```
/* 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 <stdbool.h>
#include "stm32f0xx.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* A simple enum for the three LED modes (plus "none" after reset). */
typedef enum {
  MODE_NONE = 0,
  MODE1_BACK_FORTH = 1,   // PA1
  MODE2_INV_BACK_FORTH = 2, // PA2
  MODE3_SPARKLE = 3      // PA3
} LedMode;

/* Sparkle mode phases (state machine that runs on each timer interrupt). */
typedef enum {
  SPARKLE_IDLE = 0,    // choose a new random pattern and a random hold time
  SPARKLE_HOLD,       // keep the pattern for the chosen hold time
  SPARKLE_TURNOFF     // turn off lit LEDs one-by-one with small random gaps
} SparklePhase;
/* USER CODE END PTD */

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

#define TIM_MS_TO_ARR(ms)  ((uint16_t)((ms) - 1U))

/* Two speeds required by the prac: 1000 ms and 500 ms, applied via ARR updates. */
#define ARR_SLOW  TIM_MS_TO_ARR(1000U)  /* ~1.0 s period */
#define ARR_FAST  TIM_MS_TO_ARR(500U)   /* ~0.5 s period */

/* Active-low push-buttons (pull-ups enabled): pressed == 0. */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim16;

/* USER CODE BEGIN PV */
/* --- Required state (Task 1) ---------------------------------------------
 *  - Current mode
 *  - Scanner index and direction for modes 1 & 2
 *  - Speed toggle state for PA0 (switches ARR 1000ms <-> 500ms)
 *  - Sparkle state variables & very small PRNG
 */
volatile LedMode     g_mode = MODE_NONE;
volatile uint8_t     g_idx = 0;       /* LED index 0..7 for back/forth */
volatile int8_t      g_dir = +1;      /* +1 towards LED7, -1 towards LED0 */

volatile bool        g_fast = false;   /* false=1s (ARR_SLOW), true=0.5s (ARR_FAST) */

/* Edge-detect for PA0 speed button so we toggle only once per press. */
static bool          btn0_armed = true;

/* ---------- Sparkle mode (mode 3) ---------- */
volatile SparklePhase sp_phase = SPARKLE_IDLE;
volatile uint8_t     sp_pattern = 0x00;   /* current 8-bit LED pattern */
volatile uint8_t     sp_turnoff_i = 0;    /* which bit we’re considering to turn off */

/* Simple 16-bit Galois LFSR PRNG (no <stdlib.h>), non-zero seed. */
static uint16_t lfsr = 0xA5A5u;
static inline uint16_t prng16(void) {
  /* taps: 16,14,13,11 => poly 0xB400 (per common LFSR examples) */
  uint16_t lsb = lfsr & 1u;
  lfsr >>= 1;
  if (lsb) lfsr ^= 0xB400u;
  return lfsr;
}
static inline uint8_t urand8(void)      { return (uint8_t)(prng16() & 0xFFu); }
static inline uint16_t urand_range(uint16_t min_incl, uint16_t max_incl) {
  /* uniform in [min,max], max<=65535, assume max>=min */
  uint16_t span = (uint16_t)(max_incl - min_incl + 1u);
  return (uint16_t)(min_incl + (prng16() % span));
}

/* One helper to write all eight LEDs at once from an 8-bit pattern (bit0 -> LED0). */
static inline void set_leds(uint8_t pat) {
  HAL_GPIO_WritePin(LED0_GPIO_Port, LED0_Pin, (pat & (1u<<0)) ? GPIO_PIN_SET : GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, (pat & (1u<<1)) ? GPIO_PIN_SET : GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, (pat & (1u<<2)) ? GPIO_PIN_SET : GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, (pat & (1u<<3)) ? GPIO_PIN_SET : GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, (pat & (1u<<4)) ? GPIO_PIN_SET : GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED5_GPIO_Port, LED5_Pin, (pat & (1u<<5)) ? GPIO_PIN_SET : GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED6_GPIO_Port, LED6_Pin, (pat & (1u<<6)) ? GPIO_PIN_SET : GPIO_PIN_RESET);
  HAL_GPIO_WritePin(LED7_GPIO_Port, LED7_Pin, (pat & (1u<<7)) ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

/* Keep your variables from the starter; they’re no longer needed but harmless if used elsewhere. */
bool bSpeedFlag = false;
bool bFlag1 = false, bFlag2 = false, bFlag3 = false, bIndexFlag = false;
int  j = 0;
uint8_t led_state = 0b00000000;
/* 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);
/* 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 */
/* Seed PRNG with something not-constant (ticks since reset). */
lfsr ^= (uint16_t)HAL_GetTick();

/* Start TIM16 in interrupt mode (Task 2). */
HAL_TIM_Base_Start_IT(&htim16);

/* Ensure all LEDs are OFF on startup (spec). */
set_leds(0x00);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* ---- Task 4: PA0 toggles timer delay by changing TIM16 ARR between
   * 1000 ms and 500 ms. We do edge detection so a long press doesn't
   * flip repeatedly. Note: Buttons are pull-up, so pressed == 0. ----
   *
   * We *don’t* touch ARR during Mode 3 because that mode takes full
   * control of its timing (random 100..1500ms then ~random(100)ms steps).
   */
  GPIO_PinState b0 = HAL_GPIO_ReadPin(Button0_GPIO_Port, Button0_Pin);
  if (g_mode != MODE3_SPARKLE) {
   if ((b0 == GPIO_PIN_RESET) && btn0_armed) {
    btn0_armed = false;        /* consume this press */
    g_fast = !g_fast;          /* toggle speed */

    uint16_t new_arr = g_fast ? ARR_FAST : ARR_SLOW;
    __HAL_TIM_SET_AUTORELOAD(&htim16, new_arr);
    __HAL_TIM_SET_COUNTER(&htim16, 0); /* restart period boundary */
   } else if (b0 == GPIO_PIN_SET) {
    btn0_armed = true;         /* re-arm on release */
   }
  }
 }
  /* 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();
  }
}

/**
  * @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);

/**/
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 */
/* Task 3: On each timer interrupt, show the LED pattern for the current mode.
 * Also look for PA1/PA2/PA3 to switch modes.
 *
 * Notes:
 * - Buttons are polled here ~every period; the prac warns misses may occur
 *   (bounce / sampling window) — that’s acceptable for this exercise. :contentReference[oaicite:0]{index=0}
 */
void TIM16_IRQHandler(void)
{
        // Acknowledge interrupt
  HAL_TIM_IRQHandler(&htim16);

  /* -------- Mode selection (active-low buttons on PA1..PA3) -------- */
  /* PA1 -> Mode 1 (back/forth one LED) */
  if (HAL_GPIO_ReadPin(Button1_GPIO_Port, Button1_Pin) == GPIO_PIN_RESET) {
    g_mode = MODE1_BACK_FORTH;
    g_idx = 0; g_dir = +1;
    /* Restore current speed ARR; mode 3 may have changed ARR randomly */
    __HAL_TIM_SET_AUTORELOAD(&htim16, g_fast ? ARR_FAST : ARR_SLOW);
    __HAL_TIM_SET_COUNTER(&htim16, 0);
    set_leds(0x00); /* start clean */
  }
  /* PA2 -> Mode 2 (inverse back/forth) */
  if (HAL_GPIO_ReadPin(Button2_GPIO_Port, Button2_Pin) == GPIO_PIN_RESET) {
    g_mode = MODE2_INV_BACK_FORTH;
    g_idx = 0; g_dir = +1;
    __HAL_TIM_SET_AUTORELOAD(&htim16, g_fast ? ARR_FAST : ARR_SLOW);
    __HAL_TIM_SET_COUNTER(&htim16, 0);
    set_leds(0xFF); /* all on initially; pattern logic will override on next line */
  }
  /* PA3 -> Mode 3 (sparkle) */
  if (HAL_GPIO_ReadPin(Button3_GPIO_Port, Button3_Pin) == GPIO_PIN_RESET) {
    g_mode = MODE3_SPARKLE;
    sp_phase = SPARKLE_IDLE;
    sp_turnoff_i = 0;
    set_leds(0x00);
    /* ARR will be controlled by the sparkle state machine below. */
  }

  /* ------------- Show/update pattern for the active mode ------------- */
  switch (g_mode) {

    case MODE1_BACK_FORTH:
    {
      /* Back/forth: 00000001 -> ... -> 10000000 -> ... -> 00000001 ...
       * Critical detail: when we hit an end, immediately reverse direction
       * so we *don’t* repeat the end LED on consecutive periods (per spec).
:contentReference[oaicite:1]{index=1}
       */
      uint8_t pattern = (uint8_t)(1u << g_idx);
      set_leds(pattern);

      /* Advance for next period */
      g_idx = (uint8_t)(g_idx + g_dir);
      if (g_idx >= 7) { g_idx = 7; g_dir = -1; }
      else if (g_idx == 0) { g_dir = +1; }
    }
    break;

    case MODE2_INV_BACK_FORTH:
    {
      /* Inverse back/forth: all on except one “hole” that scans. */
      uint8_t inv_pattern = (uint8_t)(~(1u << g_idx)) & 0xFFu;
      set_leds(inv_pattern);

      /* Same edge behavior as Mode 1. */
      g_idx = (uint8_t)(g_idx + g_dir);
      if (g_idx >= 7) { g_idx = 7; g_dir = -1; }
      else if (g_idx == 0) { g_dir = +1; }
    }
    break;

    case MODE3_SPARKLE:
    {
      /* State machine:
       *  IDLE    -> choose new random pattern & hold time 100..1500 ms
       *  HOLD    -> after hold, start turning off lit LEDs one-by-one
       *  TURNOFF -> at each tick, clear next ‘1’ bit; delay random(1..100) ms
       *          until all ones cleared, then back to IDLE.
       * Per the prac description. :contentReference[oaicite:2]{index=2}
       */
      switch (sp_phase) {

        case SPARKLE_IDLE:
        {
          sp_pattern = urand8();        /* random 0..255 */
```

```
 set_leds(sp_pattern);

 uint16_t hold_ms = urand_range(100u, 1500u);
 __HAL_TIM_SET_AUTORELOAD(&htim16, TIM_MS_TO_ARR(hold_ms));
 __HAL_TIM_SET_COUNTER(&htim16, 0);

 sp_turnoff_i = 0;
 sp_phase = SPARKLE_HOLD;
}
break;

case SPARKLE_HOLD:
{
 /* Time to start turning off LEDs one-by-one */
 sp_phase = SPARKLE_TURNOFF;

 /* Schedule first short random gap 1..100 ms */
 uint16_t gap = urand_range(1u, 100u);
 __HAL_TIM_SET_AUTORELOAD(&htim16, TIM_MS_TO_ARR(gap));
 __HAL_TIM_SET_COUNTER(&htim16, 0);
}
break;

case SPARKLE_TURNOFF:
{
 /* Find the next bit that is currently ON and clear it. */
 while (sp_turnoff_i < 8 && ((sp_pattern & (1u << sp_turnoff_i)) == 0)) {
  sp_turnoff_i++;
 }

 if (sp_turnoff_i < 8) {
      /* Turn this LED off, keep others as-is. */
      sp_pattern = (uint8_t)(sp_pattern & (uint8_t)~(1u << sp_turnoff_i));
      set_leds(sp_pattern);
      sp_turnoff_i++;

      /* Keep turning off with small random delays 1..100 ms. */
      uint16_t gap = urand_range(1u, 100u);
      __HAL_TIM_SET_AUTORELOAD(&htim16, TIM_MS_TO_ARR(gap));
      __HAL_TIM_SET_COUNTER(&htim16, 0);
     } else {
      /* All ones are cleared -> next sparkle. */
      sp_phase = SPARKLE_IDLE;
      /* Choose next on the *next* tick (we don’t change ARR here). */
     }
    }
    break;
   }
  }
  break;

  case MODE_NONE:
  default:
   /* Keep LEDs off until a mode is chosen. */
   set_leds(0x00);
   break;
 }
}


/* 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) */

  (void)file; (void)line;
  /* USER CODE END 6 */


}
#endif /* USE_FULL_ASSERT */
```