```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}; //create array of patterns
//set x and y
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);
```

```c
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); //start timer in interrupt mode

  // TODO: Write all "patterns" to EEPROM using SPI
for (int i=0;i<6;i=i+1)
{
    write_to_address(i,patterns[i]); //loop through array to write each pattern
}

  /* 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
      if (checkPB()==1){
          if (y ==0){
              htim16.Instance->ARR =500;
              y = 1;
          }
          else{
              htim16.Instance->ARR =1000;
              y = 0;
```

```
138             }
139         }
140     }
141     /* USER CODE END 3 */
142   }
143
144   /**
145     * @brief System Clock Configuration
146     * @retval None
147     */
148   void SystemClock_Config(void)
149   {
150     LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
151     while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
152     {
153     }
154     LL_RCC_HSI_Enable();
155
156      /* Wait till HSI is ready */
157     while(LL_RCC_HSI_IsReady() != 1)
158     {
159
160     }
161     LL_RCC_HSI_SetCalibTrimming(16);
162     LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
163     LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
164     LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
165
166      /* Wait till System clock is ready */
167     while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
168     {
169
170     }
171     LL_SetSystemCoreClock(8000000);
172
173      /* Update the time base */
174     if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
175     {
176       Error_Handler();
177     }
178   }
179
180   /**
181     * @brief TIM16 Initialization Function
182     * @param None
183     * @retval None
184     */
185   static void MX_TIM16_Init(void)
186   {
187
188     /* USER CODE BEGIN TIM16_Init 0 */
189
190     /* USER CODE END TIM16_Init 0 */
191
192     /* USER CODE BEGIN TIM16_Init 1 */
193
194     /* USER CODE END TIM16_Init 1 */
195     htim16.Instance = TIM16;
196     htim16.Init.Prescaler = 8000-1;
197     htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
198     htim16.Init.Period = 1000-1;
199     htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
200     htim16.Init.RepetitionCounter = 0;
201     htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
202     if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
203     {
204       Error_Handler();
205     }
206     /* USER CODE BEGIN TIM16_Init 2 */
```

```
207     NVIC_EnableIRQ(TIM16_IRQn);
208     /* USER CODE END TIM16_Init 2 */
209
210   }
211
212   /**
213     * @brief GPIO Initialization Function
214     * @param None
215     * @retval None
216     */
217   static void MX_GPIO_Init(void)
218   {
219     LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
220     LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
221   /* USER CODE BEGIN MX_GPIO_Init_1 */
222   /* USER CODE END MX_GPIO_Init_1 */
223
224     /* GPIO Ports Clock Enable */
225     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
226     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
227     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
228
229     /**/
230     LL_GPIO_ResetOutputPin(LED0_GPIO_Port, LED0_Pin);
231
232     /**/
233     LL_GPIO_ResetOutputPin(LED1_GPIO_Port, LED1_Pin);
234
235     /**/
236     LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
237
238     /**/
239     LL_GPIO_ResetOutputPin(LED3_GPIO_Port, LED3_Pin);
240
241     /**/
242     LL_GPIO_ResetOutputPin(LED4_GPIO_Port, LED4_Pin);
243
244     /**/
245     LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);
246
247     /**/
248     LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);
249
250     /**/
251     LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
252
253     /**/
254     LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);
255
256     /**/
257     LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);
258
259     /**/
260     LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);
261
262     /**/
263     EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
264     EXTI_InitStruct.LineCommand = ENABLE;
265     EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
266     EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
267     LL_EXTI_Init(&EXTI_InitStruct);
268
269     /**/
270     GPIO_InitStruct.Pin = LED0_Pin;
271     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
272     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
273     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
274     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
275     LL_GPIO_Init(LED0_GPIO_Port, &GPIO_InitStruct);
```

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

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

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

```
481     __disable_irq();
482     while (1)
483     {
484     }
485     /* USER CODE END Error_Handler_Debug */
486   }
487
488   #ifdef  USE_FULL_ASSERT
489   /**
490     * @brief  Reports the name of the source file and the source line number
491     *         where the assert_param error has occurred.
492     * @param  file: pointer to the source file name
493     * @param  line: assert_param error line source number
494     * @retval None
495     */
496   void assert_failed(uint8_t *file, uint32_t line)
497   {
498     /* USER CODE BEGIN 6 */
499     /* User can add his own implementation to report the file name and line number,
500        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
501     /* USER CODE END 6 */
502   }
503   #endif /* USE_FULL_ASSERT */
504
```