```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 "lcd_stm32f0.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
```

```
 58 static uint16_t positionAddress = 0;
 59 static uint8_t patterns[] = { 0b10101010, // 10101010 in binary
 60         0b01010101, // 01010101 in binary
 61         0b11001100, // 11001100 in binary
 62         0b00110011, // 00110011 in binary
 63         0b11110000, // 11110000 in binary
 64         0b00001111  // 00001111 in binary
 65         };
 66
 67
 68 /* USER CODE END PV */
 69
 70 /* Private function prototypes -----------------------------------------------*/
 71 void SystemClock_Config(void);
 72 static void MX_GPIO_Init(void);
 73 static void MX_TIM16_Init(void);
 74 /* USER CODE BEGIN PFP */
 75 void EXTI0_1_IRQHandler(void);
 76 void TIM16_IRQHandler(void);
 77 static void init_spi(void);
 78 static void write_to_address(uint16_t address, uint8_t data);
 79 static uint8_t read_from_address(uint16_t address);
 80 static void delay(uint32_t delay_in_us);
 81 /* USER CODE END PFP */
 82
 83 /* Private user code ---------------------------------------------------------*/
 84 /* USER CODE BEGIN 0 */
 85
 86 /* USER CODE END 0 */
 87
 88 /**
 89  * @brief  The application entry point.
 90  * @retval int
 91  */
 92 int main(void) {
 93     /* USER CODE BEGIN 1 */
 94     /* USER CODE END 1 */
 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
102     /* USER CODE END Init */
103
104     /* Configure the system clock */
105     SystemClock_Config();
106
107     /* USER CODE BEGIN SysInit */
108     init_spi();
109     /* USER CODE END SysInit */
110
111     /* Initialize all configured peripherals */
112     MX_GPIO_Init();
113     MX_TIM16_Init();
114     /* USER CODE BEGIN 2 */
```

```
115
116     // TODO: Start timer TIM16
117     HAL_TIM_Base_Start_IT(&htim16);
118
119     // TODO: Write all "patterns" to EEPROM using SPI
120     uint16_t eepAddress = 0;
121
122     for (uint16_t i = 0; i < sizeof(patterns); i++) {
123         write_to_address(eepAddress++, patterns[i]);
124     }
125
126     uint32_t longPeriod = 1000;
127     uint32_t shortPeriod = 500;
128     uint32_t currentPeriod = longPeriod;
129
130     /* USER CODE END 2 */
131
132     /* Infinite loop */
133     /* USER CODE BEGIN WHILE */
134     while (1) {
135         /* USER CODE END WHILE */
136
137         /* USER CODE BEGIN 3 */
138
139         // TODO: Check button PA0; if pressed, change timer delay
140         if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 0) {
141             // When the button is pressed, it alternates the delay
142             if (currentPeriod == shortPeriod) {
143                 currentPeriod = longPeriod;
144             } else if(currentPeriod == longPeriod) {
145                 currentPeriod = shortPeriod;
146             }
147
148             __HAL_TIM_SET_AUTORELOAD(&htim16, currentPeriod);
149         }
150
151         HAL_Delay(1);
152
153     }
154     /* USER CODE END 3 */
155 }
156
157 /**
158  * @brief System Clock Configuration
159  * @retval None
160  */
161
162 void SystemClock_Config(void) {
163     LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
164     while (LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0) {
165     }
166     LL_RCC_HSI_Enable();
167
168     /* Wait till HSI is ready */
169     while (LL_RCC_HSI_IsReady() != 1) {
170
171     }
```

```
172     LL_RCC_HSI_SetCalibTrimming(16);
173     LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
174     LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
175     LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
176
177     /* Wait till System clock is ready */
178     while (LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI) {
179
180     }
181     LL_SetSystemCoreClock(8000000);
182
183     /* Update the time base */
184     if (HAL_InitTick(TICK_INT_PRIORITY) != HAL_OK) {
185         Error_Handler();
186     }
187 }
188
189 /**
190  * @brief TIM16 Initialization Function
191  * @param None
192  * @retval None
193  */
194 static void MX_TIM16_Init(void) {
195
196     /* USER CODE BEGIN TIM16_Init 0 */
197
198     /* USER CODE END TIM16_Init 0 */
199
200     /* USER CODE BEGIN TIM16_Init 1 */
201
202     /* USER CODE END TIM16_Init 1 */
203     htim16.Instance = TIM16;
204     htim16.Init.Prescaler = 8000 - 1;
205     htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
206     htim16.Init.Period = 1000 - 1;
207     htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
208     htim16.Init.RepetitionCounter = 0;
209     htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
210     if (HAL_TIM_Base_Init(&htim16) != HAL_OK) {
211         Error_Handler();
212     }
213     /* USER CODE BEGIN TIM16_Init 2 */
214     NVIC_EnableIRQ(TIM16_IRQn);
215     /* USER CODE END TIM16_Init 2 */
216
217 }
218
219 /**
220  * @brief GPIO Initialization Function
221  * @param None
222  * @retval None
223  */
224 static void MX_GPIO_Init(void) {
225     LL_EXTI_InitTypeDef EXTI_InitStruct = { 0 };
226     LL_GPIO_InitTypeDef GPIO_InitStruct = { 0 };
227     /* USER CODE BEGIN MX_GPIO_Init_1 */
228     /* USER CODE END MX_GPIO_Init_1 */
```

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

```
286     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
287     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
288     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
289     LL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
290
291     /**/
292     GPIO_InitStruct.Pin = LED2_Pin;
293     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
294     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
295     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
296     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
297     LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
298
299     /**/
300     GPIO_InitStruct.Pin = LED3_Pin;
301     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
302     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
303     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
304     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
305     LL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);
306
307     /**/
308     GPIO_InitStruct.Pin = LED4_Pin;
309     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
310     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
311     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
312     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
313     LL_GPIO_Init(LED4_GPIO_Port, &GPIO_InitStruct);
314
315     /**/
316     GPIO_InitStruct.Pin = LED5_Pin;
317     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
318     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
319     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
320     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
321     LL_GPIO_Init(LED5_GPIO_Port, &GPIO_InitStruct);
322
323     /**/
324     GPIO_InitStruct.Pin = LED6_Pin;
325     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
326     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
327     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
328     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
329     LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);
330
331     /**/
332     GPIO_InitStruct.Pin = LED7_Pin;
333     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
334     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
335     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
336     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
337     LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
338
339     /* USER CODE BEGIN MX_GPIO_Init_2 */
340     /* USER CODE END MX_GPIO_Init_2 */
341 }
342
```

```c
/* 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
```

```
400     dummy = SPI2->DR;
401
402     // Send 16-bit address
403     *((uint8_t*) (&SPI2->DR)) = (address >> 8);     // Address MSB
404     while ((SPI2->SR & SPI_SR_RXNE) == 0)
405         ;       // Hang while RX is empty
406     dummy = SPI2->DR;
407     *((uint8_t*) (&SPI2->DR)) = (address);      // Address LSB
408     while ((SPI2->SR & SPI_SR_RXNE) == 0)
409         ;       // Hang while RX is empty
410     dummy = SPI2->DR;
411
412     // Send the data
413     *((uint8_t*) (&SPI2->DR)) = data;
414     while ((SPI2->SR & SPI_SR_RXNE) == 0)
415         ; // Hang while RX is empty
416     dummy = SPI2->DR;
417     GPIOB->BSRR |= GPIO_BSRR_BS_12; // Pull CS high
418     delay(5000);
419 }
420
421 // Read from EEPROM address using SPI
422 static uint8_t read_from_address(uint16_t address) {
423
424     uint8_t dummy; // Junk from the DR
425
426     // Send the read instruction
427     GPIOB->BSRR |= GPIO_BSRR_BR_12;             // Pull CS low
428     delay(1);
429     *((uint8_t*) (&SPI2->DR)) = READ;
430     while ((SPI2->SR & SPI_SR_RXNE) == 0)
431         ;       // 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)
437         ;       // Hang while RX is empty
438     dummy = SPI2->DR;
439     *((uint8_t*) (&SPI2->DR)) = (address);      // Address LSB
440     while ((SPI2->SR & SPI_SR_RXNE) == 0)
441         ;       // Hang while RX is empty
442     dummy = SPI2->DR;
443
444     // Clock in the data
445     *((uint8_t*) (&SPI2->DR)) = 0x42;               // Clock out some junk data
446     while ((SPI2->SR & SPI_SR_RXNE) == 0)
447         ;       // Hang while RX is empty
448     dummy = SPI2->DR;
449     GPIOB->BSRR |= GPIO_BSRR_BS_12;                 // Pull CS high
450     delay(5000);
451
452     return dummy;                                           // Return read data
453 }
454
455 // Timer rolled over
456 void TIM16_IRQHandler(void) {
```

```c

    struct LED {
        GPIO_TypeDef *GPIO_Port;
        uint16_t GPIO_Pin;
    };
    // Assigning LED pin numbers to structure - easier to use
    struct LED ledAssign[] = { { GPIOB, GPIO_PIN_0 }, { GPIOB, GPIO_PIN_1 }, {
            GPIOB, GPIO_PIN_2 }, { GPIOB, GPIO_PIN_3 }, { GPIOB, GPIO_PIN_4 }, {
            GPIOB, GPIO_PIN_5 }, { GPIOB, GPIO_PIN_6 }, { GPIOB, GPIO_PIN_7 } };

    // Acknowledge interrupt
    HAL_TIM_IRQHandler(&htim16);

    // Assigning address
    uint8_t bValue = read_from_address(positionAddress);

    for (int i = 0; i < 8; i++) {
        if (bValue & (1 << i)) {
            // Turning on the selected LED
            LL_GPIO_SetOutputPin(ledAssign[i].GPIO_Port, ledAssign[i].GPIO_Pin);
        } else {
            // Turning off the selected LED
            LL_GPIO_ResetOutputPin(ledAssign[i].GPIO_Port,
                    ledAssign[i].GPIO_Pin);
        }
    }
    // Increment position
    positionAddress++;

    if (positionAddress >= sizeof(patterns)) {
        positionAddress = 0;
    }

    // TODO: Change to next LED pattern; output 0x01 if the read SPI data is incorrect

    // Assign default value if incorrect SPI address
    uint8_t failsafeValue = patterns[positionAddress];
    if (bValue != failsafeValue) {
        bValue = 0b00000001;
    }
}

/* USER CODE END 4 */

/**
 * @brief  This function is executed in case of error occurrence.
 * @retval None
 */
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 */
}

```

```
514 #ifdef  USE_FULL_ASSERT
515 /**
516   * @brief  Reports the name of the source file and the source line number
517   *         where the assert_param error has occurred.
518   * @param  file: pointer to the source file name
519   * @param  line: assert_param error line source number
520   * @retval None
521   */
522 void assert_failed(uint8_t *file, uint32_t line)
523 {
524   /* USER CODE BEGIN 6 */
525   /* User can add his own implementation to report the file name and line number,
526      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
527   /* USER CODE END 6 */
528 }
529 #endif /* USE_FULL_ASSERT */
530
```