```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 <stdio.h>
#include "stm32f0xx.h"
#include <lcd_stm32f0.c>
/* 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 ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;
TIM_HandleTypeDef htim3;

/* USER CODE BEGIN PV */
uint32_t prev_millis = 0;
uint32_t curr_millis = 0;
uint32_t delay_t = 500; // Initialise delay to 500ms
uint32_t adc_val;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
```

```c
static void MX_ADC_Init(void);
static void MX_TIM3_Init(void);

/* USER CODE BEGIN PFP */
void EXTI0_1_IRQHandler(void);
void writeLCD(char *char_in);
uint32_t pollADC(void);
uint32_t ADCtoCCR(uint32_t adc_val);
/* 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_ADC_Init();
  MX_TIM3_Init();

  /* USER CODE BEGIN 2 */
  init_LCD();

  // PWM setup
  uint32_t CCR = 0;
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); // Start PWM on TIM3 Channel 3
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    // LED0
    HAL_GPIO_TogglePin(GPIOB, LED7_Pin);
```

```
115
116     // ADC to LCD; TODO: Read POT1 value and write to LCD
117
118      // Read ADC value using the given function
119         adc_val = pollADC();
120
121         // Get string of ADC value to print to LCD
122         char adc_line[16];
123         snprintf(adc_line, sizeof(adc_line), "ADC Value: %lu", adc_val);
124
125         // Display ADC value on the LCD
126         writeLCD(adc_line);
127
128     // Update PWM value; TODO: Get CRR
129         uint32_t CCR = ADCtoCCR(adc_val);
130
131     __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, CCR);
132
133     // Wait for delay ms
134     HAL_Delay (delay_t);
135     /* USER CODE END WHILE */
136
137     /* USER CODE BEGIN 3 */
138   }
139   /* USER CODE END 3 */
140 }
141
142 /**
143   * @brief System Clock Configuration
144   * @retval None
145   */
146 void SystemClock_Config(void)
147 {
148   LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
149   while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
150   {
151   }
152   LL_RCC_HSI_Enable();
153
154    /* Wait till HSI is ready */
155   while(LL_RCC_HSI_IsReady() != 1)
156   {
157
158   }
159   LL_RCC_HSI_SetCalibTrimming(16);
160   LL_RCC_HSI14_Enable();
161
162    /* Wait till HSI14 is ready */
163   while(LL_RCC_HSI14_IsReady() != 1)
164   {
165
166   }
167   LL_RCC_HSI14_SetCalibTrimming(16);
168   LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
169   LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
170   LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
171
```

```
172   /* Wait till System clock is ready */
173   while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
174   {
175
176   }
177   LL_SetSystemCoreClock(8000000);
178
179    /* Update the time base */
180   if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
181   {
182     Error_Handler();
183   }
184   LL_RCC_HSI14_EnableADCControl();
185 }
186
187 /**
188   * @brief ADC Initialization Function
189   * @param None
190   * @retval None
191   */
192 static void MX_ADC_Init(void)
193 {
194
195   /* USER CODE BEGIN ADC_Init 0 */
196   /* USER CODE END ADC_Init 0 */
197
198   ADC_ChannelConfTypeDef sConfig = {0};
199
200   /* USER CODE BEGIN ADC_Init 1 */
201
202   /* USER CODE END ADC_Init 1 */
203
204   /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number
      of conversion)
205   */
206   hadc.Instance = ADC1;
207   hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
208   hadc.Init.Resolution = ADC_RESOLUTION_12B;
209   hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
210   hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
211   hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
212   hadc.Init.LowPowerAutoWait = DISABLE;
213   hadc.Init.LowPowerAutoPowerOff = DISABLE;
214   hadc.Init.ContinuousConvMode = DISABLE;
215   hadc.Init.DiscontinuousConvMode = DISABLE;
216   hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
217   hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
218   hadc.Init.DMAContinuousRequests = DISABLE;
219   hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
220   if (HAL_ADC_Init(&hadc) != HAL_OK)
221   {
222     Error_Handler();
223   }
224
225   /** Configure for the selected ADC regular channel to be converted.
226   */
227   sConfig.Channel = ADC_CHANNEL_6;
```

```
228   sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
229   sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
230   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
231   {
232     Error_Handler();
233   }
234   /* USER CODE BEGIN ADC_Init 2 */
235   ADC1->CR |= ADC_CR_ADCAL;
236   while(ADC1->CR & ADC_CR_ADCAL);           // Calibrate the ADC
237   ADC1->CR |= (1 << 0);                     // Enable ADC
238   while((ADC1->ISR & (1 << 0)) == 0);       // Wait for ADC ready
239   /* USER CODE END ADC_Init 2 */
240
241 }
242
243 /**
244   * @brief TIM3 Initialization Function
245   * @param None
246   * @retval None
247   */
248 static void MX_TIM3_Init(void)
249 {
250
251   /* USER CODE BEGIN TIM3_Init 0 */
252
253   /* USER CODE END TIM3_Init 0 */
254
255   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
256   TIM_MasterConfigTypeDef sMasterConfig = {0};
257   TIM_OC_InitTypeDef sConfigOC = {0};
258
259   /* USER CODE BEGIN TIM3_Init 1 */
260
261   /* USER CODE END TIM3_Init 1 */
262   htim3.Instance = TIM3;
263   htim3.Init.Prescaler = 0;
264   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
265   htim3.Init.Period = 47999;
266   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
267   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
268   if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
269   {
270     Error_Handler();
271   }
272   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
273   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
274   {
275     Error_Handler();
276   }
277   if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
278   {
279     Error_Handler();
280   }
281   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
282   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
283   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
284   {
```

```
285     Error_Handler();
286   }
287   sConfigOC.OCMode = TIM_OCMODE_PWM1;
288   sConfigOC.Pulse = 0;
289   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
290   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
291   if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
292   {
293     Error_Handler();
294   }
295   /* USER CODE BEGIN TIM3_Init 2 */
296
297   /* USER CODE END TIM3_Init 2 */
298   HAL_TIM_MspPostInit(&htim3);
299
300 }
301
302 /**
303   * @brief GPIO Initialization Function
304   * @param None
305   * @retval None
306   */
307 static void MX_GPIO_Init(void)
308 {
309   LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
310   LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
311 /* USER CODE BEGIN MX_GPIO_Init_1 */
312 /* USER CODE END MX_GPIO_Init_1 */
313
314   /* GPIO Ports Clock Enable */
315   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
316   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
317   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
318
319   /**/
320   LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
321
322   /**/
323   LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);
324
325   /**/
326   LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);
327
328   /**/
329   LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);
330
331   /**/
332   EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
333   EXTI_InitStruct.LineCommand = ENABLE;
334   EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
335   EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
336   LL_EXTI_Init(&EXTI_InitStruct);
337
338   /**/
339   GPIO_InitStruct.Pin = LED7_Pin;
340   GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
341   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
```

```
342   GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
343   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
344   LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
345
346 /* USER CODE BEGIN MX_GPIO_Init_2 */
347   HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
348   HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
349 /* USER CODE END MX_GPIO_Init_2 */
350 }
351
352 /* USER CODE BEGIN 4 */
353 void EXTI0_1_IRQHandler(void)
354 {
355     // TODO: Add code to switch LED7 delay frequency
356
357     // Get the time as soon as the button is clicked
358     uint32_t curr_millis = HAL_GetTick();
359
360     // Change freq if button is pressed and 1 second has passed since last button press
361         if (HAL_GPIO_ReadPin(Button0_GPIO_Port, Button0_Pin) == 0 && curr_millis - prev_millis
    >= 1000)  // Check time since last button press
362         {
363             // Toggle the LED frequency between 1 Hz and 2 Hz
364             if (delay_t == 500)
365             {
366                 delay_t = 1000;  // 1000 delay equates to 1 Hz
367             }
368             else
369             {
370                 delay_t = 500; // 500 delay equates to to 2 Hz
371             }
372
373             // Store the time of the last button click (to be used for comparison)
374                    prev_millis = curr_millis;
375         }
376
377     HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
378 }
379
380 // TODO: Complete the writeLCD function
381 void writeLCD(char *char_in){
382     delay(3000);
383     lcd_command(CLEAR);
384     lcd_putstring(char_in);
385 }
386
387 // Get ADC value
388 uint32_t pollADC(void){
389   // TODO: Complete function body to get ADC val
390
391         // We used the HAL Adc functions to start, convert and stop the adc
392
393         HAL_ADC_Start(&hadc);
394         HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
395         uint32_t val = HAL_ADC_GetValue(&hadc);
396         HAL_ADC_Stop(&hadc);
397
```

```c
    return val;
}

// Calculate PWM CCR value - Capture/Compare Register
uint32_t ADCtoCCR(uint32_t adc_val){
  // TODO: Calculate CCR val using an appropriate equation

     // Since the ADC configured to 12-bit mode, input ADC integer is between 0-4095
     // While the Capture/Compare Register requires value in range 0-ARR(max)
     // to have control on the PWM duty cycle
    uint32_t ADCvalueRange = 4095;
    uint32_t ARRvalueRange = 47999;
    // Corresponds to 1kHz frequency for PWM signal
    // With Duty cycle = CCR/ARR

    // Equation to calculate appropriate CCR value
    uint32_t val = (adc_val * ARRvalueRange) / ADCvalueRange;

    return val;
}

void ADC1_COMP_IRQHandler(void)
{
    adc_val = HAL_ADC_GetValue(&hadc); // read adc value
    HAL_ADC_IRQHandler(&hadc); //Clear flags
}
/* 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 */
```

```
455 }
456 #endif /* USE_FULL_ASSERT */
457
```