```
/* 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 <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 ---------------------------------------------------
---*/
```

```
TIM_HandleTypeDef htim16;

/* USER CODE BEGIN PV */
// TODO: Define any input variables
uint8_t binArr[] = { 0xAA, // 10101010 in binary
            0x55, // 01010101 in binary
            0xCC, // 11001100 in binary
            0x33, // 00110011 in binary
            0xF0, // 11110000 in binary
            0x0F  // 00001111 in binary
            };
static uint8_t patterns[] = {};
volatile uint32_t delayDuration = 1;
static uint16_t Index = 0;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------
---*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM16_Init(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
	HAL_TIM_Base_Start_IT(&htim16);

	// TODO: Write all "patterns" to EEPROM using SPI
	for (uint16_t address = 0; address < sizeof(binArr); address++) {
		write_to_address(address, binArr[address]);
	}

	/* USER CODE END 2 */

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

		/* USER CODE END WHILE */

		/* USER CODE BEGIN 3 */
		// TODO: Check button PA0; if pressed, change timer delay
		uint16_t indata = GPIOA->IDR;
	// Check if Pushbutton 0 is pressed

		if((indata & GPIO_IDR_0)==0) {
			// Button is pressed, toggle between 1-second and half-
second delay

			if (delayDuration == 1) {
				delayDuration = 0.5;// change flag to allow
alternation
				TIM16 -> ARR = 500;// 0.5 second delay using timer

			} else{
				delayDuration = 1; // change flag to allow
alternation
				TIM16 -> ARR = 1000;// 1 second delay using timer
			}

				// 1 second delay
			}
		//}
		/* 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();
		}
	}

	/**
	* @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(LED0_GPIO_Port, LED0_Pin);

            /**/
            LL_GPIO_ResetOutputPin(LED1_GPIO_Port, LED1_Pin);

            /**/
            LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);

            /**/
            LL_GPIO_ResetOutputPin(LED3_GPIO_Port, LED3_Pin);

            /**/
            LL_GPIO_ResetOutputPin(LED4_GPIO_Port, LED4_Pin);

            /**/
            LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);

            /**/
            LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);

            /**/
            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 = LED0_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(LED0_GPIO_Port, &GPIO_InitStruct);

            /**/
            GPIO_InitStruct.Pin = LED1_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(LED1_GPIO_Port, &GPIO_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED2_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(LED2_GPIO_Port, &GPIO_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED3_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(LED3_GPIO_Port, &GPIO_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED4_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(LED4_GPIO_Port, &GPIO_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED5_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(LED5_GPIO_Port, &GPIO_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED6_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(LED6_GPIO_Port, &GPIO_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 */
  /* USER CODE END MX_GPIO_Init_2 */
}

/* 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
      }
```

```
      void set_leds(uint8_t data) {
          // Configure GPIO pins for the LEDs as outputs
          GPIOB->ODR = data;
          // ... Repeat for other LEDs as needed ...
      }

// 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 data = read_from_address(Index);
            set_leds(data);
            if (data != binArr[Index]) {
                  // SPI failure, set LEDs to 0b00000001
            set_leds(0x01);
            } else {
                  // Display the binary value on the LEDs
                  set_leds(data);
            }
            Index++;
            if (Index >= sizeof(binArr)) {
                  Index = 0; // back to the beginning
            }
            __HAL_TIM_CLEAR_FLAG(&htim16, TIM_FLAG_UPDATE);
      }

      /* USER CODE END 4 */

      /**
      * @brief  This function is executed in case of error occurrence.
      * @retval None
      */
      // Function to set LEDs based on the binary data

      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 */
```