```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)
  {
      curr_millis = HAL_GetTick();
      // Get the time as soon as the button is clicked
```

```
115      if (curr_millis - prev_millis >= 100){
116          HAL_GPIO_TogglePin(GPIOB, LED7_Pin);
117          prev_millis = curr_millis;
118      }
119
120    // ADC to LCD; TODO: Read POT1 value and write to LCD
121
122     // Read ADC value using the given function
123        adc_val = pollADC();
124
125        // Get string of ADC value to print to LCD
126        char adc_line[16];
127        snprintf(adc_line, sizeof(adc_line), "ADC Value: %lu", adc_val);
128
129        // Display ADC value on the LCD
130        writeLCD(adc_line);
131
132    // Update PWM value; TODO: Get CRR
133        uint32_t CCR = ADCtoCCR(adc_val);
134
135    __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, CCR);
136
137    // Wait for delay ms
138    HAL_Delay (delay_t);
139    /* USER CODE END WHILE */
140
141    /* USER CODE BEGIN 3 */
142  }
143  /* USER CODE END 3 */
144 }
145
146 /**
147   * @brief System Clock Configuration
148   * @retval None
149   */
150 void SystemClock_Config(void)
151 {
152   LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
153   while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
154   {
155   }
156   LL_RCC_HSI_Enable();
157
158    /* Wait till HSI is ready */
159   while(LL_RCC_HSI_IsReady() != 1)
160   {
161
162   }
163   LL_RCC_HSI_SetCalibTrimming(16);
164   LL_RCC_HSI14_Enable();
165
166    /* Wait till HSI14 is ready */
167   while(LL_RCC_HSI14_IsReady() != 1)
168   {
169
170   }
171   LL_RCC_HSI14_SetCalibTrimming(16);
```

```
172   LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
173   LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
174   LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
175
176    /* Wait till System clock is ready */
177   while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
178   {
179
180   }
181   LL_SetSystemCoreClock(8000000);
182
183    /* Update the time base */
184   if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
185   {
186     Error_Handler();
187   }
188   LL_RCC_HSI14_EnableADCControl();
189 }
190
191 /**
192   * @brief ADC Initialization Function
193   * @param None
194   * @retval None
195   */
196 static void MX_ADC_Init(void)
197 {
198
199   /* USER CODE BEGIN ADC_Init 0 */
200   /* USER CODE END ADC_Init 0 */
201
202   ADC_ChannelConfTypeDef sConfig = {0};
203
204   /* USER CODE BEGIN ADC_Init 1 */
205
206   /* USER CODE END ADC_Init 1 */
207
208   /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number
    of conversion)
209   */
210   hadc.Instance = ADC1;
211   hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
212   hadc.Init.Resolution = ADC_RESOLUTION_12B;
213   hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
214   hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
215   hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
216   hadc.Init.LowPowerAutoWait = DISABLE;
217   hadc.Init.LowPowerAutoPowerOff = DISABLE;
218   hadc.Init.ContinuousConvMode = DISABLE;
219   hadc.Init.DiscontinuousConvMode = DISABLE;
220   hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
221   hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
222   hadc.Init.DMAContinuousRequests = DISABLE;
223   hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
224   if (HAL_ADC_Init(&hadc) != HAL_OK)
225   {
226     Error_Handler();
227   }
```

```
228
229   /** Configure for the selected ADC regular channel to be converted.
230   */
231   sConfig.Channel = ADC_CHANNEL_6;
232   sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
233   sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
234   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
235   {
236     Error_Handler();
237   }
238   /* USER CODE BEGIN ADC_Init 2 */
239   ADC1->CR |= ADC_CR_ADCAL;
240   while(ADC1->CR & ADC_CR_ADCAL);           // Calibrate the ADC
241   ADC1->CR |= (1 << 0);                     // Enable ADC
242   while((ADC1->ISR & (1 << 0)) == 0);       // Wait for ADC ready
243   /* USER CODE END ADC_Init 2 */
244
245 }
246
247 /**
248   * @brief TIM3 Initialization Function
249   * @param None
250   * @retval None
251   */
252 static void MX_TIM3_Init(void)
253 {
254
255   /* USER CODE BEGIN TIM3_Init 0 */
256
257   /* USER CODE END TIM3_Init 0 */
258
259   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
260   TIM_MasterConfigTypeDef sMasterConfig = {0};
261   TIM_OC_InitTypeDef sConfigOC = {0};
262
263   /* USER CODE BEGIN TIM3_Init 1 */
264
265   /* USER CODE END TIM3_Init 1 */
266   htim3.Instance = TIM3;
267   htim3.Init.Prescaler = 0;
268   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
269   htim3.Init.Period = 47999;
270   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
271   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
272   if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
273   {
274     Error_Handler();
275   }
276   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
277   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
278   {
279     Error_Handler();
280   }
281   if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
282   {
283     Error_Handler();
284   }
```

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

```
342   /**/
343   GPIO_InitStruct.Pin = LED7_Pin;
344   GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
345   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
346   GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
347   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
348   LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
349
350 /* USER CODE BEGIN MX_GPIO_Init_2 */
351   HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
352   HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
353 /* USER CODE END MX_GPIO_Init_2 */
354 }
355
356 /* USER CODE BEGIN 4 */
357 void EXTI0_1_IRQHandler(void)
358 {
359     // TODO: Add code to switch LED7 delay frequency
360
361     curr_millis = HAL_GetTick(); // GetTick fn gives us the current system time elapsed
362
363         if (__HAL_GPIO_EXTI_GET_IT(Button0_Pin) != 0)
364         {
365          if (curr_millis - prev_millis >= 150) // Checked to see if enough time has passed
    since the previous time the button was clicked
366          {
367             // Toggle the LED frequency between 1 Hz and 2 Hz
368              if (delay_t == 500)
369                 {
370                    delay_t = 1000; // 1000 delay equates to 1 Hz
371
372                 }
373              else
374                 {
375                    delay_t = 500;     // 500 delay equates to to 2 Hz
376                 }
377
378              prev_millis = curr_millis; // Save that old system time to be compared later
379              }
380         }
381
382     HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
383 }
384
385 // TODO: Complete the writeLCD function
386 void writeLCD(char *char_in){
387     delay(3000);
388     lcd_command(CLEAR);
389     lcd_putstring(char_in);
390 }
391
392 // Get ADC value
393 uint32_t pollADC(void){
394   // TODO: Complete function body to get ADC val
395
396         // We used the HAL Adc functions to start, convert and stop the adc
397
```

```
398         HAL_ADC_Start(&hadc);
399         HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
400         uint32_t val = HAL_ADC_GetValue(&hadc);
401         HAL_ADC_Stop(&hadc);
402 
403     return val;
404 }
405 
406 // Calculate PWM CCR value - Capture/Compare Register
407 uint32_t ADCtoCCR(uint32_t adc_val){
408   // TODO: Calculate CCR val using an appropriate equation
409 
410      // Since the ADC configured to 12-bit mode, input ADC integer is between 0-4095
411      // While the Capture/Compare Register requires value in range 0-ARR(max)
412      // to have control on the PWM duty cycle
413 
414     uint32_t ADCvalueRange = 4095;
415     uint32_t ARRvalueRange = 47999;
416 
417     // Corresponds to 1kHz frequency for PWM signal
418     // With Duty cycle = CCR/ARR
419 
420     // Equation to calculate appropriate CCR value
421     uint32_t val = (adc_val * ARRvalueRange) / ADCvalueRange;
422 
423     return val;
424 }
425 
426 void ADC1_COMP_IRQHandler(void)
427 {
428     adc_val = HAL_ADC_GetValue(&hadc); // read adc value
429     HAL_ADC_IRQHandler(&hadc); //Clear flags
430 }
431 /* USER CODE END 4 */
432 
433 /**
434   * @brief  This function is executed in case of error occurrence.
435   * @retval None
436   */
437 void Error_Handler(void)
438 {
439   /* USER CODE BEGIN Error_Handler_Debug */
440   /* User can add his own implementation to report the HAL error return state */
441   __disable_irq();
442   while (1)
443   {
444   }
445   /* USER CODE END Error_Handler_Debug */
446 }
447 
448 #ifdef  USE_FULL_ASSERT
449 /**
450   * @brief  Reports the name of the source file and the source line number
451   *         where the assert_param error has occurred.
452   * @param  file: pointer to the source file name
453   * @param  line: assert_param error line source number
454   * @retval None
```

```
455   */
456 void assert_failed(uint8_t *file, uint32_t line)
457 {
458   /* USER CODE BEGIN 6 */
459   /* User can add his own implementation to report the file name and line number,
460      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
461   /* USER CODE END 6 */
462 }
463 #endif /* USE_FULL_ASSERT */
464
```