисунке 4.

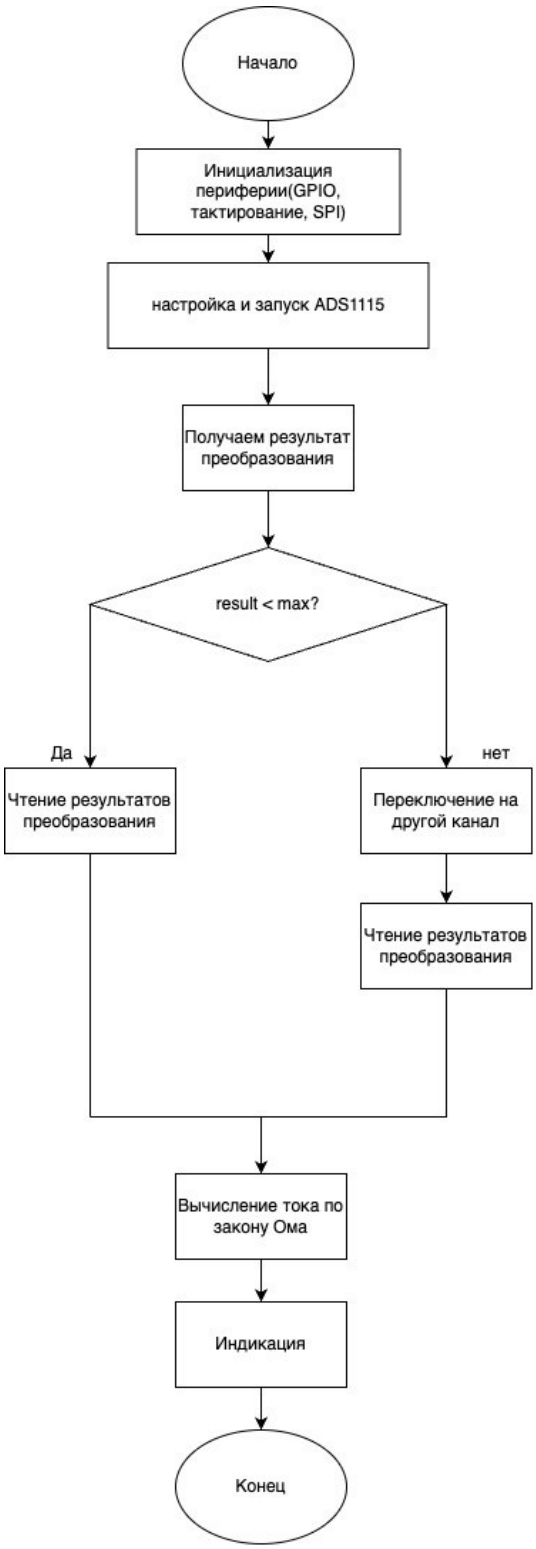

Рисунок 4 – Алгоритм работы устройства

3.2 Разработка кода

3.2.1 Выбор программного обеспечения

Для разработки ПО под STM32 можно использовать различные IDE.

Какие плюсы у данного ПО: абсолютно бесплатно, нет ограничения по размеру кода, есть неплохой отладчик, простая установка и настройка. Так же, стоит отметить, что данная платформа кроссплатформенная - есть версии для Windows, Linux и даже MacOS. Ознакомиться с STM32CubeIDE можно в [?]

3.2.2 Инициализация периферии

В STM32CubeIDE встроен STM32CubeMx – программный продукт, позволяющий при помощи достаточно понятного графического интерфейса произвести настройку любой имеющейся на борту микроконтроллера периферии. Подробнее об этом можно прочитать в [?]

Сначала в настройках Reset and Clock Controller(RCC) подключаем кварцевые резонаторы, как показано на рисунке 5.

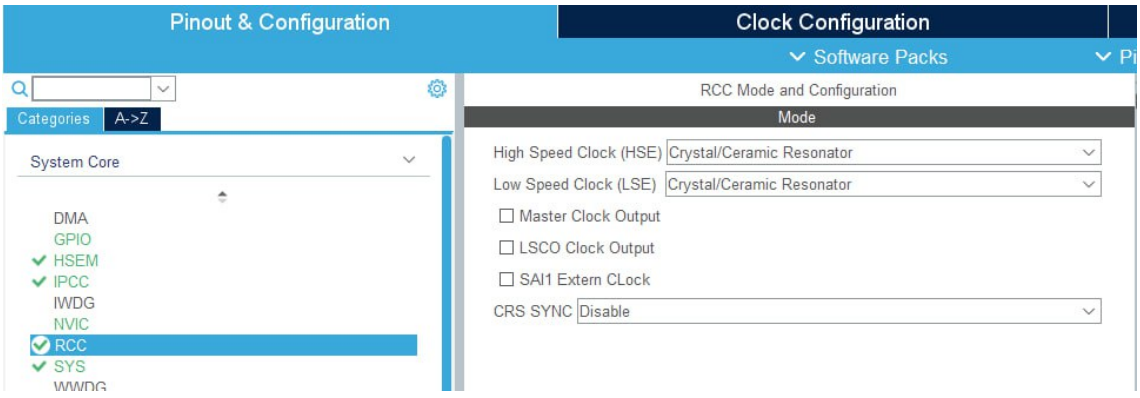

Рисунок 5 – Настройки RCC

Затем подключим порты ввода-вывода и настроим их как внешний источник прерываний, как показано на рисунке 6.

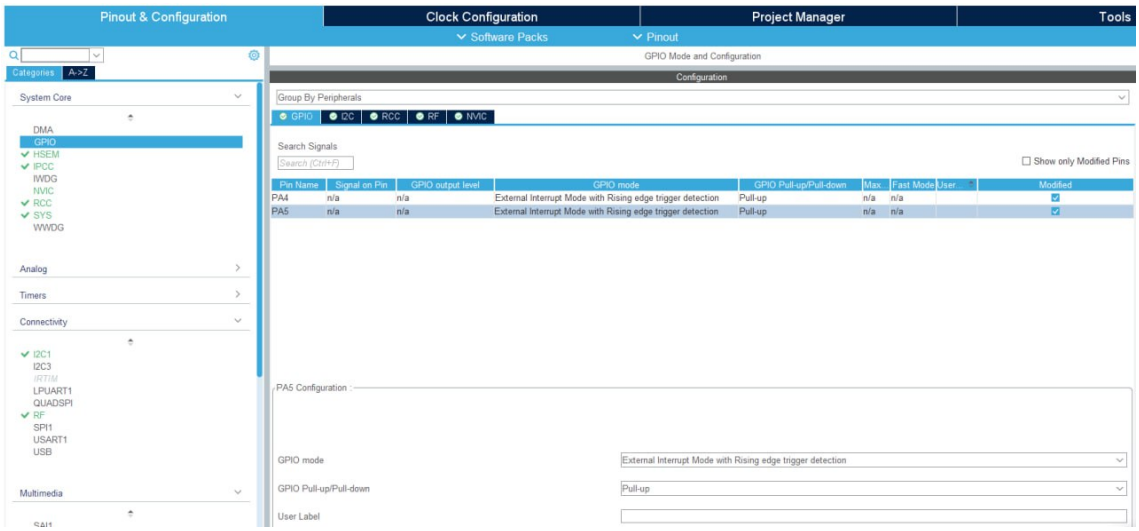

Рисунок 6 – Настройки портов ввода-вывода

Затем подключим интерфейс CAN, как показано на рисунке 7.

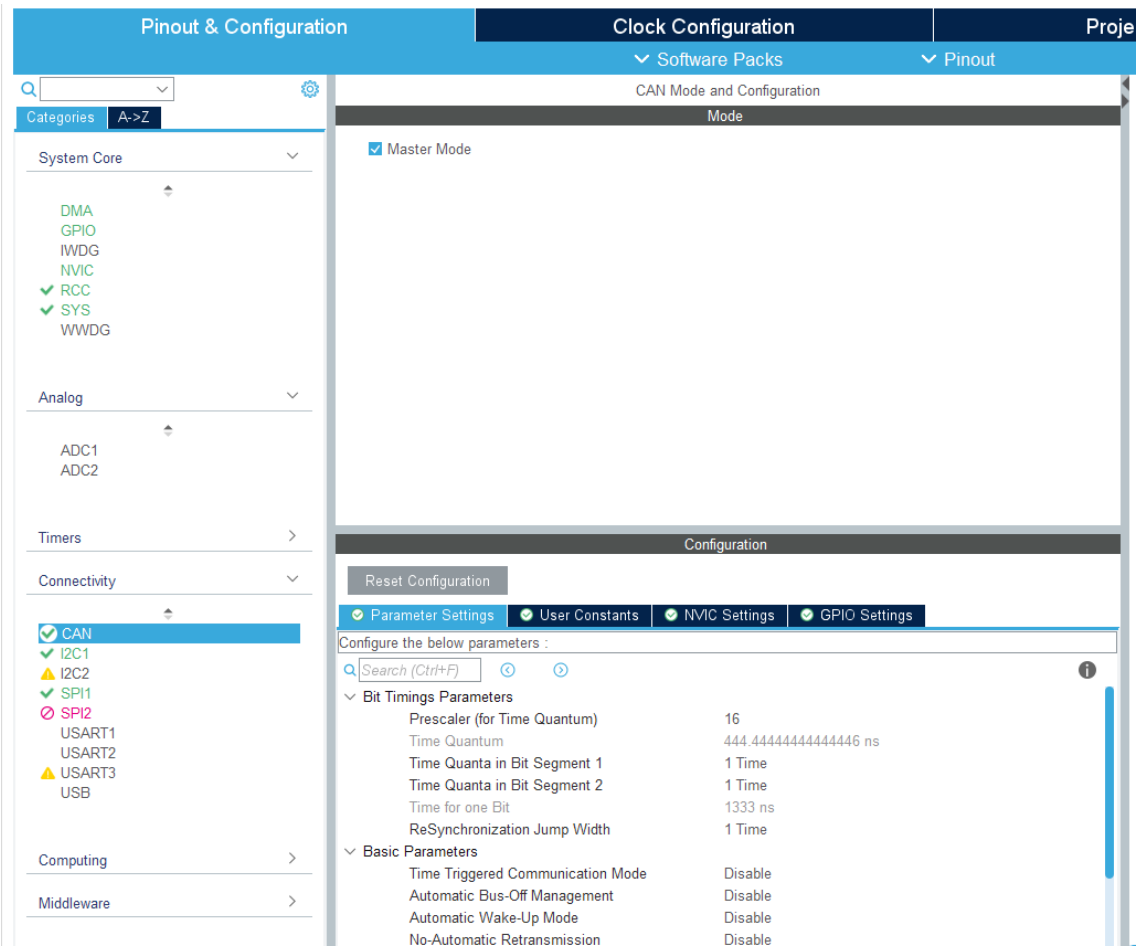

Рисунок 7 – Настройки портов ввода-вывода

После этого можно настроить тактирование на вкладке Clock Configuration, как показано на рисунке 8.

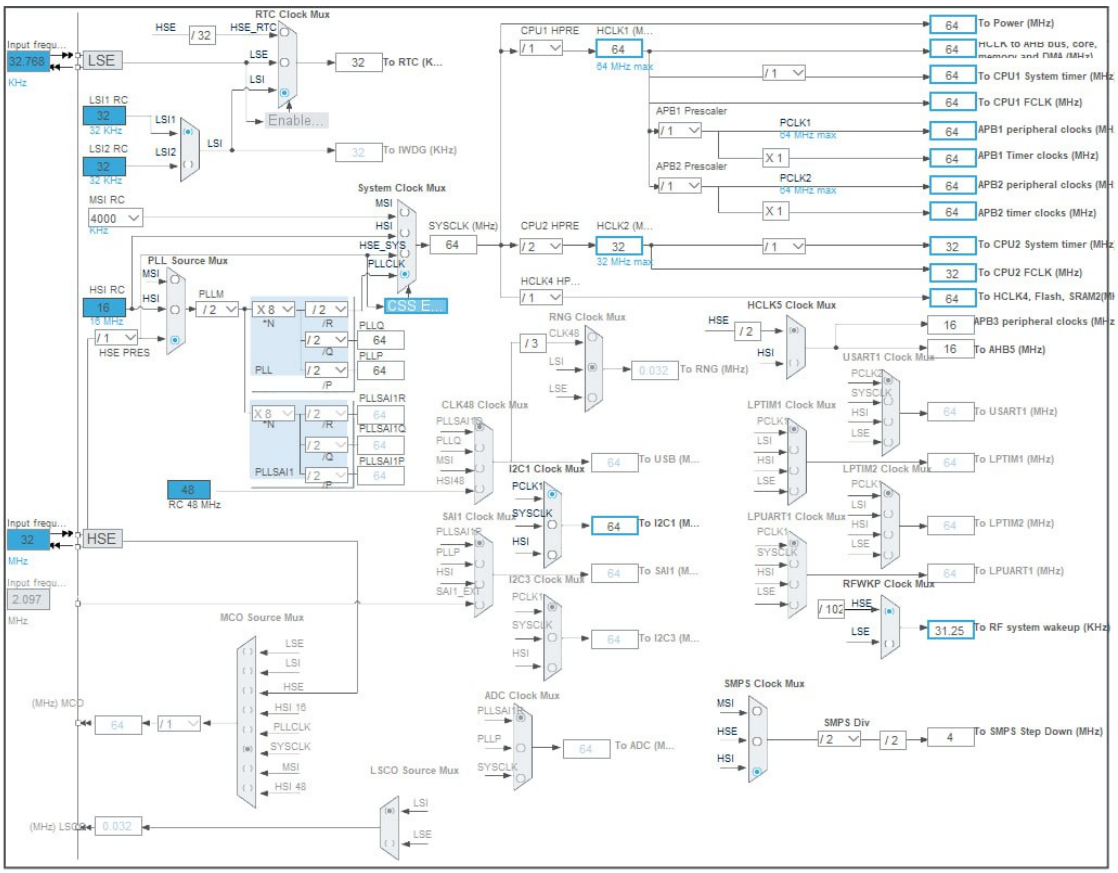

Рисунок 8 – Настройки тактирования

ЗАКЛЮЧЕНИЕ

В данном курсовом проекте рассмотрены принципы разработки устройств на базе микроконтроллеров. Был разработан измеритель постоянного тока. Данные передаются по интерфейсу CAN и выводится на экран. В процессе работы были разработаны структурная и принципиальная схемы устройства, были проведены необходимые расчёты для получения заданной погрешности, осуществлен выбор микроконтроллера и вспомогательных компонентов схемы.

Конфигуратор кода STM32CubeIDE предоставляет все необходимые библиотеки для реализации устройства, а также обеспечивает необходимые настройки микроконтроллера перед началом реализации алгоритма основной программы. Далее был разработан алгоритм программы и текст программы на языке Си.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Data Sheet на акселерометр ADS1115 [Электронный ресурс]. URL:https://cdn-shop.adafruit.com/datasheets/ads1115.pdf (Дата обращения: 15.05.2023)

2 Data Sheet на микроконтроллер STM32F103C8T6 [Электронный ресурс]. URL:http://inverter48.ru/datasheet/mcu/STM32F103C8T6.pdf (Дата обращения: 16.05.2023)

3 Data Sheet на трансивер MCP2551 [Электронный ресурс]. URL:https://static.chipdip.ru/lib/993/DOC012993895.pdf (Дата обращения: 16.05.2023)

4 Robert L., Picoammeter Design[Текст]/Robert Lacoste//CIRCUIT CELLAR –2010. –№237 – С. 62-66.

ПРИЛОЖЕНИЕ А

Основная программа

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "st7735.h"
#include "fonts.h"
#include "ADS1115.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
ADS1115_Config_t configReg; //структура, в которой храним настройки ADS1115
ADS1115_Handle_t *pADS; //указатель на структуру, в которой хранится информация о шине, и адресе ADS1115
/* 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 ---------------------------------------------------------*/
CAN_HandleTypeDef hcan;

I2C_HandleTypeDef hi2c1;

SPI_HandleTypeDef hspi1;
DMA_HandleTypeDef hdma_spi1_tx;

/* USER CODE BEGIN PV */
```

```
uint8_t r = 0; //задает ориентацию экрана(потом уберем)

uint16_t result;//храним результат преобразования

float voltage; //переменная для пересчитанного кода АЦП в вольты
char Buffer1[100];

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_SPI1_Init(void);
static void MX_CAN_Init(void);
static void MX_I2C1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void CAN_Data_Send(uint8_t Tx_Data){

	CAN_TxHeaderTypeDef Tx_Header;

	Tx_Header.StdId = 0x201;
	Tx_Header.IDE = CAN_ID_STD;
	Tx_Header.RTR = CAN_RTR_DATA;
	Tx_Header.DLC = 0x08;
	HAL_CAN_AddTxMessage(&hcan, &Tx_Header, Tx_Data, (uint32_t *)CAN_TX_MAILBOX0);
}

/* 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_SPI1_Init();
  MX_CAN_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */
  ST7735_Init();
  ST7735_Backlight_On();

  pADS = ADS1115_init(&hi2c1, ADS1115_ADDRESS, configReg);//инициализируем ADS
  ADS1115_SetDefault(pADS);
      ST7735_SetRotation(1);
      ST7735_FillScreen(ST7735_BLACK);

      voltage=10.;
  /* USER CODE END 2 */

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

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

      result = ADS1115_getData(pADS);//делаем тестовое преобразование
      if (result < 0xFFFF) //если результат не превышает максимального, то остаемся на первом канале
      {
            voltage = raw_to_voltage(result, pADS);//
      }
      else
      {
            configReg.channel = Channel_2; // иначе переключаемся на другой канал
            ADS1115_updateConfig(pADS, configReg); //обновляем конфигурацию
            result = ADS1115_getData(pADS);//считываем результат заново
      }



       //ST7735_FillScreen(ST7735_BLACK);

       sprintf(Buffer1,"%.3f d", voltage);

       HAL_Delay(10);

       ST7735_DrawString(30, 30,  Buffer1, Font_16x26, ST7735_GREEN, ST7735_BLACK);

       HAL_Delay(100);

       CAN_Data_Send((uint8_t)result);


  }
  /* USER CODE END 3 */
}
```

```
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief CAN Initialization Function
  * @param None
  * @retval None
  */
static void MX_CAN_Init(void)
{

  /* USER CODE BEGIN CAN_Init 0 */

  /* USER CODE END CAN_Init 0 */

  /* USER CODE BEGIN CAN_Init 1 */

  /* USER CODE END CAN_Init 1 */
  hcan.Instance = CAN1;
  hcan.Init.Prescaler = 16;
  hcan.Init.Mode = CAN_MODE_NORMAL;
  hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan.Init.TimeSeg1 = CAN_BS1_1TQ;
  hcan.Init.TimeSeg2 = CAN_BS2_1TQ;
  hcan.Init.TimeTriggeredMode = DISABLE;
```

```
  hcan.Init.AutoBusOff = DISABLE;
  hcan.Init.AutoWakeUp = DISABLE;
  hcan.Init.AutoRetransmission = DISABLE;
  hcan.Init.ReceiveFifoLocked = DISABLE;
  hcan.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN_Init 2 */

  /* USER CODE END CAN_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */
```

```
  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_4;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_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_Channel3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel3_IRQn);

}

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

  /* 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, ST7735_RES_Pin|ST7735_DC_Pin|ST7735_CS_Pin|ST7735_BL_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pins : ST7735_RES_Pin ST7735_DC_Pin ST7735_CS_Pin ST7735_BL_Pin */
```

```
  GPIO_InitStruct.Pin = ST7735_RES_Pin|ST7735_DC_Pin|ST7735_CS_Pin|ST7735_BL_Pin;
  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 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 */

  /* 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,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```