```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"
#include<stdbool.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};
 58 uint16_t j=0;
 59 int k=0;
 60 uint16_t address = 0;
 61 uint8_t button_val=0;
 62 uint8_t arr_value;
 63 bool pressed;
 64
 65
 66 /* USER CODE END PV */
 67
 68 /* Private function prototypes -----------------------------------------------*/
 69 void SystemClock_Config(void);
 70 static void MX_GPIO_Init(void);
 71 static void MX_TIM16_Init(void);
 72 /* USER CODE BEGIN PFP */
 73 void EXTI0_1_IRQHandler(void);
 74 void TIM16_IRQHandler(void);
 75 static void init_spi(void);
 76 static void write_to_address(uint16_t address, uint8_t data);
 77 static uint8_t read_from_address(uint16_t address);
 78 static void delay(uint32_t delay_in_us);
 79 /* USER CODE END PFP */
 80
 81 /* Private user code ---------------------------------------------------------*/
 82 /* USER CODE BEGIN 0 */
 83
 84 /* USER CODE END 0 */
 85
 86 /**
 87   * @brief  The application entry point.
 88   * @retval int
 89   */
 90 int main(void)
 91 {
 92   /* USER CODE BEGIN 1 */
 93   /* USER CODE END 1 */
 94
 95   /* MCU Configuration--------------------------------------------------------*/
 96
 97   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 98   HAL_Init();
 99
100   /* USER CODE BEGIN Init */
101   /* USER CODE END Init */
102
103   /* Configure the system clock */
104   SystemClock_Config();
105
106   /* USER CODE BEGIN SysInit */
107   init_spi();
108   /* USER CODE END SysInit */
109
110   /* Initialize all configured peripherals */
111   MX_GPIO_Init();
112   MX_TIM16_Init();
113   /* USER CODE BEGIN 2 */
```

```
114
115   // TODO: Start timer TIM16
116   HAL_TIM_Base_Start_IT(&htim16);
117   arr_value= TIM16->ARR;
118
119   // TODO: Write all "patterns" to EEPROM using SPI
120 for(uint16_t i=0; i<sizeof(patterns);i++){
121     write_to_address(i, patterns[i]);
122 }
123
124   /* USER CODE END 2 */
125
126   /* Infinite loop */
127   /* USER CODE BEGIN WHILE */
128   while (1)
129   {
130     /* USER CODE END WHILE */
131
132     /* USER CODE BEGIN 3 */
133
134     // TODO: Check button PA0; if pressed, change timer delay
135 button_val=HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
136
137 if(button_val==0){
138 if(pressed==true){
139 pressed=false;
140 }else{
141     pressed = true;
142 }
143 }
144 if(pressed){
145     TIM16->ARR = arr_value/2;
146 }
147 else{
148     TIM16->ARR = arr_value;
149 }
150
151
152   }
153   /* USER CODE END 3 */
154 }
155
156 /**
157   * @brief System Clock Configuration
158   * @retval None
159   */
160 void SystemClock_Config(void)
161 {
162   LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
163   while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
164   {
165   }
166   LL_RCC_HSI_Enable();
167
168    /* Wait till HSI is ready */
169   while(LL_RCC_HSI_IsReady() != 1)
170   {
```

```
171
172   }
173   LL_RCC_HSI_SetCalibTrimming(16);
174   LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
175   LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
176   LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
177
178    /* Wait till System clock is ready */
179   while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
180   {
181
182   }
183   LL_SetSystemCoreClock(8000000);
184
185    /* Update the time base */
186   if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
187   {
188     Error_Handler();
189   }
190 }
191
192 /**
193   * @brief TIM16 Initialization Function
194   * @param None
195   * @retval None
196   */
197 static void MX_TIM16_Init(void)
198 {
199
200   /* USER CODE BEGIN TIM16_Init 0 */
201
202   /* USER CODE END TIM16_Init 0 */
203
204   /* USER CODE BEGIN TIM16_Init 1 */
205
206   /* USER CODE END TIM16_Init 1 */
207   htim16.Instance = TIM16;
208   htim16.Init.Prescaler = 8000-1;
209   htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
210   htim16.Init.Period = 1000-1;
211   htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
212   htim16.Init.RepetitionCounter = 0;
213   htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
214   if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
215   {
216     Error_Handler();
217   }
218   /* USER CODE BEGIN TIM16_Init 2 */
219   NVIC_EnableIRQ(TIM16_IRQn);
220   /* USER CODE END TIM16_Init 2 */
221
222 }
223
224 /**
225   * @brief GPIO Initialization Function
226   * @param None
227   * @retval None
```

```
228  */
229static void MX_GPIO_Init(void)
230{
231  LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
232  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
233/* USER CODE BEGIN MX_GPIO_Init_1 */
234/* USER CODE END MX_GPIO_Init_1 */
235
236  /* GPIO Ports Clock Enable */
237  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
238  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
239  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
240
241  /**/
242  LL_GPIO_ResetOutputPin(LED0_GPIO_Port, LED0_Pin);
243
244  /**/
245  LL_GPIO_ResetOutputPin(LED1_GPIO_Port, LED1_Pin);
246
247  /**/
248  LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
249
250  /**/
251  LL_GPIO_ResetOutputPin(LED3_GPIO_Port, LED3_Pin);
252
253  /**/
254  LL_GPIO_ResetOutputPin(LED4_GPIO_Port, LED4_Pin);
255
256  /**/
257  LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);
258
259  /**/
260  LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);
261
262  /**/
263  LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
264
265  /**/
266  LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);
267
268  /**/
269  LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);
270
271  /**/
272  LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);
273
274  /**/
275  EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
276  EXTI_InitStruct.LineCommand = ENABLE;
277  EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
278  EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
279  LL_EXTI_Init(&EXTI_InitStruct);
280
281  /**/
282  GPIO_InitStruct.Pin = LED0_Pin;
283  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
284  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
```

```
285  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
286  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
287  LL_GPIO_Init(LED0_GPIO_Port, &GPIO_InitStruct);
288
289  /**/
290  GPIO_InitStruct.Pin = LED1_Pin;
291  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
292  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
293  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
294  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
295  LL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
296
297  /**/
298  GPIO_InitStruct.Pin = LED2_Pin;
299  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
300  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
301  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
302  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
303  LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
304
305  /**/
306  GPIO_InitStruct.Pin = LED3_Pin;
307  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
308  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
309  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
310  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
311  LL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);
312
313  /**/
314  GPIO_InitStruct.Pin = LED4_Pin;
315  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
316  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
317  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
318  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
319  LL_GPIO_Init(LED4_GPIO_Port, &GPIO_InitStruct);
320
321  /**/
322  GPIO_InitStruct.Pin = LED5_Pin;
323  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
324  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
325  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
326  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
327  LL_GPIO_Init(LED5_GPIO_Port, &GPIO_InitStruct);
328
329  /**/
330  GPIO_InitStruct.Pin = LED6_Pin;
331  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
332  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
333  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
334  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
335  LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);
336
337  /**/
338  GPIO_InitStruct.Pin = LED7_Pin;
339  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
340  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
341  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
```

```
342   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
343   LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
344
345 /* USER CODE BEGIN MX_GPIO_Init_2 */
346 /* USER CODE END MX_GPIO_Init_2 */
347 }
348
349 /* USER CODE BEGIN 4 */
350
351 // Initialise SPI
352 static void init_spi(void) {
353
354   // Clock to PB
355   RCC->AHBENR |= RCC_AHBENR_GPIOBEN;    // Enable clock for SPI port
356
357   // Set pin modes
358   GPIOB->MODER |= GPIO_MODER_MODER13_1; // Set pin SCK (PB13) to Alternate Function
359   GPIOB->MODER |= GPIO_MODER_MODER14_1; // Set pin MISO (PB14) to Alternate Function
360   GPIOB->MODER |= GPIO_MODER_MODER15_1; // Set pin MOSI (PB15) to Alternate Function
361   GPIOB->MODER |= GPIO_MODER_MODER12_0; // Set pin CS (PB12) to output push-pull
362   GPIOB->BSRR |= GPIO_BSRR_BS_12;       // Pull CS high
363
364   // Clock enable to SPI
365   RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
366   SPI2->CR1 |= SPI_CR1_BIDIOE;                                  // Enable output
367   SPI2->CR1 |= (SPI_CR1_BR_0 |  SPI_CR1_BR_1);                  // Set Baud to fpclk / 16
368   SPI2->CR1 |= SPI_CR1_MSTR;                                    // Set to master mode
369   SPI2->CR2 |= SPI_CR2_FRXTH;                                   // Set RX threshold to be 8
    bits
370   SPI2->CR2 |= SPI_CR2_SSOE;                                    // Enable slave output to work
    in master mode
371   SPI2->CR2 |= (SPI_CR2_DS_0 | SPI_CR2_DS_1 | SPI_CR2_DS_2);    // Set to 8-bit mode
372   SPI2->CR1 |= SPI_CR1_SPE;                                     // Enable the SPI peripheral
373 }
374
375 // Implements a delay in microseconds
376 static void delay(uint32_t delay_in_us) {
377   volatile uint32_t counter = 0;
378   delay_in_us *= 3;
379   for(; counter < delay_in_us; counter++) {
380     __asm("nop");
381     __asm("nop");
382   }
383 }
384
385 // Write to EEPROM address using SPI
386 static void write_to_address(uint16_t address, uint8_t data) {
387
388     uint8_t dummy; // Junk from the DR
389
390     // Set the Write Enable latch
391     GPIOB->BSRR |= GPIO_BSRR_BR_12; // Pull CS low
392     delay(1);
393     *((uint8_t*)(&SPI2->DR)) = WREN;
394     while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
395     dummy = SPI2->DR;
396     GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
```

```
397     delay(5000);
398
399     // Send write instruction
400     GPIOB->BSRR |= GPIO_BSRR_BR_12;             // Pull CS low
401     delay(1);
402     *((uint8_t*)(&SPI2->DR)) = WRITE;
403     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
404     dummy = SPI2->DR;
405
406     // Send 16-bit address
407     *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
408     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
409     dummy = SPI2->DR;
410     *((uint8_t*)(&SPI2->DR)) = (address);       // Address LSB
411     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
412     dummy = SPI2->DR;
413
414     // Send the data
415     *((uint8_t*)(&SPI2->DR)) = data;
416     while ((SPI2->SR & SPI_SR_RXNE) == 0); // Hang while RX is empty
417     dummy = SPI2->DR;
418     GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
419     delay(5000);
420 }
421
422 // Read from EEPROM address using SPI
423 static uint8_t read_from_address(uint16_t address) {
424
425     uint8_t dummy; // Junk from the DR
426
427     // Send the read instruction
428     GPIOB->BSRR |= GPIO_BSRR_BR_12;             // Pull CS low
429     delay(1);
430     *((uint8_t*)(&SPI2->DR)) = READ;
431     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
432     dummy = SPI2->DR;
433
434     // Send 16-bit address
435     *((uint8_t*)(&SPI2->DR)) = (address >> 8);  // Address MSB
436     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
437     dummy = SPI2->DR;
438     *((uint8_t*)(&SPI2->DR)) = (address);       // Address LSB
439     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
440     dummy = SPI2->DR;
441
442     // Clock in the data
443     *((uint8_t*)(&SPI2->DR)) = 0x42;                // Clock out some junk data
444     while ((SPI2->SR & SPI_SR_RXNE) == 0);      // Hang while RX is empty
445     dummy = SPI2->DR;
446     GPIOB->BSRR |= GPIO_BSRR_BS_12;                 // Pull CS high
447     delay(5000);
448
449     return dummy;                                           // Return read data
450 }
451
452 // Timer rolled over
453 void TIM16_IRQHandler(void)
```

```
454{
455    // Acknowledge interrupt
456    HAL_TIM_IRQHandler(&htim16);
457    GPIOB->ODR &=0x00;
458    // TODO: Change to next LED pattern; output 0x01 if the read SPI data is incorrect
459    uint8_t val = read_from_address(j);
460    if(val!=patterns[j]){
461        GPIOB -> ODR |= 0b00000001;
462    }
463    else{
464        GPIOB -> ODR |= val;
465    }
466    j++;
467    if(j==sizeof(patterns)){
468        j=0;
469    }
470
471}
472
473/* USER CODE END 4 */
474
475/**
476  * @brief  This function is executed in case of error occurrence.
477  * @retval None
478  */
479void Error_Handler(void)
480{
481  /* USER CODE BEGIN Error_Handler_Debug */
482  /* User can add his own implementation to report the HAL error return state */
483  __disable_irq();
484  while (1)
485  {
486  }
487  /* USER CODE END Error_Handler_Debug */
488}
489
490#ifdef  USE_FULL_ASSERT
491/**
492  * @brief  Reports the name of the source file and the source line number
493  *         where the assert_param error has occurred.
494  * @param  file: pointer to the source file name
495  * @param  line: assert_param error line source number
496  * @retval None
497  */
498void assert_failed(uint8_t *file, uint32_t line)
499{
500  /* USER CODE BEGIN 6 */
501  /* User can add his own implementation to report the file name and line number,
502     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
503  /* USER CODE END 6 */
504}
505#endif /* USE_FULL_ASSERT */
506
```