```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};
static uint16_t index = 0;

```

```c
/* 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 i=0; i<8; i++){
     write_to_address(i, patterns[i]);
 }


```

```
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       GPIOA-> MODER &= ~GPIO_MODER_MODER0;
130       GPIOA->PUPDR|=GPIO_PUPDR_PUPDR0_0;
131       uint8_t PA0_NotPressed = ((GPIOA->IDR & GPIO_IDR_0)!=0);
132       if (PA0_NotPressed == 0){
133           __HAL_TIM_SET_AUTORELOAD(&htim16,500-1);
134       }
135   }
136   /* USER CODE END 3 */
137 }
138
139 /**
140   * @brief System Clock Configuration
141   * @retval None
142   */
143 void SystemClock_Config(void)
144 {
145   LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
146   while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
147   {
148   }
149   LL_RCC_HSI_Enable();
150
151    /* Wait till HSI is ready */
152   while(LL_RCC_HSI_IsReady() != 1)
153   {
154
155   }
156   LL_RCC_HSI_SetCalibTrimming(16);
157   LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
158   LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
159   LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
160
161    /* Wait till System clock is ready */
162   while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
163   {
164
165   }
166   LL_SetSystemCoreClock(8000000);
167
168    /* Update the time base */
169   if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
170   {
171     Error_Handler();
172   }
173 }
174
175 /**
176   * @brief TIM16 Initialization Function
```

```
177   * @param None
178   * @retval None
179   */
180 static void MX_TIM16_Init(void)
181 {
182
183   /* USER CODE BEGIN TIM16_Init 0 */
184
185   /* USER CODE END TIM16_Init 0 */
186
187   /* USER CODE BEGIN TIM16_Init 1 */
188
189   /* USER CODE END TIM16_Init 1 */
190   htim16.Instance = TIM16;
191   htim16.Init.Prescaler = 8000-1;
192   htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
193   htim16.Init.Period = 1000-1;
194   htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
195   htim16.Init.RepetitionCounter = 0;
196   htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
197   if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
198   {
199     Error_Handler();
200   }
201   /* USER CODE BEGIN TIM16_Init 2 */
202   NVIC_EnableIRQ(TIM16_IRQn);
203   /* USER CODE END TIM16_Init 2 */
204
205 }
206
207 /**
208   * @brief GPIO Initialization Function
209   * @param None
210   * @retval None
211   */
212 static void MX_GPIO_Init(void)
213 {
214   LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
215   LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
216 /* USER CODE BEGIN MX_GPIO_Init_1 */
217 /* USER CODE END MX_GPIO_Init_1 */
218
219   /* GPIO Ports Clock Enable */
220   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
221   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
222   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
223
224   /**/
225   LL_GPIO_ResetOutputPin(LED0_GPIO_Port, LED0_Pin);
226
227   /**/
228   LL_GPIO_ResetOutputPin(LED1_GPIO_Port, LED1_Pin);
229
230   /**/
231   LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
232
233   /**/
234   LL_GPIO_ResetOutputPin(LED3_GPIO_Port, LED3_Pin);
235
```

```
236  /**/
237  LL_GPIO_ResetOutputPin(LED4_GPIO_Port, LED4_Pin);
238
239  /**/
240  LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);
241
242  /**/
243  LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);
244
245  /**/
246  LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
247
248  /**/
249  LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);
250
251  /**/
252  LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);
253
254  /**/
255  LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);
256
257  /**/
258  EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
259  EXTI_InitStruct.LineCommand = ENABLE;
260  EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
261  EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
262  LL_EXTI_Init(&EXTI_InitStruct);
263
264  /**/
265  GPIO_InitStruct.Pin = LED0_Pin;
266  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
267  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
268  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
269  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
270  LL_GPIO_Init(LED0_GPIO_Port, &GPIO_InitStruct);
271
272  /**/
273  GPIO_InitStruct.Pin = LED1_Pin;
274  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
275  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
276  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
277  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
278  LL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
279
280  /**/
281  GPIO_InitStruct.Pin = LED2_Pin;
282  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
283  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
284  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
285  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
286  LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
287
288  /**/
289  GPIO_InitStruct.Pin = LED3_Pin;
290  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
291  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
292  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
293  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
294  LL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);
```

```
295
296   /**/
297   GPIO_InitStruct.Pin = LED4_Pin;
298   GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
299   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
300   GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
301   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
302   LL_GPIO_Init(LED4_GPIO_Port, &GPIO_InitStruct);
303
304   /**/
305   GPIO_InitStruct.Pin = LED5_Pin;
306   GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
307   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
308   GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
309   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
310   LL_GPIO_Init(LED5_GPIO_Port, &GPIO_InitStruct);
311
312   /**/
313   GPIO_InitStruct.Pin = LED6_Pin;
314   GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
315   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
316   GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
317   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
318   LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);
319
320   /**/
321   GPIO_InitStruct.Pin = LED7_Pin;
322   GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
323   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
324   GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
325   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
326   LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
327
328 /* USER CODE BEGIN MX_GPIO_Init_2 */
329 /* USER CODE END MX_GPIO_Init_2 */
330 }
331
332 /* USER CODE BEGIN 4 */
333
334 // Initialise SPI
335 static void init_spi(void) {
336
337   // Clock to PB
338   RCC->AHBENR |= RCC_AHBENR_GPIOBEN;    // Enable clock for SPI port
339
340   // Set pin modes
341   GPIOB->MODER |= GPIO_MODER_MODER13_1; // Set pin SCK (PB13) to Alternate Function
342   GPIOB->MODER |= GPIO_MODER_MODER14_1; // Set pin MISO (PB14) to Alternate Function
343   GPIOB->MODER |= GPIO_MODER_MODER15_1; // Set pin MOSI (PB15) to Alternate Function
344   GPIOB->MODER |= GPIO_MODER_MODER12_0; // Set pin CS (PB12) to output push-pull
345   GPIOB->BSRR |= GPIO_BSRR_BS_12;       // Pull CS high
346
347   // Clock enable to SPI
348   RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
349   SPI2->CR1 |= SPI_CR1_BIDIOE;                                  // Enable output
350   SPI2->CR1 |= (SPI_CR1_BR_0 |  SPI_CR1_BR_1);                  // Set Baud to fpclk /
    16
351   SPI2->CR1 |= SPI_CR1_MSTR;                                    // Set to master mode
352   SPI2->CR2 |= SPI_CR2_FRXTH;                                   // Set RX threshold to
```

```
   be 8 bits
353   SPI2->CR2 |= SPI_CR2_SSOE;                                    // Enable slave output
   to work in master mode
354   SPI2->CR2 |= (SPI_CR2_DS_0 | SPI_CR2_DS_1 | SPI_CR2_DS_2);    // Set to 8-bit mode
355   SPI2->CR1 |= SPI_CR1_SPE;                                     // Enable the SPI
   peripheral
356 }
357
358 // Implements a delay in microseconds
359 static void delay(uint32_t delay_in_us) {
360   volatile uint32_t counter = 0;
361   delay_in_us *= 3;
362   for(; counter < delay_in_us; counter++) {
363     __asm("nop");
364     __asm("nop");
365   }
366 }
367
368 // Write to EEPROM address using SPI
369 static void write_to_address(uint16_t address, uint8_t data) {
370
371     uint8_t dummy; // Junk from the DR
372
373     // Set the Write Enable latch
374     GPIOB->BSRR |= GPIO_BSRR_BR_12; // Pull CS low
375     delay(1);
376     *((uint8_t*)(&SPI2->DR)) = WREN;
377     while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
378     dummy = SPI2->DR;
379     GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
380     delay(5000);
381
382     // Send write instruction
383     GPIOB->BSRR |= GPIO_BSRR_BR_12;             // Pull CS low
384     delay(1);
385     *((uint8_t*)(&SPI2->DR)) = WRITE;
386     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
387     dummy = SPI2->DR;
388
389     // Send 16-bit address
390     *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
391     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
392     dummy = SPI2->DR;
393     *((uint8_t*)(&SPI2->DR)) = (address);       // Address LSB
394     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
395     dummy = SPI2->DR;
396
397     // Send the data
398     *((uint8_t*)(&SPI2->DR)) = data;
399     while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
400     dummy = SPI2->DR;
401     GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
402     delay(5000);
403 }
404
405 // Read from EEPROM address using SPI
406 static uint8_t read_from_address(uint16_t address) {
407
408     uint8_t dummy; // Junk from the DR
```

```
409
410     // Send the read instruction
411     GPIOB->BSRR |= GPIO_BSRR_BR_12;             // Pull CS low
412     delay(1);
413     *((uint8_t*)(&SPI2->DR)) = READ;
414     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
415     dummy = SPI2->DR;
416
417     // Send 16-bit address
418     *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
419     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
420     dummy = SPI2->DR;
421     *((uint8_t*)(&SPI2->DR)) = (address);       // Address LSB
422     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
423     dummy = SPI2->DR;
424
425     // Clock in the data
426     *((uint8_t*)(&SPI2->DR)) = 0x42;                // Clock out some junk data
427     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
428     dummy = SPI2->DR;
429     GPIOB->BSRR |= GPIO_BSRR_BS_12;                 // Pull CS high
430     delay(5000);
431
432     return dummy;                                           // Return read data
433 }
434
435 // Timer rolled over
436 void TIM16_IRQHandler(void)
437 {
438     MX_GPIO_Init();
439     // Acknowledge interrupt
440     HAL_TIM_IRQHandler(&htim16);
441
442     // TODO: Change to next LED pattern; output 0x01 if the read SPI data is incorrect
443     if (read_from_address(index) == patterns[index]){
444         GPIOB->ODR |= read_from_address(index);
445     }
446     else{
447         GPIOB-> ODR |= 0x01;
448     }
449     if (index> sizeof(patterns)){
450     index = 0;
451     }
452     else{
453         index++;
454     }
455 }
456
457 /* USER CODE END 4 */
458
459 /**
460   * @brief  This function is executed in case of error occurrence.
461   * @retval None
462   */
463 void Error_Handler(void)
464 {
465   /* USER CODE BEGIN Error_Handler_Debug */
466   /* User can add his own implementation to report the HAL error return state */
467   __disable_irq();
```

```
468   while (1)
469   {
470   }
471   /* USER CODE END Error_Handler_Debug */
472 }
473
474 #ifdef  USE_FULL_ASSERT
475 /**
476   * @brief  Reports the name of the source file and the source line number
477   *         where the assert_param error has occurred.
478   * @param  file: pointer to the source file name
479   * @param  line: assert_param error line source number
480   * @retval None
481   */
482 void assert_failed(uint8_t *file, uint32_t line)
483 {
484   /* USER CODE BEGIN 6 */
485   /* User can add his own implementation to report the file name and line number,
486      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
487   /* USER CODE END 6 */
488 }
489 #endif /* USE_FULL_ASSERT */
490
```