```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; //for the debounce delay
uint32_t curr_millis = 0;
uint32_t delay_t = 500; // Initialise delay to 500ms
uint32_t adc_val;
char print_val[5];
/* USER CODE END PV */

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

```
 66 uint32_t ADCtoCCR(uint32_t adc_val);
 67 /* USER CODE END PFP */
 68
 69 /* Private user code ---------------------------------------------------------*/
 70 /* USER CODE BEGIN 0 */
 71
 72 /* USER CODE END 0 */
 73
 74 /**
 75   * @brief  The application entry point.
 76   * @retval int
 77   */
 78 int main(void)
 79 {
 80   /* USER CODE BEGIN 1 */
 81   /* USER CODE END 1 */
 82
 83   /* MCU Configuration--------------------------------------------------------*/
 84
 85   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 86   HAL_Init();
 87
 88   /* USER CODE BEGIN Init */
 89   /* USER CODE END Init */
 90
 91   /* Configure the system clock */
 92   SystemClock_Config();
 93
 94   /* USER CODE BEGIN SysInit */
 95   /* USER CODE END SysInit */
 96
 97   /* Initialize all configured peripherals */
 98   MX GPIO Init();
 99   MX_ADC_Init();
100   MX_TIM3_Init();
101
102   /* USER CODE BEGIN 2 */
103   init_LCD();
104   // PWM setup
105   uint32_t CCR = 0;
106   HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); // Start PWM on TIM3 Channel 3
107   /* USER CODE END 2 */
108
109   /* Infinite loop */
110   /* USER CODE BEGIN WHILE */
111   while (1)
112   {
113     // Toggle LED0
114     //HAL_GPIO_WritePin(GPIOB, LED7_Pin, GPIO_PIN_SET);
115     HAL GPIO TogglePin(GPIOB, LED7 Pin);
116
117     // ADC to LCD; TODO: Read POT1 value and write to LCD
118     pollADC(); //update print_val to last ADC value
119     writeLCD(print_val); //send print_val to LCD
120
121
122     // Update PWM value; TODO: Get CRR
123     CCR = ADCtoCCR(adc_val); //update CRR to converted adc_val
124
125     __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, CCR);
126
127     // Wait for delay ms
128     HAL_Delay (delay_t);
129     /* USER CODE END WHILE */
130
```

```
131     /* USER CODE BEGIN 3 */
132   }
133   /* USER CODE END 3 */
134 }
135
136 /**
137   * @brief System Clock Configuration
138   * @retval None
139   */
140 void SystemClock_Config(void)
141 {
142   LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);
143   while(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_0)
144   {
145   }
146   LL_RCC_HSI_Enable();
147
148    /* Wait till HSI is ready */
149   while(LL_RCC_HSI_IsReady() != 1)
150   {
151
152   }
153   LL_RCC_HSI_SetCalibTrimming(16);
154   LL_RCC_HSI14_Enable();
155
156    /* Wait till HSI14 is ready */
157   while(LL_RCC_HSI14_IsReady() != 1)
158   {
159
160   }
161   LL_RCC_HSI14_SetCalibTrimming(16);
162   LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
163   LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
164   LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
165
166    /* Wait till System clock is ready */
167   while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
168   {
169
170   }
171   LL_SetSystemCoreClock(8000000);
172
173    /* Update the time base */
174   if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
175   {
176     Error_Handler();
177   }
178   LL_RCC_HSI14_EnableADCControl();
179 }
180
181 /**
182   * @brief ADC Initialization Function
183   * @param None
184   * @retval None
185   */
186 static void MX_ADC_Init(void)
187 {
188
189   /* USER CODE BEGIN ADC_Init 0 */
190   /* USER CODE END ADC_Init 0 */
191
192   ADC_ChannelConfTypeDef sConfig = {0};
193
194   /* USER CODE BEGIN ADC_Init 1 */
195
```

```
196   /* USER CODE END ADC_Init 1 */
197
198   /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and
    number of conversion)
199   */
200   hadc.Instance = ADC1;
201   hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
202   hadc.Init.Resolution = ADC_RESOLUTION_12B;
203   hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
204   hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
205   hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
206   hadc.Init.LowPowerAutoWait = DISABLE;
207   hadc.Init.LowPowerAutoPowerOff = DISABLE;
208   hadc.Init.ContinuousConvMode = DISABLE;
209   hadc.Init.DiscontinuousConvMode = DISABLE;
210   hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
211   hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
212   hadc.Init.DMAContinuousRequests = DISABLE;
213   hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
214   if (HAL_ADC_Init(&hadc) != HAL_OK)
215   {
216     Error_Handler();
217   }
218
219   /** Configure for the selected ADC regular channel to be converted.
220   */
221   sConfig.Channel = ADC_CHANNEL_6;
222   sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
223   sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
224   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
225   {
226     Error_Handler();
227   }
228   /* USER CODE BEGIN ADC_Init 2 */
229   ADC1->CR |= ADC_CR_ADCAL;
230   while(ADC1->CR & ADC_CR_ADCAL);           // Calibrate the ADC
231   ADC1->CR |= (1 << 0);                     // Enable ADC
232   while((ADC1->ISR & (1 << 0)) == 0);       // Wait for ADC ready
233   /* USER CODE END ADC_Init 2 */
234
235 }
236
237 /**
238   * @brief TIM3 Initialization Function
239   * @param None
240   * @retval None
241   */
242 static void MX_TIM3_Init(void)
243 {
244
245   /* USER CODE BEGIN TIM3_Init 0 */
246
247   /* USER CODE END TIM3_Init 0 */
248
249   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
250   TIM_MasterConfigTypeDef sMasterConfig = {0};
251   TIM_OC_InitTypeDef sConfigOC = {0};
252
253   /* USER CODE BEGIN TIM3_Init 1 */
254
255   /* USER CODE END TIM3_Init 1 */
256   htim3.Instance = TIM3;
257   htim3.Init.Prescaler = 0;
258   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
259   htim3.Init.Period = 47999;
```

```
260   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
261   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
262   if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
263   {
264     Error_Handler();
265   }
266   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
267   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
268   {
269     Error_Handler();
270   }
271   if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
272   {
273     Error_Handler();
274   }
275   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
276   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
277   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
278   {
279     Error_Handler();
280   }
281   sConfigOC.OCMode = TIM_OCMODE_PWM1;
282   sConfigOC.Pulse = 0;
283   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
284   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
285   if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
286   {
287     Error_Handler();
288   }
289   /* USER CODE BEGIN TIM3_Init 2 */
290
291   /* USER CODE END TIM3_Init 2 */
292   HAL_TIM_MspPostInit(&htim3);
293
294 }
295
296 /**
297   * @brief GPIO Initialization Function
298   * @param None
299   * @retval None
300   */
301 static void MX_GPIO_Init(void)
302 {
303   LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
304   LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
305 /* USER CODE BEGIN MX_GPIO_Init_1 */
306 /* USER CODE END MX_GPIO_Init_1 */
307
308   /* GPIO Ports Clock Enable */
309   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
310   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
311   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
312
313   /**/
314   LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
315
316   /**/
317   LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);
318
319   /**/
320   LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);
321
322   /**/
323   LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);
324
```

```
325   /**/
326   EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
327   EXTI_InitStruct.LineCommand = ENABLE;
328   EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
329   EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
330   LL_EXTI_Init(&EXTI_InitStruct);
331
332   /**/
333   GPIO_InitStruct.Pin = LED7_Pin;
334   GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
335   GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
336   GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
337   GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
338   LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
339
340 /* USER CODE BEGIN MX_GPIO_Init_2 */
341   HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
342   HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
343 /* USER CODE END MX_GPIO_Init_2 */
344 }
345
346 /* USER CODE BEGIN 4 */
347 void EXTI0_1_IRQHandler(void)
348 {
349     // TODO: Add code to switch LED7 delay frequency
350     //switch between 1Hz and 2Hz, will only switch if there is more than 100 mils
    difference, creating a debounce delay
351     curr_millis = HAL_GetTick();
352     if (curr_millis > prev_millis + 100){
353         if (delay_t == 500){
354             delay_t = 1000;
355         }
356         else{
357             delay_t = 500;
358         }
359     }
360
361     prev_millis = curr_millis;
362     HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
363 }
364
365 // TODO: Complete the writeLCD function
366 //send char_in parameter to LCD
367 void writeLCD(char *char_in){
368     delay(3000);
369     lcd_command(CLEAR);
370     lcd_putstring(char_in);
371 }
372
373 // Get ADC value
374 uint32_t pollADC(void){
375   // TODO: Complete function body to get ADC val
376     HAL_ADC_Start(&hadc);
377     HAL_ADC_PollForConversion(&hadc, 100);
378     ADC1_COMP_IRQHandler();
379     HAL_ADC_Stop(&hadc);
380     sprintf(print_val, "%d", adc_val); //convert adc_val to a string and save to
    print val
381 }
382
383 // Calculate PWM CCR value
384 uint32_t ADCtoCCR(uint32_t adc_val){
385   // TODO: Calculate CCR val using an appropriate equation
386     uint32_t val = adc_val*47999/4095;
387     return val;
```

```
388 }
389
390 void ADC1_COMP_IRQHandler(void)
391 {
392     adc_val = HAL_ADC_GetValue(&hadc); // read adc value
393     HAL_ADC_IRQHandler(&hadc); //Clear flags
394 }
395 /* USER CODE END 4 */
396
397 /**
398   * @brief  This function is executed in case of error occurrence.
399   * @retval None
400   */
401 void Error_Handler(void)
402 {
403   /* USER CODE BEGIN Error_Handler_Debug */
404   /* User can add his own implementation to report the HAL error return state */
405   __disable_irq();
406   while (1)
407   {
408   }
409   /* USER CODE END Error_Handler_Debug */
410 }
411
412 #ifdef  USE_FULL_ASSERT
413 /**
414   * @brief  Reports the name of the source file and the source line number
415   *         where the assert_param error has occurred.
416   * @param  file: pointer to the source file name
417   * @param  line: assert_param error line source number
418   * @retval None
419   */
420 void assert_failed(uint8_t *file, uint32_t line)
421 {
422   /* USER CODE BEGIN 6 */
423   /* User can add his own implementation to report the file name and line number,
424      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
425   /* USER CODE END 6 */
426 }
427 #endif /* USE_FULL_ASSERT */
428
```