```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"
/* 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
#include <stdlib.h>
char mode = 0; //The mode of the LEDs
char direction = 0; //The direction that the LED moves in modes 1 and 2
char index = 0; //The index of the LED in modes 1 and 2
char speed = 0;
uint32_t randint(uint32_t min, uint32_t max) {
 return (rand()%(max-min))+min;
}

/* 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 */
```

```c
/* 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
//  RCC->APB2ENR |= RCC_APB2ENR_TIM16EN;
//  TIM16->PSC = 8000-1;
//  TIM16->ARR = 1000-1;
  TIM16->DIER |= TIM_DIER_UIE;
  NVIC_EnableIRQ(TIM16_IRQn);
  TIM16->CR1 |= TIM_CR1_CEN;

  /* 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
if (!(GPIOA->IDR&(1<<0))) {  //if button 0 is pressed
 speed = !speed;
 if (speed) {TIM16->ARR = 500-1;} //set delay to 0.5 seconds
 else {TIM16->ARR = 1000-1;} //set delay to 1 second
}
if (!(GPIOA->IDR&(1<<1))) {  //if button 1 is pressed
 mode = 1;
 if (speed) {TIM16->ARR = 500-1;} //set delay to 0.5 seconds
 else {TIM16->ARR = 1000-1;} //set delay to 1 second
}
```

```
131 if (!(GPIOA->IDR&(1<<2))) {  //if button 2 is pressed
132  mode = 2;
133  if (speed) {TIM16->ARR = 500-1;} //set delay to 0.5 seconds
134  else {TIM16->ARR = 1000-1;} //set delay to 1 second
135 }
136 if (!(GPIOA->IDR&(1<<3))) {  //if button 3 is pressed
137  mode = 3;
138  GPIOB->ODR &= ~0xFF;
139 }
140
141
142
143   }
144   /* USER CODE END 3 */
145 }
146
147 /**
148   * @brief System Clock Configuration
149   * @retval None
150   */
151 void SystemClock_Config(void)
152 {
153   LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
154   while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
155   {
156   }
157   LL_RCC_HSI_Enable();
158
159    /* Wait till HSI is ready */
160   while(LL_RCC_HSI_IsReady() != 1)
161   {
162
163   }
164   LL_RCC_HSI_SetCalibTrimming(16);
165   LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
166   LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
167   LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
168
169    /* Wait till System clock is ready */
170   while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
171   {
172
173   }
174   LL_SetSystemCoreClock(8000000);
175
176    /* Update the time base */
177   if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
178   {
179     Error_Handler();
180   }
181 }
182
183 /**
184   * @brief TIM16 Initialization Function
185   * @param None
186   * @retval None
187   */
188 static void MX_TIM16_Init(void)
189 {
190
191   /* USER CODE BEGIN TIM16_Init 0 */
192
193   /* USER CODE END TIM16_Init 0 */
194
195   /* USER CODE BEGIN TIM16_Init 1 */
```

```c
196
197   /* USER CODE END TIM16_Init 1 */
198   htim16.Instance = TIM16;
199   htim16.Init.Prescaler = 8000-1;
200   htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
201   htim16.Init.Period = 1000-1;
202   htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
203   htim16.Init.RepetitionCounter = 0;
204   htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
205   if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
206   {
207     Error_Handler();
208   }
209   /* USER CODE BEGIN TIM16_Init 2 */
210   NVIC_EnableIRQ(TIM16_IRQn);
211   /* USER CODE END TIM16_Init 2 */
212
213 }
214
215 /**
216   * @brief GPIO Initialization Function
217   * @param None
218   * @retval None
219   */
220 static void MX_GPIO_Init(void)
221 {
222   LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
223 /* USER CODE BEGIN MX_GPIO_Init_1 */
224 /* USER CODE END MX_GPIO_Init_1 */
225
226   /* GPIO Ports Clock Enable */
227   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
228   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
229   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
230
231   /**/
232   LL_GPIO_ResetOutputPin(LED0_GPIO_Port, LED0_Pin);
233
234   /**/
235   LL_GPIO_ResetOutputPin(LED1_GPIO_Port, LED1_Pin);
236
237   /**/
238   LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
239
240   /**/
241   LL_GPIO_ResetOutputPin(LED3_GPIO_Port, LED3_Pin);
242
243   /**/
244   LL_GPIO_ResetOutputPin(LED4_GPIO_Port, LED4_Pin);
245
246   /**/
247   LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);
248
249   /**/
250   LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);
251
252   /**/
253   LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
254
255   /**/
256   GPIO_InitStruct.Pin = Button0_Pin;
257   GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
258   GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
259   LL_GPIO_Init(Button0_GPIO_Port, &GPIO_InitStruct);
260
```

```
261  /**/
262  GPIO_InitStruct.Pin = Button1_Pin;
263  GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
264  GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
265  LL_GPIO_Init(Button1_GPIO_Port, &GPIO_InitStruct);
266
267  /**/
268  GPIO_InitStruct.Pin = Button2_Pin;
269  GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
270  GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
271  LL_GPIO_Init(Button2_GPIO_Port, &GPIO_InitStruct);
272
273  /**/
274  GPIO_InitStruct.Pin = Button3_Pin;
275  GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
276  GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
277  LL_GPIO_Init(Button3_GPIO_Port, &GPIO_InitStruct);
278
279  /**/
280  GPIO_InitStruct.Pin = LED0_Pin;
281  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
282  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
283  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
284  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
285  LL_GPIO_Init(LED0_GPIO_Port, &GPIO_InitStruct);
286
287  /**/
288  GPIO_InitStruct.Pin = LED1_Pin;
289  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
290  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
291  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
292  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
293  LL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
294
295  /**/
296  GPIO_InitStruct.Pin = LED2_Pin;
297  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
298  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
299  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
300  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
301  LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
302
303  /**/
304  GPIO_InitStruct.Pin = LED3_Pin;
305  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
306  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
307  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
308  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
309  LL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);
310
311  /**/
312  GPIO_InitStruct.Pin = LED4_Pin;
313  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
314  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
315  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
316  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
317  LL_GPIO_Init(LED4_GPIO_Port, &GPIO_InitStruct);
318
319  /**/
320  GPIO_InitStruct.Pin = LED5_Pin;
321  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
322  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
323  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
324  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
325  LL_GPIO_Init(LED5_GPIO_Port, &GPIO_InitStruct);
```

```
326
327   /**/
328   GPIO_InitStruct.Pin = LED6_Pin;
329   GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
330   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
331   GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
332   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
333   LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);
334
335   /**/
336   GPIO_InitStruct.Pin = LED7_Pin;
337   GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
338   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
339   GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
340   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
341   LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
342
343 /* USER CODE BEGIN MX_GPIO_Init_2 */
344 /* USER CODE END MX_GPIO_Init_2 */
345 }
346
347 /* USER CODE BEGIN 4 */
348 void TIM16_IRQHandler(void)
349 {
350  // Acknowledge interrupt
351  HAL_TIM_IRQHandler(&htim16);//#
352
353
354  // TODO: Change LED pattern
355  TIM16->SR &= ~TIM_SR_UIF; //clear flag
356
357  if (mode==0) {return;} //if there's no mode yet
358
359  if (mode==3) {
360   if (GPIOB->ODR&0xFF) {  //if there are active LEDs
361    for (uint32_t i=1; i<=(1<<7); i=i<<1) {  //check all the LEDs
362     if (GPIOB->ODR&i) {  //if this LED is on
363      GPIOB->ODR &= ~(i); //turn off the LED
364      //check LEDs again
365      if (GPIOB->ODR&0xFF) {  //if there are active LEDs
366       TIM16->ARR = randint(9, 100); //note: the effects of changing the ARR only appear after
367 //    TIM16->ARR = 100-1;
368      } else {
369       TIM16->ARR = randint(99, 1500); //note: the effects of changing the ARR only appear after
370 //    TIM16->ARR = 500-1;
371      }
372      break;
373     }
374    }
375   } else {
376    GPIOB->ODR |= randint(0, 256); //turn on random LEDs
377    TIM16->ARR = randint(9, 100); //note: the effects of changing the ARR only appear after the
378 //  TIM16->ARR = 100-1;
379   }
380
381  } else {  //modes 1 and 2
382   //clear the LEDs
383   if (mode==2) {
384    GPIOB->ODR |= 0xFF;
385   } else {
386    GPIOB->ODR &= ~0xFF;
387   }
388   //change the index
389   if (direction) {
390    if (index==7) {
```

```
    direction = 0;
    index = 6;
   } else {
    index++;
   }
  } else {
   if (index==0) {
    direction = 1;
    index = 1;
   } else {
    index--;
   }
  }
  //write to the correct LED
  if (mode==2) {
   GPIOB->ODR &= ~(1<<index);
  } else {
   GPIOB->ODR |= (1<<index);
  }
 }



}


/* 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 */

```