

## GitHub Link

[https://github.com/LuthoYRN/MNGLUT008\\_CBXLIS001\\_EEE3096S/blob/main/Prac1/main.c](https://github.com/LuthoYRN/MNGLUT008_CBXLIS001_EEE3096S/blob/main/Prac1/main.c)

## Description of LED Sequence Implementation

The code initializes an array of 8-bit unsigned integers, **led\_patterns[]**, which stores 9 distinct LED patterns. The variable **current** is used to track the index of the currently displayed pattern, starting from 0. The LED patterns are updated within the **TIM16\_IRQHandler** function, which is called upon a timer interrupt. When the interrupt occurs, the LED output is updated to match the current pattern, and the **current** index is incremented, looping back to 0 after reaching the last pattern.

## Description of Timer Interrupt Functionality

The timer interrupt functionality is handled by the **TIM16\_IRQHandler** function. This function checks if the TIM16 update interrupt flag is set and clears it. It then updates the LED outputs to reflect the pattern specified by **current** variable and increments the **current** to point to the next pattern. Additionally, the code includes logic to change the timer delay based on the state of four pushbuttons. The pin states of PA0-PA3, corresponding to the pushbuttons, are checked continuously within an infinite while loop in the main method via if statements. Inside each statement, TIM16 is configured for different delays by changing the Prescaler and ARR value based on calculations (using an 8MHZ clock), except for the last pin PA3, which just resets the **current** index to 0 to restart the pattern sequence.

$$\begin{aligned}\text{Cycle Length} &= \frac{1}{f_{\text{APBbus}}} \\ \text{Cycles Needed} &= \frac{\text{Delay}}{\text{Cycle Length}} \\ \text{Prescaler} &= \frac{\text{Cycles Needed}}{2^{16}} \\ \text{Auto-Reload Register} &= \frac{\text{Number of Cycles}}{\text{Prescaler}}\end{aligned}$$

Calculation used to configure Prescaler and ARR values

## Appendix

```
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 /* USER CODE END Includes */
27
28 /* Private typedef -----*/
29 /* USER CODE BEGIN PTD */
30
31 /* USER CODE END PTD */
32
33 /* Private define -----*/
34 /* USER CODE BEGIN PD */
35 /* USER CODE END PD */
36
37 /* Private macro -----*/
38 /* USER CODE BEGIN PM */
39
40 /* USER CODE END PM */
41
42 /* Private variables -----*/
43 TIM_HandleTypeDef htim16;
44
45 /* USER CODE BEGIN PV */
46 // TODO: Define input variable
47 int current = 0;
48 uint8_t led_patterns[] = {0b11101001, 0b11010010, 0b10100100, 0b01001000,
49                          0b10010000, 0b00100000, 0b01000000, 0b10000000, 0x0};
50
51 /* USER CODE END PV */
52
53 /* Private function prototypes -----*/
54 void SystemClock_Config(void);
55 static void MX_GPIO_Init(void);
56 static void MX_TIM16_Init(void);
57 /* USER CODE BEGIN PFP */
58 void TIM16_IRQHandler(void);
59 /* USER CODE END PFP */
60
61 /* Private user code -----*/
62 /* USER CODE BEGIN 0 */
63
64 /* USER CODE END 0 */
65
```

```

66  /**
67   * @brief The application entry point.
68   * @retval int
69   */
70  int main(void)
71  {
72      /* USER CODE BEGIN 1 */
73      /* USER CODE END 1 */
74
75      /* MCU Configuration-----*/
76
77      /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
78      HAL_Init();
79
80      /* USER CODE BEGIN Init */
81      /* USER CODE END Init */
82
83      /* Configure the system clock */
84      SystemClock_Config();
85
86      /* USER CODE BEGIN SysInit */
87      /* USER CODE END SysInit */
88
89      /* Initialize all configured peripherals */
90      MX_GPIO_Init();
91      MX_TIM16_Init();
92      /* USER CODE BEGIN 2 */
93
94      // TODO: Start timer TIM16
95      HAL_TIM_Base_Start_IT(&htim16);
96      /* USER CODE END 2 */
97
98      /* Infinite loop */
99      /* USER CODE BEGIN WHILE */
100     while (1)
101     {
102         /* USER CODE END WHILE */
103
104         /* USER CODE BEGIN 3 */
105         // TODO: Check pushbuttons to change timer delay
106         if (LL_GPIO_IsInputPinSet(GPIOA, LL_GPIO_PIN_0) == 0){
107             //0.5 seconds delay
108             HAL_TIM_Base_Stop(&htim16);
109             if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
110             {
111                 Error_Handler();
112             }
113             htim16.Init.Period = 64516;
114             htim16.Init.Prescaler =61;
115             HAL_TIM_Base_Start_IT(&htim16);
116         }
117         if (LL_GPIO_IsInputPinSet(GPIOA, LL_GPIO_PIN_1)== 0){
118             //2 seconds delay
119             HAL_TIM_Base_Stop(&htim16);
120             if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
121             {
122                 Error_Handler();
123             }
124             htim16.Init.Period = 65306;
125             htim16.Init.Prescaler = 245;
126             HAL_TIM_Base_Start_IT(&htim16);
127         }
128         if (LL_GPIO_IsInputPinSet(GPIOA, LL_GPIO_PIN_2)== 0){
129             //1 seconds delay
130             HAL_TIM_Base_Stop(&htim16);
131             if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
132             {

```

```

133         Error_Handler();
134     }
135     htim16.Init.Period = 65040;
136     htim16.Init.Prescaler =123 ;
137
138     HAL_TIM_Base_Start_IT(&htim16);
139 }
140 if (LL_GPIO_IsInputPinSet(GPIOA, LL_GPIO_PIN_3 )== 0){
141     //back to pattern 1
142     current = 0;
143 }
144 }
145 /* USER CODE END 3 */
146 }
147
148 /**
149  * @brief System Clock Configuration
150  * @retval None
151  */
152 void SystemClock_Config(void)
153 {
154     LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
155     while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
156     {
157     }
158     LL_RCC_HSI_Enable();
159
160     /* Wait till HSI is ready */
161     while(LL_RCC_HSI_IsReady() != 1)
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
189 static void MX_TIM16_Init(void)
190 {
191
192     /* USER CODE BEGIN TIM16_Init 0 */
193
194     /* USER CODE END TIM16_Init 0 */
195
196     /* USER CODE BEGIN TIM16_Init 1 */
197
198     /* USER CODE END TIM16_Init 1 */
199     htim16.Instance = TIM16;

```

```

200 htim16.Init.Prescaler = 123;
201 htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
202 htim16.Init.Period = 65040;
203 htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
204 htim16.Init.RepetitionCounter = 0;
205 htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
206 if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
207 {
208     Error_Handler();
209 }
210 /* USER CODE BEGIN TIM16_Init 2 */
211 NVIC_EnableIRQ(TIM16_IRQn);
212 /* USER CODE END TIM16_Init 2 */
213
214 }
215
216 /**
217  * @brief GPIO Initialization Function
218  * @param None
219  * @retval None
220  */
221 static void MX_GPIO_Init(void)
222 {
223     LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
224     /* USER CODE BEGIN MX_GPIO_Init_1 */
225     /* USER CODE END MX_GPIO_Init_1 */
226
227     /* GPIO Ports Clock Enable */
228     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
229     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
230     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
231
232     /**/
233     LL_GPIO_ResetOutputPin(LED0_GPIO_Port, LED0_Pin);
234
235     /**/
236     LL_GPIO_ResetOutputPin(LED1_GPIO_Port, LED1_Pin);
237
238     /**/
239     LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
240
241     /**/
242     LL_GPIO_ResetOutputPin(LED3_GPIO_Port, LED3_Pin);
243
244     /**/
245     LL_GPIO_ResetOutputPin(LED4_GPIO_Port, LED4_Pin);
246
247     /**/
248     LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);
249
250     /**/
251     LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);
252
253     /**/
254     LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
255
256     /**/
257     GPIO_InitStruct.Pin = Button0_Pin;
258     GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
259     GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
260     LL_GPIO_Init(Button0_GPIO_Port, &GPIO_InitStruct);
261
262     /**/
263     GPIO_InitStruct.Pin = Button1_Pin;
264     GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
265     GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
266     LL_GPIO_Init(Button1_GPIO_Port, &GPIO_InitStruct);

```

```

267
268 /**/
269 GPIO_InitStruct.Pin = Button2_Pin;
270 GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
271 GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
272 LL_GPIO_Init(Button2_GPIO_Port, &GPIO_InitStruct);
273
274 /**/
275 GPIO_InitStruct.Pin = Button3_Pin;
276 GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
277 GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
278 LL_GPIO_Init(Button3_GPIO_Port, &GPIO_InitStruct);
279
280 /**/
281 GPIO_InitStruct.Pin = LED0_Pin;
282 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
283 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
284 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
285 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
286 LL_GPIO_Init(LED0_GPIO_Port, &GPIO_InitStruct);
287
288 /**/
289 GPIO_InitStruct.Pin = LED1_Pin;
290 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
291 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
292 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
293 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
294 LL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
295
296 /**/
297 GPIO_InitStruct.Pin = LED2_Pin;
298 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
299 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
300 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
301 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
302 LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
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 GPIO_InitStruct.Pin = LED6_Pin;
328 GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
329 GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
330 GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
331 GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
332 LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);
333

```

```

334  /**/
335  GPIO_InitStruct.Pin = LED7_Pin;
336  GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
337  GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
338  GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
339  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
340  LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
341
342  /**/
343  GPIO_InitStruct.Pin = LL_GPIO_PIN_9;
344  GPIO_InitStruct.Mode = LL_GPIO_MODE_INPUT;
345  GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
346  LL_GPIO_Init(GPIOB, &GPIO_InitStruct);
347
348  /* USER CODE BEGIN MX_GPIO_Init_2 */
349  /* USER CODE END MX_GPIO_Init_2 */
350  }
351
352  /* USER CODE BEGIN 4 */
353
354  // Timer rolled over
355  void TIM16_IRQHandler(void)
356  {
357      // Acknowledge interrupt
358      HAL_TIM_IRQHandler(&htim16);
359      // TODO: Change LED pattern
360      if ( __HAL_TIM_GET_IT_SOURCE(&htim16, TIM_IT_UPDATE)) {
361          __HAL_TIM_CLEAR_FLAG(&htim16, TIM_FLAG_UPDATE);
362          LL_GPIO_WriteOutputPort(LED0_GPIO_Port, led_patterns[current]);
363          current++;
364          if (current>8)current=0;
365      }
366  }
367  /* USER CODE END 4 */
368
369  /**
370   * @brief This function is executed in case of error occurrence.
371   * @retval None
372   */
373  void Error_Handler(void)
374  {
375      /* USER CODE BEGIN Error_Handler_Debug */
376      /* User can add his own implementation to report the HAL error return state */
377      __disable_irq();
378      while (1)
379      {
380      }
381      /* USER CODE END Error_Handler_Debug */
382  }
383
384  #ifndef USE_FULL_ASSERT
385  /**
386   * @brief Reports the name of the source file and the source line number
387   *        where the assert_param error has occurred.
388   * @param file: pointer to the source file name
389   * @param line: assert_param error line source number
390   * @retval None
391   */
392
393  void assert_failed(uint8_t *file, uint32_t line)
394  {
395      /* USER CODE BEGIN 6 */
396      /* User can add his own implementation to report the file name and line number,
397       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
398      /* USER CODE END 6 */
399  }
400  #endif /* USE_FULL_ASSERT */

```