```c
/* 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
static uint8_t patterns[] = {0b10101010, 0b01010101, 0b11001100, 0b00110011, 0b11110000,
0b00001111};// creates an array of 6 to create patterns on board
uint16_t x =0;
int y = 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);
```

```c
69  static void init_spi(void);
70  static void write_to_address(uint16_t address, uint8_t data);
71  static uint8_t read_from_address(uint16_t address);
72  static void delay(uint32_t delay_in_us);
73  /* USER CODE END PFP */
74
75  /* Private user code ---------------------------------------------------------*/
76  /* USER CODE BEGIN 0 */
77
78  /* USER CODE END 0 */
79
80  /**
81    * @brief  The application entry point.
82    * @retval int
83    */
84  int main(void)
85  {
86    /* USER CODE BEGIN 1 */
87    /* USER CODE END 1 */
88
89    /* MCU Configuration--------------------------------------------------------*/
90
91    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
92    HAL_Init();
93
94    /* USER CODE BEGIN Init */
95    /* USER CODE END Init */
96
97    /* Configure the system clock */
98    SystemClock_Config();
99
100   /* USER CODE BEGIN SysInit */
101   init_spi();
102   /* USER CODE END SysInit */
103
104   /* Initialize all configured peripherals */
105   MX_GPIO_Init();
106   MX_TIM16_Init();
107   /* USER CODE BEGIN 2 */
108
109   // TODO: Start timer TIM16
110 HAL_TIM_Base_Start_IT(&htim16); //enables timer start in interrupt mode
111
112   // TODO: Write all "patterns" to EEPROM using SPI
113 for (int i=0;i<6;i=i+1)
114 {
115     write_to_address(i,patterns[i]);// write
116 }
117
118   /* USER CODE END 2 */
119
120   /* Infinite loop */
121   /* USER CODE BEGIN WHILE */
122   while (1)
123   {
124     /* USER CODE END WHILE */
125
126     /* USER CODE BEGIN 3 */
127
128     // TODO: Check button PA0; if pressed, change timer delay
129       if (checkPB()==1){
130           if (y ==0){
131               htim16.Instance->ARR =500;
132               y = 1;
133           }
134           else{
135               htim16.Instance->ARR =1000;
136               y = 0;
137           }
```

```
        }
    }
    /* 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);
```

```
207     /* USER CODE END TIM16_Init 2 */
208
209   }
210
211   /**
212     * @brief GPIO Initialization Function
213     * @param None
214     * @retval None
215     */
216   static void MX_GPIO_Init(void)
217   {
218     LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
219     LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
220   /* USER CODE BEGIN MX_GPIO_Init_1 */
221   /* USER CODE END MX_GPIO_Init_1 */
222
223     /* GPIO Ports Clock Enable */
224     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
225     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
226     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
227
228     /**/
229     LL_GPIO_ResetOutputPin(LED0_GPIO_Port, LED0_Pin);
230
231     /**/
232     LL_GPIO_ResetOutputPin(LED1_GPIO_Port, LED1_Pin);
233
234     /**/
235     LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
236
237     /**/
238     LL_GPIO_ResetOutputPin(LED3_GPIO_Port, LED3_Pin);
239
240     /**/
241     LL_GPIO_ResetOutputPin(LED4_GPIO_Port, LED4_Pin);
242
243     /**/
244     LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);
245
246     /**/
247     LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);
248
249     /**/
250     LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
251
252     /**/
253     LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);
254
255     /**/
256     LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);
257
258     /**/
259     LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);
260
261     /**/
262     EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
263     EXTI_InitStruct.LineCommand = ENABLE;
264     EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
265     EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
266     LL_EXTI_Init(&EXTI_InitStruct);
267
268     /**/
269     GPIO_InitStruct.Pin = LED0_Pin;
270     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
271     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
272     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
273     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
274     LL_GPIO_Init(LED0_GPIO_Port, &GPIO_InitStruct);
275
```

```
276     /**/
277     GPIO_InitStruct.Pin = LED1_Pin;
278     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
279     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
280     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
281     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
282     LL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
283
284     /**/
285     GPIO_InitStruct.Pin = LED2_Pin;
286     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
287     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
288     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
289     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
290     LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
291
292     /**/
293     GPIO_InitStruct.Pin = LED3_Pin;
294     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
295     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
296     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
297     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
298     LL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);
299
300     /**/
301     GPIO_InitStruct.Pin = LED4_Pin;
302     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
303     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
304     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
305     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
306     LL_GPIO_Init(LED4_GPIO_Port, &GPIO_InitStruct);
307
308     /**/
309     GPIO_InitStruct.Pin = LED5_Pin;
310     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
311     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
312     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
313     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
314     LL_GPIO_Init(LED5_GPIO_Port, &GPIO_InitStruct);
315
316     /**/
317     GPIO_InitStruct.Pin = LED6_Pin;
318     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
319     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
320     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
321     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
322     LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);
323
324     /**/
325     GPIO_InitStruct.Pin = LED7_Pin;
326     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
327     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
328     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
329     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
330     LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
331
332   /* USER CODE BEGIN MX_GPIO_Init_2 */
333   /* USER CODE END MX_GPIO_Init_2 */
334   }
335
336   /* USER CODE BEGIN 4 */
337
338   // Initialise SPI
339   static void init_spi(void) {
340
341     // Clock to PB
342     RCC->AHBENR |= RCC_AHBENR_GPIOBEN;    // Enable clock for SPI port
343
344     // Set pin modes
```

```
345     GPIOB->MODER |= GPIO_MODER_MODER13_1; // Set pin SCK (PB13) to Alternate Function
346     GPIOB->MODER |= GPIO_MODER_MODER14_1; // Set pin MISO (PB14) to Alternate Function
347     GPIOB->MODER |= GPIO_MODER_MODER15_1; // Set pin MOSI (PB15) to Alternate Function
348     GPIOB->MODER |= GPIO_MODER_MODER12_0; // Set pin CS (PB12) to output push-pull
349     GPIOB->BSRR |= GPIO_BSRR_BS_12;       // Pull CS high
350
351     // Clock enable to SPI
352     RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
353     SPI2->CR1 |= SPI_CR1_BIDIOE;                                    // Enable output
354     SPI2->CR1 |= (SPI_CR1_BR_0 |  SPI_CR1_BR_1);                    // Set Baud to fpclk / 16
355     SPI2->CR1 |= SPI_CR1_MSTR;                                      // Set to master mode
356     SPI2->CR2 |= SPI_CR2_FRXTH;                                     // Set RX threshold to be 8
        bits
357     SPI2->CR2 |= SPI_CR2_SSOE;                                      // Enable slave output to
        work in master mode
358     SPI2->CR2 |= (SPI_CR2_DS_0 | SPI_CR2_DS_1 | SPI_CR2_DS_2);     // Set to 8-bit mode
359     SPI2->CR1 |= SPI_CR1_SPE;                                       // Enable the SPI peripheral
360   }
361
362   // Implements a delay in microseconds
363   static void delay(uint32_t delay_in_us) {
364     volatile uint32_t counter = 0;
365     delay_in_us *= 3;
366     for(; counter < delay_in_us; counter++) {
367       __asm("nop");
368       __asm("nop");
369     }
370   }
371
372   // Write to EEPROM address using SPI
373   static void write_to_address(uint16_t address, uint8_t data) {
374
375       uint8_t dummy; // Junk from the DR
376
377       // Set the Write Enable latch
378       GPIOB->BSRR |= GPIO_BSRR_BR_12; // Pull CS low
379       delay(1);
380       *((uint8_t*)(&SPI2->DR)) = WREN;
381       while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
382       dummy = SPI2->DR;
383       GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
384       delay(5000);
385
386       // Send write instruction
387       GPIOB->BSRR |= GPIO_BSRR_BR_12;             // Pull CS low
388       delay(1);
389       *((uint8_t*)(&SPI2->DR)) = WRITE;
390       while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
391       dummy = SPI2->DR;
392
393       // Send 16-bit address
394       *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
395       while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
396       dummy = SPI2->DR;
397       *((uint8_t*)(&SPI2->DR)) = (address);       // Address LSB
398       while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
399       dummy = SPI2->DR;
400
401       // Send the data
402       *((uint8_t*)(&SPI2->DR)) = data;
403       while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
404       dummy = SPI2->DR;
405       GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
406       delay(5000);
407   }
408
409   // Read from EEPROM address using SPI
410   static uint8_t read_from_address(uint16_t address) {
411
```

```
412       uint8_t dummy; // Junk from the DR
413
414       // Send the read instruction
415       GPIOB->BSRR |= GPIO_BSRR_BR_12;             // Pull CS low
416       delay(1);
417       *((uint8_t*)(&SPI2->DR)) = READ;
418       while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
419       dummy = SPI2->DR;
420
421       // Send 16-bit address
422       *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
423       while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
424       dummy = SPI2->DR;
425       *((uint8_t*)(&SPI2->DR)) = (address);       // Address LSB
426       while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
427       dummy = SPI2->DR;
428
429       // Clock in the data
430       *((uint8_t*)(&SPI2->DR)) = 0x42;                // Clock out some junk data
431       while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
432       dummy = SPI2->DR;
433       GPIOB->BSRR |= GPIO_BSRR_BS_12;                 // Pull CS high
434       delay(5000);
435
436       return dummy;                                               // Return read data
437   }
438
439   // Timer rolled over
440   void TIM16_IRQHandler(void)
441   {
442       // Acknowledge interrupt
443       HAL_TIM_IRQHandler(&htim16);
444
445       // TODO: Change to next LED pattern; output 0x01 if the read SPI data is incorrect
446       GPIOB->ODR &= read_from_address(x);  //clear the memory before going to next iteration
447       if (x>5){
448           x=0; // checks to see if addresses in
449       };
450       if(read_from_address(x)==patterns[x]){
451           GPIOB->ODR |= read_from_address(x);// sends data to pins
452
453           x=x+1;
454   }
455   else{
456       GPIOB->ODR |= 0b00000001; //indicates failure
457   }
458
459
460
461   }
462   /* USER CODE END 4 */
463   int checkPB(void){
464       if ((GPIOA -> IDR & GPIO_IDR_0)==0){//created function to chec if input to IDR is 1 or 0
465           return 1;
466       }
467       else{
468           return 0;
469       }
470   }
471
472   /**
473     * @brief  This function is executed in case of error occurrence.
474     * @retval None
475     */
476   void Error_Handler(void)
477   {
478     /* USER CODE BEGIN Error_Handler_Debug */
479     /* User can add his own implementation to report the HAL error return state */
480     __disable_irq();
```

```
481     while (1)
482     {
483     }
484     /* USER CODE END Error_Handler_Debug */
485   }
486
487   #ifdef  USE_FULL_ASSERT
488   /**
489     * @brief  Reports the name of the source file and the source line number
490     *         where the assert_param error has occurred.
491     * @param  file: pointer to the source file name
492     * @param  line: assert_param error line source number
493     * @retval None
494     */
495   void assert_failed(uint8_t *file, uint32_t line)
496   {
497     /* USER CODE BEGIN 6 */
498     /* User can add his own implementation to report the file name and line number,
499        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
500     /* USER CODE END 6 */
501   }
502   #endif /* USE_FULL_ASSERT */
503
```