

```

1/* USER CODE BEGIN Header */
2/**
3 *****
4 * @file           : main.c
5 * @brief          : Main program body
6 *****
7 * @attention
8 *
9 * Copyright (c) 2023 STMicroelectronics.
10 * All rights reserved.
11 *
12 * This software is licensed under terms that can be found in the LICENSE file
13 * in the root directory of this software component.
14 * If no LICENSE file comes with this software, it is provided AS-IS.
15 *
16 *****
17 */
18/* USER CODE END Header */
19/* Includes -----*/
20#include "main.h"
21
22/* Private includes -----*/
23/* USER CODE BEGIN Includes */
24#include <stdint.h>
25#include "stm32f0xx.h"
26#include <stdlib.h>
27/* USER CODE END Includes */
28
29/* Private typedef -----*/
30/* USER CODE BEGIN PTD */
31
32/* USER CODE END PTD */
33
34/* Private define -----*/
35/* USER CODE BEGIN PD */
36
37/* USER CODE END PD */
38
39/* Private macro -----*/
40/* USER CODE BEGIN PM */
41
42/* USER CODE END PM */
43
44/* Private variables -----*/
45TIM_HandleTypeDef htim16;
46
47/* USER CODE BEGIN PV */
48// TODO: Define input variables
49#define SW0_PIN GPIO_IDR_0
50#define SW1_PIN GPIO_IDR_1
51#define SW2_PIN GPIO_IDR_2
52#define SW3_PIN GPIO_IDR_3
53
54#define LED0_PIN GPIO_ODR_4
55#define LED1_PIN GPIO_ODR_5
56#define LED2_PIN GPIO_ODR_6
57#define LED3_PIN GPIO_ODR_7
58
59// Mode definitions
60typedef enum {
61     MODE_OFF = 0,

```

```

62  MODE_1_BACK_FORTH = 1,
63  MODE_2_INVERSE_BACK_FORTH = 2,
64  MODE_3_SPARKLE = 3
65 } led_mode_t;
66
67 // Global variables
68 volatile led_mode_t current_mode = MODE_OFF;
69 volatile uint8_t led_position = 0;
70 volatile int8_t direction = 1; // 1 for forward, -1 for backward
71 volatile uint8_t fast_mode = 0; // 0 for 1s, 1 for 0.5s
72 volatile uint8_t sparkle_state = 0; // For sparkle mode state machine
73 volatile uint8_t sparkle_pattern = 0;
74 volatile uint8_t sparkle_delay_counter = 0;
75 volatile uint8_t sparkle_off_counter = 0;
76 volatile uint8_t leds_to_turn_off = 0;
77
78 // Button state tracking for debouncing
79 uint8_t prev_button_state[4] = {1, 1, 1, 1}; // Pull-up means 1 when not pressed
80
81 /* USER CODE END PV */
82
83 /* Private function prototypes -----*/
84 void SystemClock_Config(void);
85 static void MX_GPIO_Init(void);
86 static void MX_TIM16_Init(void);
87 /* USER CODE BEGIN PFP */
88 void TIM16_IRQHandler(void);
89 void set_tim16_delay_ms(uint16_t ms)
90 {
91
92     if (ms < 1) ms = 1;
93     __HAL_TIM_DISABLE(&htim16);
94     __HAL_TIM_SET_AUTORELOAD(&htim16, ms - 1); // Update ARR register directly
95     __HAL_TIM_SET_COUNTER(&htim16, 0); // Reset counter
96     __HAL_TIM_ENABLE(&htim16); // Re-enable timer
97 }
98
99 void update_led_pattern(void);
100 void set_led_pattern(uint8_t pattern);
101 uint8_t get_random_byte(void);
102 /* USER CODE END PFP */
103
104 /* Private user code -----*/
105 /* USER CODE BEGIN 0 */
106
107 /* USER CODE END 0 */
108
109 /**
110  * @brief The application entry point.
111  * @retval int
112  */
113 int main(void)
114 {
115
116     /* USER CODE BEGIN 1 */
117     /* USER CODE END 1 */
118
119     /* MCU Configuration-----*/
120
121     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */

```

```

123 HAL_Init();
124
125 /* USER CODE BEGIN Init */
126 /* USER CODE END Init */
127
128 /* Configure the system clock */
129 SystemClock_Config();
130
131 /* USER CODE BEGIN SysInit */
132 /* USER CODE END SysInit */
133
134 /* Initialize all configured peripherals */
135 MX_GPIO_Init();
136 MX_TIM16_Init();
137 /* USER CODE BEGIN 2 */
138
139 // TODO: Start timer TIM16
140 HAL_TIM_Base_Start_IT(&htim16);
141
142 // Initialize random seed
143 srand(HAL_GetTick());
144
145 /* USER CODE END 2 */
146
147 /* Infinite loop */
148 /* USER CODE BEGIN WHILE */
149 while (1)
150 {
151
152     /* USER CODE END WHILE */
153
154     /* USER CODE BEGIN 3 */
155
156     // TODO: Check pushbuttons to change timer delay
157
158     // Read current button states
159     uint8_t current_button_state[4];
160     current_button_state[0] = (GPIOA->IDR & GPIO_IDR_0) ? 1 : 0;
161     current_button_state[1] = (GPIOA->IDR & GPIO_IDR_1) ? 1 : 0;
162     current_button_state[2] = (GPIOA->IDR & GPIO_IDR_2) ? 1 : 0;
163     current_button_state[3] = (GPIOA->IDR & GPIO_IDR_3) ? 1 : 0;
164
165     // Check for button presses (falling edge detection)
166
167     // PA0 - Toggle timing (1s <-> 0.5s)
168     if (prev_button_state[0] == 1 && current_button_state[0] == 0) {
169         fast_mode = !fast_mode;
170         if (fast_mode) {
171             htim16.Init.Period = 500-1; // 0.5 second
172         } else {
173             htim16.Init.Period = 1000-1; // 1 second
174         }
175         HAL_TIM_Base_Init(&htim16);
176     }
177
178     // PA1 - Mode 1: Back/forth
179     if (prev_button_state[1] == 1 && current_button_state[1] == 0) {
180         current_mode = MODE_1_BACK_FORTH;
181         led_position = 0;
182         direction = 1;
183

```

```

184
185 // PA2 - Mode 2: Inverse back/forth
186 if (prev_button_state[2] == 1 && current_button_state[2] == 0) {
187     current_mode = MODE_2_INVERSE_BACK_FORTH;
188     led_position = 0;
189     direction = 1;
190 }
191
192 // PA3 - Mode 3: Sparkle
193 if (prev_button_state[3] == 1 && current_button_state[3] == 0) {
194     current_mode = MODE_3_SPARKLE;
195     sparkle_state = 0;
196     sparkle_delay_counter = 0; //
197     sparkle_off_counter = 0;
198 }
199
200 // Update previous button states
201 for (int i = 0; i < 4; i++) {
202     prev_button_state[i] = current_button_state[i];
203 }
204
205 HAL_Delay(10); // Small delay for debouncing
206
207
208 /* USER CODE END 3 */
209
210
211 /**
212  * @brief System Clock Configuration
213  * @retval None
214  */
215 void SystemClock_Config(void)
216 {
217     LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
218     while (LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
219     {
220     }
221     LL_RCC_HSI_Enable();
222
223     /* Wait till HSI is ready */
224     while (LL_RCC_HSI_IsReady() != 1)
225     {
226     }
227
228     LL_RCC_HSI_SetCalibTrimming(16);
229     LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
230     LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
231     LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
232
233     /* Wait till System clock is ready */
234     while (LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
235     {
236     }
237
238     LL_SetSystemCoreClock(8000000);
239
240     /* Update the time base */
241     if (HAL_InitTick(TICK_INT_PRIORITY) != HAL_OK)
242     {
243         Error_Handler();
244     }

```

```

245
246
247 /**
248  * @brief TIM16 Initialization Function
249  * @param None
250  * @retval None
251  */
252 static void MX_TIM16_Init(void)
253 {
254
255     /* USER CODE BEGIN TIM16_Init 0 */
256
257     /* USER CODE END TIM16_Init 0 */
258
259     /* USER CODE BEGIN TIM16_Init 1 */
260
261     /* USER CODE END TIM16_Init 1 */
262     htim16.Instance = TIM16;
263     htim16.Init.Prescaler = 8000-1;
264     htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
265     htim16.Init.Period = 1000-1;
266     htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
267     htim16.Init.RepetitionCounter = 0;
268     htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
269     if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
270     {
271         Error_Handler();
272     }
273     /* USER CODE BEGIN TIM16_Init 2 */
274     NVIC_EnableIRQ(TIM16_IRQn);
275     /* USER CODE END TIM16_Init 2 */
276
277 }
278
279 /**
280  * @brief GPIO Initialization Function
281  * @param None
282  * @retval None
283  */
284 static void MX_GPIO_Init(void)
285 {
286     LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
287     /* USER CODE BEGIN MX_GPIO_Init_1 */
288     /* USER CODE END MX_GPIO_Init_1 */
289
290     /* GPIO Ports Clock Enable */
291     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
292     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
293     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
294
295     /**/
296     LL_GPIO_ResetOutputPin(LED0_GPIO_Port, LED0_Pin);
297
298     /**/
299     LL_GPIO_ResetOutputPin(LED1_GPIO_Port, LED1_Pin);
300
301     /**/
302     LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
303
304     /**/
305     LL_GPIO_ResetOutputPin(LED3_GPIO_Port, LED3_Pin);

```

```

306
307 /**/
308 LL_GPIO_ResetOutputPin(LED4_GPIO_Port, LED4_Pin);
309
310 /**/
311 LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);
312
313 /**/
314 LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);
315
316 /**/
317 LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
318
319 /**/
320 GPIO_InitStruct.Pin = Button0_Pin;
321 GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
322 GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
323 LL_GPIO_Init(Button0_GPIO_Port, &GPIO_InitStruct);
324
325 /**/
326 GPIO_InitStruct.Pin = Button1_Pin;
327 GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
328 GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
329 LL_GPIO_Init(Button1_GPIO_Port, &GPIO_InitStruct);
330
331 /**/
332 GPIO_InitStruct.Pin = Button2_Pin;
333 GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
334 GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
335 LL_GPIO_Init(Button2_GPIO_Port, &GPIO_InitStruct);
336
337 /**/
338 GPIO_InitStruct.Pin = Button3_Pin;
339 GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
340 GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
341 LL_GPIO_Init(Button3_GPIO_Port, &GPIO_InitStruct);
342
343 /**/
344 GPIO_InitStruct.Pin = LED0_Pin;
345 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
346 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
347 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
348 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
349 LL_GPIO_Init(LED0_GPIO_Port, &GPIO_InitStruct);
350
351 /**/
352 GPIO_InitStruct.Pin = LED1_Pin;
353 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
354 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
355 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
356 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
357 LL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
358
359 /**/
360 GPIO_InitStruct.Pin = LED2_Pin;
361 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
362 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
363 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
364 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
365 LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
366

```

```

367m /**/
368 GPIO_InitStruct.Pin = LED3_Pin;
369 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
370 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
371 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
372 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
373 LL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);
374
375 /**/
376 GPIO_InitStruct.Pin = LED4_Pin;
377 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
378 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
379 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
380 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
381 LL_GPIO_Init(LED4_GPIO_Port, &GPIO_InitStruct);
382
383 /**/
384 GPIO_InitStruct.Pin = LED5_Pin;
385 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
386 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
387 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
388 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
389 LL_GPIO_Init(LED5_GPIO_Port, &GPIO_InitStruct);
390
391 /**/
392 GPIO_InitStruct.Pin = LED6_Pin;
393 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
394 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
395 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
396 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
397 LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);
398
399 /**/
400 GPIO_InitStruct.Pin = LED7_Pin;
401 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
402 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
403 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
404 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
405 LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
406
407 /* USER CODE BEGIN MX_GPIO_Init_2 */
408 /* USER CODE END MX_GPIO_Init_2 */
409 }
410
411 /* USER CODE BEGIN 4 */
412 void TIM16_IRQHandler(void)
413 {
414     // Acknowledge interrupt
415     HAL_TIM_IRQHandler(&tim16);
416
417     // TODO: Change LED pattern
418     update_led_pattern();
419 }
420
421 void update_led_pattern(void)
422 {
423     switch (current_mode)
424     {
425         case MODE_OFF:
426             set_led_pattern(0x00); // All LEDs off
427             break;

```

```

428
429     case MODE_1_BACK_FORTH:
430         // Single LED cycling back and forth
431         set_led_pattern(1 << led_position);
432
433         // Update position for next interrupt
434         led_position += direction;
435         if (led_position > 7) {
436             direction = -1;
437             led_position = 7; // Skip repeating LED7
438         } else if (led_position < 0) {
439             direction = 1;
440             led_position = 0; // Skip repeating LED0
441         }
442         break;
443
444     case MODE_2_INVERSE_BACK_FORTH:
445         // All LEDs on except one, cycling back and forth
446         set_led_pattern(0xFF ^ (1 << led_position));
447
448         // Update position for next interrupt (same logic as Mode 1)
449         led_position += direction;
450         if (led_position > 7) {
451             direction = -1;
452             led_position = 7;
453         } else if (led_position < 0) {
454             direction = 1;
455             led_position = 0;
456         }
457         break;
458
459     case MODE_3_SPARKLE:
460         switch (sparkle_state) {
461             case 0: // All LEDs off → generate new random pattern
462                 sparkle_pattern = get_random_byte(); // 0-255
463                 leds_to_turn_off = sparkle_pattern;
464                 set_led_pattern(sparkle_pattern);
465
466                 set_tim16_delay_ms((rand() % 1401) + 100); // Hold for 100-1500 ms
467                 sparkle_state = 1;
468                 break;
469
470             case 1: // Done holding, now begin turning LEDs off
471                 sparkle_state = 2;
472                 break;
473
474             case 2: // Turn off one LED at a time
475                 if (leds_to_turn_off != 0) {
476                     uint8_t index = 0;
477                     uint8_t mask = leds_to_turn_off;
478                     while ((mask & 1) == 0 && index < 8) {
479                         mask >>= 1;
480                         index++;
481                     }
482
483                     if (index < 8) {
484                         sparkle_pattern &= ~(1 << index);
485                         leds_to_turn_off &= ~(1 << index);
486                         set_led_pattern(sparkle_pattern);
487                     }
488

```



```

489         // Set delay
490         set_tim16_delay_ms(100); // 100
491         sparkle_state = 3;
492     } else {
493         sparkle_state = 0; // Restart
494     }
495     break;
496
497     case 3: // Done waiting → go back to turn off next LED
498         sparkle_state = 2;
499         break;
500 }
501 break;
502
503
504
505 }
506
507 }
508
509 void set_led_pattern(uint8_t pattern)
510 {
511     // Assuming LEDs are connected to GPIOB pins 0-7
512     // Clear all LEDs first
513     GPIOB->ODR &= 0xFF00;
514     // Set the pattern
515     GPIOB->ODR |= pattern;
516 }
517
518 uint8_t get_random_byte(void)
519 {
520     return (uint8_t)(rand() & 0xFF);
521 }
522
523 /* USER CODE END 4 */
524
525 /**
526  * @brief This function is executed in case of error occurrence.
527  * @retval None
528  */
529 void Error_Handler(void)
530 {
531     /* USER CODE BEGIN Error_Handler_Debug */
532     /* User can add his own implementation to report the HAL error return state */
533     __disable_irq();
534     while (1)
535     {
536     }
537     /* USER CODE END Error_Handler_Debug */
538 }
539
540 #ifndef USE_FULL_ASSERT
541 /**
542  * @brief Reports the name of the source file and the source line number
543  * where the assert_param error has occurred.
544  * @param file: pointer to the source file name
545  * @param line: assert_param error line source number
546  * @retval None
547  */
548 void assert_failed(uint8_t *file, uint32_t line)
549 {

```

```
550main.c/*USER CODE BEGIN 6 */
551 /* User can add his own implementation to report the file name and line number,
552    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
553 /* USER CODE END 6 */
554
555 #endif /* USE_FULL_ASSERT */
556
```