```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 <stdlib.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* 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 input variables
#define SW0_PIN GPIO_IDR_0
#define SW1_PIN GPIO_IDR_1
#define SW2_PIN GPIO_IDR_2
#define SW3_PIN GPIO_IDR_3

#define LED0_PIN GPIO_ODR_4
#define LED1_PIN GPIO_ODR_5
#define LED2_PIN GPIO_ODR_6
#define LED3_PIN GPIO_ODR_7

// Mode definitions
typedef enum {
    MODE_OFF = 0,
    MODE_1_BACK_FORTH = 1,
    MODE_2_INVERSE_BACK_FORTH = 2,
    MODE_3_SPARKLE = 3
} led_mode_t;

// Global variables
volatile led_mode_t current_mode = MODE_OFF;
volatile uint8_t led_position = 0;
volatile int8_t direction = 1;  // 1 for forward, -1 for backward
volatile uint8_t fast_mode = 0; // 0 for 1s, 1 for 0.5s
volatile uint8_t sparkle_state = 0; // For sparkle mode state machine
volatile uint8_t sparkle_pattern = 0;
volatile uint8_t sparkle_delay_counter = 0;
volatile uint8_t sparkle_off_counter = 0;
volatile uint8_t leds_to_turn_off = 0;

// Button state tracking for debouncing
uint8_t prev_button_state[4] = {1, 1, 1, 1}; // Pull-up means 1 when not pressed

/* 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 TIM16_IRQHandler(void);
void set_tim16_delay_ms(uint16_t ms)
{

    if (ms < 1) ms = 1;
    __HAL_TIM_DISABLE(&htim16);
    __HAL_TIM_SET_AUTORELOAD(&htim16, ms - 1);// Update ARR register directly
    __HAL_TIM_SET_COUNTER(&htim16, 0); // Reset counter
    __HAL_TIM_ENABLE(&htim16); // Re-enable timer
}

void update_led_pattern(void);
void set_led_pattern(uint8_t pattern);
uint8_t get_random_byte(void);
/* 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 */
  /* 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);

  // Initialize random seed
  srand(HAL_GetTick());

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    // TODO: Check pushbuttons to change timer delay

    // Read current button states
    uint8_t current_button_state[4];
    current_button_state[0] = (GPIOA->IDR & GPIO_IDR_0) ? 1 : 0;
    current_button_state[1] = (GPIOA->IDR & GPIO_IDR_1) ? 1 : 0;
    current_button_state[2] = (GPIOA->IDR & GPIO_IDR_2) ? 1 : 0;
    current_button_state[3] = (GPIOA->IDR & GPIO_IDR_3) ? 1 : 0;

    // Check for button presses (falling edge detection)

    // PA0 - Toggle timing (1s <-> 0.5s)
    if (prev_button_state[0] == 1 && current_button_state[0] == 0) {
        fast_mode = !fast_mode;
        if (fast_mode) {
            htim16.Init.Period = 500-1;   // 0.5 second
        } else {
            htim16.Init.Period = 1000-1;  // 1 second
        }
        HAL_TIM_Base_Init(&htim16);
    }

    // PA1 - Mode 1: Back/forth
    if (prev_button_state[1] == 1 && current_button_state[1] == 0) {
        current_mode = MODE_1_BACK_FORTH;
        led_position = 0;
        direction = 1;
    }

    // PA2 - Mode 2: Inverse back/forth
    if (prev_button_state[2] == 1 && current_button_state[2] == 0) {
        current_mode = MODE_2_INVERSE_BACK_FORTH;
        led_position = 0;
        direction = 1;
    }

    // PA3 - Mode 3: Sparkle
    if (prev_button_state[3] == 1 && current_button_state[3] == 0) {
        current_mode = MODE_3_SPARKLE;
        sparkle_state = 0;
        sparkle_delay_counter = 0; //
        sparkle_off_counter = 0;
    }

    // Update previous button states
    for (int i = 0; i < 4; i++) {
        prev_button_state[i] = current_button_state[i];
    }

    HAL_Delay(10); // Small delay for debouncing

  }
  /* 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();
  }
```

```c
}

/**
  * @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);
  /* USER CODE END TIM16_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
  LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);

  /**/
  LL_GPIO_ResetOutputPin(LED0_GPIO_Port, LED0_Pin);

  /**/
  LL_GPIO_ResetOutputPin(LED1_GPIO_Port, LED1_Pin);

  /**/
  LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);

  /**/
  LL_GPIO_ResetOutputPin(LED3_GPIO_Port, LED3_Pin);
```

```

/**/
LL_GPIO_ResetOutputPin(LED4_GPIO_Port, LED4_Pin);

/**/
LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);

/**/
LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);

/**/
LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);

/**/
GPIO_InitStruct.Pin = Button0_Pin;
GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
LL_GPIO_Init(Button0_GPIO_Port, &GPIO_InitStruct);

/**/
GPIO_InitStruct.Pin = Button1_Pin;
GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
LL_GPIO_Init(Button1_GPIO_Port, &GPIO_InitStruct);

/**/
GPIO_InitStruct.Pin = Button2_Pin;
GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
LL_GPIO_Init(Button2_GPIO_Port, &GPIO_InitStruct);

/**/
GPIO_InitStruct.Pin = Button3_Pin;
GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
LL_GPIO_Init(Button3_GPIO_Port, &GPIO_InitStruct);

/**/
GPIO_InitStruct.Pin = LED0_Pin;
GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
LL_GPIO_Init(LED0_GPIO_Port, &GPIO_InitStruct);

/**/
GPIO_InitStruct.Pin = LED1_Pin;
GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
LL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);

/**/
GPIO_InitStruct.Pin = LED2_Pin;
GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);

```

```c
  /**/
  GPIO_InitStruct.Pin = LED3_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED4_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(LED4_GPIO_Port, &GPIO_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED5_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(LED5_GPIO_Port, &GPIO_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED6_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);

  /**/
  GPIO_InitStruct.Pin = LED7_Pin;
  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
  LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void TIM16_IRQHandler(void)
{
 // Acknowledge interrupt
 HAL_TIM_IRQHandler(&htim16);

 // TODO: Change LED pattern
 update_led_pattern();
}

void update_led_pattern(void)
{
    switch (current_mode)
    {
        case MODE_OFF:
            set_led_pattern(0x00); // All LEDs off
            break;

        case MODE_1_BACK_FORTH:
            // Single LED cycling back and forth
            set_led_pattern(1 << led_position);

            // Update position for next interrupt
            led_position += direction;
            if (led_position > 7) {
                direction = -1;
                led_position = 7; // Skip repeating LED7
            } else if (led_position < 0) {
                direction = 1;
                led_position = 0; // Skip repeating LED0
            }
            break;

        case MODE_2_INVERSE_BACK_FORTH:
            // All LEDs on except one, cycling back and forth
            set_led_pattern(0xFF ^ (1 << led_position));

            // Update position for next interrupt (same logic as Mode 1)
            led_position += direction;
            if (led_position > 7) {
                direction = -1;
                led_position = 7;
            } else if (led_position < 0) {
                direction = 1;
                led_position = 0;
            }
            break;

        case MODE_3_SPARKLE:
            switch (sparkle_state) {
                case 0: // All LEDs off → generate new random pattern
                    sparkle_pattern = get_random_byte();       // 0–255
                    leds_to_turn_off = sparkle_pattern;
                    set_led_pattern(sparkle_pattern);

                    set_tim16_delay_ms((rand() % 1401) + 100); // Hold for 100–1500 ms
                    sparkle_state = 1;
                    break;

                case 1: // Done holding, now begin turning LEDs off
                    sparkle_state = 2;
                    break;

                case 2: // Turn off one LED at a time
                    if (leds_to_turn_off != 0) {
                        uint8_t index = 0;
                        uint8_t mask = leds_to_turn_off;
                        while (((mask & 1) == 0) && index < 8) {
                            mask >>= 1;
                            index++;
                        }

                        if (index < 8) {
                            sparkle_pattern &= ~(1 << index);
                            leds_to_turn_off &= ~(1 << index);
                            set_led_pattern(sparkle_pattern);
                        }

                        // Set delay
                        set_tim16_delay_ms(100); // 100
                        sparkle_state = 3;
                    } else {
                        sparkle_state = 0; // Restart
                    }
                    break;

                case 3: // Done waiting → go back to turn off next LED
                    sparkle_state = 2;
                    break;
            }
            break;



    }

}

void set_led_pattern(uint8_t pattern)
{
    // Assuming LEDs are connected to GPIOB pins 0-7
    // Clear all LEDs first
    GPIOB->ODR &= 0xFF00;
    // Set the pattern
    GPIOB->ODR |= pattern;
}

uint8_t get_random_byte(void)
{
    return (uint8_t)(rand() & 0xFF);
}

/* 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 */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

```