/\* USER CODE BEGIN Header \*/

/\*\*

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  \* @file           : main.c

  \* @brief          : Main program body

  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  \* @attention

  \*

  \* Copyright (c) 2023 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 <stdio.h>

#include "stm32f0xx.h"

#include <lcd\_stm32f0.c>

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

// Definitions for SPI usage

#define MEM\_SIZE 8192 // bytes

#define WREN 0b00000110 // enable writing

#define WRDI 0b00000100 // disable writing

#define RDSR 0b00000101 // read status register

#define WRSR 0b00000001 // write status register

#define READ 0b00000011

#define WRITE 0b00000010

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

ADC\_HandleTypeDef hadc;

TIM\_HandleTypeDef htim3;

TIM\_HandleTypeDef htim6;

TIM\_HandleTypeDef htim16;

/\* USER CODE BEGIN PV \*/

// TODO: Define input variables

uint32\_t current\_time = 0;

uint32\_t prev\_time = 0;

uint32\_t delay\_led = 500; //500ms delay

//array of 8-bit binary integers

uint8\_t data[6] = {0b10101010, 0b01010101, 0b11001100, 0b00110011, 0b11110000, 0b00001111}; //Data array

uint16\_t address = 0;//EEprom address

uint32\_t adc\_val;

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_ADC\_Init(void);

static void MX\_TIM3\_Init(void);

static void MX\_TIM16\_Init(void);

static void MX\_TIM6\_Init(void);

/\* USER CODE BEGIN PFP \*/

void EXTI0\_1\_IRQHandler(void);

void TIM16\_IRQHandler(void);

void writeLCD(char \*char\_in);

// ADC functions

uint32\_t pollADC(void);

uint32\_t ADCtoCCR(uint32\_t adc\_val);

// SPI functions

static void init\_spi(void);

static void write\_to\_address(uint16\_t address, uint8\_t data);

static uint8\_t read\_from\_address(uint16\_t address);

static void spi\_delay(uint32\_t delay\_in\_us);

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* 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 \*/

  init\_spi();

  MX\_GPIO\_Init();

  MX\_ADC\_Init();

  MX\_TIM3\_Init();

  MX\_TIM16\_Init();

  MX\_TIM6\_Init();

  /\* USER CODE BEGIN 2 \*/

  // Initialise LCD

  init\_LCD();

  // Start timers

  HAL\_TIM\_Base\_Start\_IT(&htim6);

  HAL\_TIM\_Base\_Start\_IT(&htim16);

  // PWM setup

  uint32\_t CCR = 0;

  HAL\_TIM\_PWM\_Start(&htim3, TIM\_CHANNEL\_3); // Start PWM on TIM3 Channel 3

  // TODO: Write bytes to EEPROM using "write\_to\_address"

  uint8\_t index = 0;

  while(index < 6){

    write\_to\_address(address, data[index]);

    index++;

    spi\_delay(100);

  }

  /\* USER CODE END 2 \*/

  /\* Infinite loop \*/

  /\* USER CODE BEGIN WHILE \*/

  while (1)

  {

  // TODO: Poll ADC

adc\_val = pollADC();//read analogue adc value from potentiometer

  // TODO: Get CRR

  CCR = ADCtoCCR(adc\_val);

  // Update PWM value   ( divide by 4  to make it turn off)

  \_\_HAL\_TIM\_SetCompare(&htim3, TIM\_CHANNEL\_3, CCR/4);

    /\* USER CODE END WHILE \*/

    /\* USER CODE BEGIN 3 \*/

  }

  /\* USER CODE END 3 \*/

}

/\*\*

  \* @brief System Clock Configuration

  \* @retval None

  \*/

void SystemClock\_Config(void)

{

  LL\_FLASH\_SetLatency(LL\_FLASH\_LATENCY\_0);

  while(LL\_FLASH\_GetLatency() != LL\_FLASH\_LATENCY\_0)

  {

  }

  LL\_RCC\_HSI\_Enable();

   /\* Wait till HSI is ready \*/

  while(LL\_RCC\_HSI\_IsReady() != 1)

  {

  }

  LL\_RCC\_HSI\_SetCalibTrimming(16);

  LL\_RCC\_HSI14\_Enable();

   /\* Wait till HSI14 is ready \*/

  while(LL\_RCC\_HSI14\_IsReady() != 1)

  {

  }

  LL\_RCC\_HSI14\_SetCalibTrimming(16);

  LL\_RCC\_SetAHBPrescaler(LL\_RCC\_SYSCLK\_DIV\_1);

  LL\_RCC\_SetAPB1Prescaler(LL\_RCC\_APB1\_DIV\_1);

  LL\_RCC\_SetSysClkSource(LL\_RCC\_SYS\_CLKSOURCE\_HSI);

   /\* Wait till System clock is ready \*/

  while(LL\_RCC\_GetSysClkSource() != LL\_RCC\_SYS\_CLKSOURCE\_STATUS\_HSI)

  {

  }

  LL\_SetSystemCoreClock(8000000);

   /\* Update the time base \*/

  if (HAL\_InitTick (TICK\_INT\_PRIORITY) != HAL\_OK)

  {

    Error\_Handler();

  }

  LL\_RCC\_HSI14\_EnableADCControl();

}

/\*\*

  \* @brief ADC Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_ADC\_Init(void)

{

  /\* USER CODE BEGIN ADC\_Init 0 \*/

  /\* USER CODE END ADC\_Init 0 \*/

  ADC\_ChannelConfTypeDef sConfig = {0};

  /\* USER CODE BEGIN ADC\_Init 1 \*/

  /\* USER CODE END ADC\_Init 1 \*/

  /\*\* Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

  \*/

  hadc.Instance = ADC1;

  hadc.Init.ClockPrescaler = ADC\_CLOCK\_ASYNC\_DIV1;

  hadc.Init.Resolution = ADC\_RESOLUTION\_12B;

  hadc.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

  hadc.Init.ScanConvMode = ADC\_SCAN\_DIRECTION\_FORWARD;

  hadc.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

  hadc.Init.LowPowerAutoWait = DISABLE;

  hadc.Init.LowPowerAutoPowerOff = DISABLE;

  hadc.Init.ContinuousConvMode = DISABLE;

  hadc.Init.DiscontinuousConvMode = DISABLE;

  hadc.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

  hadc.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

  hadc.Init.DMAContinuousRequests = DISABLE;

  hadc.Init.Overrun = ADC\_OVR\_DATA\_PRESERVED;

  if (HAL\_ADC\_Init(&hadc) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\*\* Configure for the selected ADC regular channel to be converted.

  \*/

  sConfig.Channel = ADC\_CHANNEL\_6;

  sConfig.Rank = ADC\_RANK\_CHANNEL\_NUMBER;

  sConfig.SamplingTime = ADC\_SAMPLETIME\_1CYCLE\_5;

  if (HAL\_ADC\_ConfigChannel(&hadc, &sConfig) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\* USER CODE BEGIN ADC\_Init 2 \*/

  ADC1->CR |= ADC\_CR\_ADCAL;

  while(ADC1->CR & ADC\_CR\_ADCAL);     // Calibrate the ADC

  ADC1->CR |= (1 << 0);           // Enable ADC

  while((ADC1->ISR & (1 << 0)) == 0);   // Wait for ADC ready

  /\* USER CODE END ADC\_Init 2 \*/

}

/\*\*

  \* @brief TIM3 Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_TIM3\_Init(void)

{

  /\* USER CODE BEGIN TIM3\_Init 0 \*/

  /\* USER CODE END TIM3\_Init 0 \*/

  TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

  TIM\_MasterConfigTypeDef sMasterConfig = {0};

  TIM\_OC\_InitTypeDef sConfigOC = {0};

  /\* USER CODE BEGIN TIM3\_Init 1 \*/

  /\* USER CODE END TIM3\_Init 1 \*/

  htim3.Instance = TIM3;

  htim3.Init.Prescaler = 0;

  htim3.Init.CounterMode = TIM\_COUNTERMODE\_UP;

  htim3.Init.Period = 47999;

  htim3.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

  htim3.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

  if (HAL\_TIM\_Base\_Init(&htim3) != HAL\_OK)

  {

    Error\_Handler();

  }

  sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

  if (HAL\_TIM\_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL\_OK)

  {

    Error\_Handler();

  }

  if (HAL\_TIM\_PWM\_Init(&htim3) != HAL\_OK)

  {

    Error\_Handler();

  }

  sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

  sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

  if (HAL\_TIMEx\_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL\_OK)

  {

    Error\_Handler();

  }

  sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

  sConfigOC.Pulse = 0;

  sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

  sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

  if (HAL\_TIM\_PWM\_ConfigChannel(&htim3, &sConfigOC, TIM\_CHANNEL\_3) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\* USER CODE BEGIN TIM3\_Init 2 \*/

  /\* USER CODE END TIM3\_Init 2 \*/

  HAL\_TIM\_MspPostInit(&htim3);

}

/\*\*

  \* @brief TIM6 Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_TIM6\_Init(void)

{

  /\* USER CODE BEGIN TIM6\_Init 0 \*/

  /\* USER CODE END TIM6\_Init 0 \*/

  TIM\_MasterConfigTypeDef sMasterConfig = {0};

  /\* USER CODE BEGIN TIM6\_Init 1 \*/

  /\* USER CODE END TIM6\_Init 1 \*/

  htim6.Instance = TIM6;

  htim6.Init.Prescaler = 8000-1;

  htim6.Init.CounterMode = TIM\_COUNTERMODE\_UP;

  htim6.Init.Period = 500-1;

  htim6.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_ENABLE;

  if (HAL\_TIM\_Base\_Init(&htim6) != HAL\_OK)

  {

    Error\_Handler();

  }

  sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

  sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

  if (HAL\_TIMEx\_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\* USER CODE BEGIN TIM6\_Init 2 \*/

  NVIC\_EnableIRQ(TIM6\_IRQn);

  /\* USER CODE END TIM6\_Init 2 \*/

}

/\*\*

  \* @brief TIM16 Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_TIM16\_Init(void)

{

  /\* USER CODE BEGIN TIM16\_Init 0 \*/

  /\* USER CODE END TIM16\_Init 0 \*/

  /\* USER CODE BEGIN TIM16\_Init 1 \*/

  /\* USER CODE END TIM16\_Init 1 \*/

  htim16.Instance = TIM16;

  htim16.Init.Prescaler = 8000-1;

  htim16.Init.CounterMode = TIM\_COUNTERMODE\_UP;

  htim16.Init.Period = 1000-1;

  htim16.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

  htim16.Init.RepetitionCounter = 0;

  htim16.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_ENABLE;

  if (HAL\_TIM\_Base\_Init(&htim16) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\* USER CODE BEGIN TIM16\_Init 2 \*/

  NVIC\_EnableIRQ(TIM16\_IRQn);

  /\* USER CODE END TIM16\_Init 2 \*/

}

/\*\*

  \* @brief GPIO Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_GPIO\_Init(void)

{

  LL\_EXTI\_InitTypeDef EXTI\_InitStruct = {0};

  LL\_GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

  /\* GPIO Ports Clock Enable \*/

  LL\_AHB1\_GRP1\_EnableClock(LL\_AHB1\_GRP1\_PERIPH\_GPIOF);

  LL\_AHB1\_GRP1\_EnableClock(LL\_AHB1\_GRP1\_PERIPH\_GPIOA);

  LL\_AHB1\_GRP1\_EnableClock(LL\_AHB1\_GRP1\_PERIPH\_GPIOB);

  /\*\*/

  LL\_GPIO\_ResetOutputPin(LED7\_GPIO\_Port, LED7\_Pin);

  /\*\*/

  LL\_SYSCFG\_SetEXTISource(LL\_SYSCFG\_EXTI\_PORTA, LL\_SYSCFG\_EXTI\_LINE0);

  /\*\*/

  LL\_GPIO\_SetPinPull(Button0\_GPIO\_Port, Button0\_Pin, LL\_GPIO\_PULL\_UP);

  /\*\*/

  LL\_GPIO\_SetPinMode(Button0\_GPIO\_Port, Button0\_Pin, LL\_GPIO\_MODE\_INPUT);

  /\*\*/

  EXTI\_InitStruct.Line\_0\_31 = LL\_EXTI\_LINE\_0;

  EXTI\_InitStruct.LineCommand = ENABLE;

  EXTI\_InitStruct.Mode = LL\_EXTI\_MODE\_IT;

  EXTI\_InitStruct.Trigger = LL\_EXTI\_TRIGGER\_RISING;

  LL\_EXTI\_Init(&EXTI\_InitStruct);

  /\*\*/

  GPIO\_InitStruct.Pin = LED7\_Pin;

  GPIO\_InitStruct.Mode = LL\_GPIO\_MODE\_OUTPUT;

  GPIO\_InitStruct.Speed = LL\_GPIO\_SPEED\_FREQ\_LOW;

  GPIO\_InitStruct.OutputType = LL\_GPIO\_OUTPUT\_PUSHPULL;

  GPIO\_InitStruct.Pull = LL\_GPIO\_PULL\_NO;

  LL\_GPIO\_Init(LED7\_GPIO\_Port, &GPIO\_InitStruct);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

  HAL\_NVIC\_SetPriority(EXTI0\_1\_IRQn, 0, 0);

  HAL\_NVIC\_EnableIRQ(EXTI0\_1\_IRQn);

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

void EXTI0\_1\_IRQHandler(void)

{

  // TODO: Add code to switch LED7 delay frequency

  current\_time = HAL\_GetTick();

  //ensures unwanted noise within duration is not registered

  if((current\_time - prev\_time)> 200){

    if(delay\_led = 500 ){ //if frequency of led is 2Hz

      delay\_led = 1000;//toggle the frequency of LED by changing delay

      htim6.Init.Period = delay\_led -1;

    }else if(delay\_led = 1000){ //if frequency of led is 1Hz

      delay\_led = 1000;

      htim6.Init.Period = delay\_led -1;

    }

    //update TIM6 with the new period; ensure execution complete

        if (HAL\_TIM\_Base\_Init(&htim6) != HAL\_OK)

        {

          Error\_Handler();

        }

  }

  prev\_time = current\_time;//update the last time since click

  HAL\_GPIO\_EXTI\_IRQHandler(Button0\_Pin); // Clear interrupt flags

}

void TIM6\_IRQHandler(void)

{

  // Acknowledge interrupt

  HAL\_TIM\_IRQHandler(&htim6);

  // Toggle LED7

  HAL\_GPIO\_TogglePin(GPIOB, LED7\_Pin);

}

void TIM16\_IRQHandler(void)

{

  // Acknowledge interrupt

  HAL\_TIM\_IRQHandler(&htim16);

  // TODO: Initialise a string to output second line on LC-D

  char charArray[16];//buffer

  // TODO: Change LED pattern; output 0x01 if the read SPI data is incorrect

  if (address > 5){

      address= 0;

    }

    //validate byte at address

  uint8\_t num = read\_from\_address(address);

  spi\_delay(100);

  snprintf(charArray, sizeof(charArray), "%d", read\_from\_address(address));

  writeLCD(charArray);

  //iterate address

  address++;

}

// TODO: Complete the writeLCD function

void writeLCD(char \*char\_in){

  delay(3000);

  lcd\_command(CLEAR);

  lcd\_putstring("EEPROM byte:");

  lcd\_command(LINE\_TWO);

  lcd\_putstring(char\_in);

}

// Get ADC value

uint32\_t pollADC(void){

  HAL\_ADC\_Start(&hadc); // start the adc

  HAL\_ADC\_PollForConversion(&hadc, 100); // poll for conversion

  uint32\_t val = HAL\_ADC\_GetValue(&hadc); // get the adc value

  HAL\_ADC\_Stop(&hadc); // stop adc

  return val;

}

// Calculate PWM CCR value

uint32\_t ADCtoCCR(uint32\_t adc\_val){

  // TODO: Calculate CCR value (val) using an appropriate equation

  uint32\_t val\_ccr;

  val\_ccr = (adc\_val \* 47999) / 4095;

  return val\_ccr;

}

void ADC1\_COMP\_IRQHandler(void)

{

  adc\_val = HAL\_ADC\_GetValue(&hadc); // read adc value

  HAL\_ADC\_IRQHandler(&hadc); //Clear flags

}

// Initialise SPI

static void init\_spi(void) {

  // Clock to PB

  RCC->AHBENR |= RCC\_AHBENR\_GPIOBEN;  // Enable clock for SPI port

  // Set pin modes

  GPIOB->MODER |= GPIO\_MODER\_MODER13\_1; // Set pin SCK (PB13) to Alternate Function

  GPIOB->MODER |= GPIO\_MODER\_MODER14\_1; // Set pin MISO (PB14) to Alternate Function

  GPIOB->MODER |= GPIO\_MODER\_MODER15\_1; // Set pin MOSI (PB15) to Alternate Function

  GPIOB->MODER |= GPIO\_MODER\_MODER12\_0; // Set pin CS (PB12) to output push-pull

  GPIOB->BSRR |= GPIO\_BSRR\_BS\_12;     // Pull CS high

  // Clock enable to SPI

  RCC->APB1ENR |= RCC\_APB1ENR\_SPI2EN;

  SPI2->CR1 |= SPI\_CR1\_BIDIOE;                  // Enable output

  SPI2->CR1 |= (SPI\_CR1\_BR\_0 |  SPI\_CR1\_BR\_1);          // Set Baud to fpclk / 16

  SPI2->CR1 |= SPI\_CR1\_MSTR;                  // Set to master mode

  SPI2->CR2 |= SPI\_CR2\_FRXTH;                   // Set RX threshold to be 8 bits

  SPI2->CR2 |= SPI\_CR2\_SSOE;                  // Enable slave output to work in master mode

  SPI2->CR2 |= (SPI\_CR2\_DS\_0 | SPI\_CR2\_DS\_1 | SPI\_CR2\_DS\_2);  // Set to 8-bit mode

  SPI2->CR1 |= SPI\_CR1\_SPE;                   // Enable the SPI peripheral

}

// Implements a delay in microseconds

static void spi\_delay(uint32\_t delay\_in\_us) {

  volatile uint32\_t counter = 0;

  delay\_in\_us \*= 3;

  for(; counter < delay\_in\_us; counter++) {

    \_\_asm("nop");

    \_\_asm("nop");

  }

}

// Write to EEPROM address using SPI

static void write\_to\_address(uint16\_t address, uint8\_t data) {

  uint8\_t dummy; // Junk from the DR

  // Set the Write Enable latch

  GPIOB->BSRR |= GPIO\_BSRR\_BR\_12; // Pull CS low

  spi\_delay(1);

  \*((uint8\_t\*)(&SPI2->DR)) = WREN;

  while ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

  dummy = SPI2->DR;

  GPIOB->BSRR |= GPIO\_BSRR\_BS\_12; // Pull CS high

  spi\_delay(5000);

  // Send write instruction

  GPIOB->BSRR |= GPIO\_BSRR\_BR\_12;       // Pull CS low

  spi\_delay(1);

  \*((uint8\_t\*)(&SPI2->DR)) = WRITE;

  while ((SPI2->SR & SPI\_SR\_RXNE) == 0);    // Hang while RX is empty

  dummy = SPI2->DR;

  // Send 16-bit address

  \*((uint8\_t\*)(&SPI2->DR)) = (address >> 8);  // Address MSB

  while ((SPI2->SR & SPI\_SR\_RXNE) == 0);    // Hang while RX is empty

  dummy = SPI2->DR;

  \*((uint8\_t\*)(&SPI2->DR)) = (address);     // Address LSB

  while ((SPI2->SR & SPI\_SR\_RXNE) == 0);    // Hang while RX is empty

  dummy = SPI2->DR;

  // Send the data

  \*((uint8\_t\*)(&SPI2->DR)) = data;

  while ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

  dummy = SPI2->DR;

  GPIOB->BSRR |= GPIO\_BSRR\_BS\_12; // Pull CS high

  spi\_delay(5000);

}

// Read from EEPROM address using SPI

static uint8\_t read\_from\_address(uint16\_t address) {

  uint8\_t dummy; // Junk from the DR

  // Send the read instruction

  GPIOB->BSRR |= GPIO\_BSRR\_BR\_12;       // Pull CS low

  spi\_delay(1);

  \*((uint8\_t\*)(&SPI2->DR)) = READ;

  while ((SPI2->SR & SPI\_SR\_RXNE) == 0);    // Hang while RX is empty

  dummy = SPI2->DR;

  // Send 16-bit address

  \*((uint8\_t\*)(&SPI2->DR)) = (address >> 8);  // Address MSB

  while ((SPI2->SR & SPI\_SR\_RXNE) == 0);    // Hang while RX is empty

  dummy = SPI2->DR;

  \*((uint8\_t\*)(&SPI2->DR)) = (address);     // Address LSB

  while ((SPI2->SR & SPI\_SR\_RXNE) == 0);    // Hang while RX is empty

  dummy = SPI2->DR;

  // Clock in the data

  \*((uint8\_t\*)(&SPI2->DR)) = 0x42;      // Clock out some junk data

  while ((SPI2->SR & SPI\_SR\_RXNE) == 0);    // Hang while RX is empty

  dummy = SPI2->DR;

  GPIOB->BSRR |= GPIO\_BSRR\_BS\_12;       // Pull CS high

  spi\_delay(5000);

  return dummy;               // Return read data

}

/\* 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 \*/