```
/* 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"
#include "tim.h"
#include "gpio.h"

/* Private includes
----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdint.h>
#include "stm32f0xx.h"
/* 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
---------------------------------------------------------*/

/* USER CODE BEGIN PV */
// TODO: Define any input variables
static uint8_t patterns[] = {0b10101010, 0b01010101, 0b11001100,
0b00110011, 0b11110000, 0b00001111};
uint8_t eeprom_address = 0;

/* USER CODE END PV */

/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
void EXTI0_1_IRQHandler(void);
void TIM16_IRQHandler(void);
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 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 */
    init_spi();
    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_TIM16_Init();
    /* USER CODE BEGIN 2 */
    // TODO: Start timer TIM16
    if (HAL_TIM_Base_Start_IT(&htim16) != HAL_OK) {
        Error_Handler();
    }

    // TODO: Write all "patterns" to EEPROM using SPI
    for (uint8_t i = 0; i < 6; i++) write_to_address(i,
patterns[i]);

    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    uint8_t Button0Pushed = 0b0;
    while (1)
    {
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */

        // TODO: Check button PA0; if pressed, change timer delay
        if ((HAL_GPIO_ReadPin(Button0_GPIO_Port, Button0_Pin) ==
GPIO_PIN_RESET) && (Button0Pushed == 0b0)) {
            Button0Pushed = 0b1;
            if (TIM16->ARR == 500-1) TIM16->ARR = 1000-1;
            else TIM16->ARR = 500-1;
        }

        if (HAL_GPIO_ReadPin(Button0_GPIO_Port, Button0_Pin) ==
GPIO_PIN_SET) Button0Pushed = 0b0;
    }
    /* 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_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();
    }
}

/* USER CODE BEGIN 4 */

// 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 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
    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
    delay(5000);

    // Send write instruction
    GPIOB->BSRR |= GPIO_BSRR_BR_12;                    // Pull CS
low
    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
    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
    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
    delay(5000);

    return dummy;
                            // Return read data
```

```
}

// Timer rolled over
void TIM16_IRQHandler(void)
{
    // Acknowledge interrupt
    HAL_TIM_IRQHandler(&htim16);

    // TODO: Change to next LED pattern; output 0x01 if the read SPI
data is incorrect
    uint8_t led_pattern = read_from_address(eeprom_address);

    GPIOB->ODR &= 0xFF00;
    if (led_pattern == patterns[eeprom_address])
        GPIOB->ODR |= led_pattern;
    else
        GPIOB->ODR |= 0b1;

    eeprom_address++;
    eeprom_address %= 6;
}

/* 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 */
```