```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;
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
105   // PWM setup
106   uint32_t CCR = 0;
107   HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); // Start PWM on TIM3 Channel 3
108   /* USER CODE END 2 */
109
110   /* Infinite loop */
111   /* USER CODE BEGIN WHILE */
112   while (1)
113   {
114     // Toggle LED0
115     //HAL_GPIO_WritePin(GPIOB, LED7_Pin, GPIO_PIN_SET);
116     HAL_GPIO_TogglePin(GPIOB, LED7_Pin);
117
118     // ADC to LCD; TODO: Read POT1 value and write to LCD
119
120     pollADC(); //obtain reading from ADC and update print_val
121     writeLCD(print_val); //send ADC value saved as print_val to LCD
122
123
124     // Update PWM value; TODO: Get CRR
125     CCR = ADCtoCCR(adc_val); //set CCR value
126
127     __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, CCR);
128
129     // Wait for delay ms
130     HAL_Delay (delay_t);
```

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

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

```
260   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
261   htim3.Init.Period = 47999;
262   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
263   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
264   if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
265   {
266     Error_Handler();
267   }
268   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
269   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
270   {
271     Error_Handler();
272   }
273   if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
274   {
275     Error_Handler();
276   }
277   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
278   sMasterConfig.MasterSlaveMode = TIM MASTERSLAVEMODE DISABLE;
279   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
280   {
281     Error_Handler();
282   }
283   sConfigOC.OCMode = TIM_OCMODE_PWM1;
284   sConfigOC.Pulse = 0;
285   sConfigOC.OCPolarity = TIM OCPOLARITY HIGH;
286   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
287   if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
288   {
289     Error_Handler();
290   }
291   /* USER CODE BEGIN TIM3_Init 2 */
292
293   /* USER CODE END TIM3_Init 2 */
294   HAL_TIM_MspPostInit(&htim3);
295
296 }
297
298 /**
299   * @brief GPIO Initialization Function
300   * @param None
301   * @retval None
302   */
303 static void MX_GPIO_Init(void)
304 {
305   LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
306   LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
307 /* USER CODE BEGIN MX_GPIO_Init_1 */
308 /* USER CODE END MX_GPIO_Init_1 */
309
310   /* GPIO Ports Clock Enable */
311   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
312   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
313   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
314
315   /**/
316   LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
317
318   /**/
319   LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);
320
321   /**/
322   LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);
323
324   /**/
```

```
325   LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);
326
327   /**/
328   EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
329   EXTI_InitStruct.LineCommand = ENABLE;
330   EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
331   EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
332   LL_EXTI_Init(&EXTI_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 /* USER CODE BEGIN MX_GPIO_Init_2 */
343   HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
344   HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
345 /* USER CODE END MX_GPIO_Init_2 */
346 }
347
348 /* USER CODE BEGIN 4 */
349 void EXTI0_1_IRQHandler(void)
350 {
351     // TODO: Add code to switch LED7 delay frequency
352     if (delay_t == 500){ //switch between delay times to change LED toggling frequency
353         delay_t = 250;
354     }
355     else{
356         delay_t = 500;
357     }
358     HAL GPIO EXTI IRQHandler(Button0 Pin); // Clear interrupt flags
359 }
360
361 // TODO: Complete the writeLCD function
362 void writeLCD(char *char_in){
363     delay(3000);
364     lcd_command(CLEAR);
365     lcd_putstring(char_in); //send char_in parameter to LCD
366 }
367
368 // Get ADC value
369 uint32_t pollADC(void){
370   // TODO: Complete function body to get ADC val
371     HAL_ADC_Start(&hadc);
372     HAL_ADC_PollForConversion(&hadc, 100); //poll ADC for value
373     ADC1_COMP_IRQHandler(); //obtain last value
374     HAL_ADC_Stop(&hadc);
375     sprintf(print_val, "%d", adc_val); //save adc_val as a string to print_val
376 }
377
378 // Calculate PWM CCR value
379 uint32_t ADCtoCCR(uint32_t adc_val){
380   // TODO: Calculate CCR val using an appropriate equation
381     uint32_t val = adc_val*47999/4095;
382     return val;
383 }
384
385 void ADC1_COMP_IRQHandler(void)
386 {
387     adc_val = HAL_ADC_GetValue(&hadc); // read adc value
388     HAL_ADC_IRQHandler(&hadc); //Clear flags
389 }
```

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