Practical 2 - Creating a DAC

Lawrence Hawke† and Zainodine Dawood‡
EEE3096S Class of 2024
University of Cape Town
South Africa
†HWKLAW001 ‡DWDZAI004

***Abstract*—A digital to analogue converter is an essential part of electronics, whether one wants to listen to music, watch TV, use communication systems or perform data aquisition using scientific instruments. You are going to need a DAC (digital to analogue converter).**
In this practical we will be building a DAC using the pulse width modulation functionality of a micro controller and a passive low pass filter.
Below you will find the git hub link.

GIT HUB LINK:

https://github.com/Lawrenceismyname/EEE3096Spracs

I. Introduction

The purpose of this practical is to create a Digital to Analogue Converter using our ST Micro controller. Using the STM cube IDE and the HAL libraries we coded our STM development board to output analogue voltages. This practical involves creating 3 LUTs (lookup tables) for a sine, saw and triangular wave. These LUTs contain 128 values ranging from 0 to 1023. When an input is 1023 it will output high 3.3V. When the input goes lower it will output a lower voltage. This is the DAC functionality we will be providing.

II. Methodology

A. Hardware

We used the STM32 development board and a passive low pass filter, made up of 150nF and 1.5k Ω

B. Implementation

MATLAB code:

```
% Define the number of samples
num_samples = 128;

% Define the range of the LUT values
max_value = 1023;

% Generate the sinusoidal LUT
x = linspace(0, 2*pi, num_samples);
sin_lut = round((sin(x) + 1) * (max_value / 2));

% Generate the sawtooth wave LUT
saw_lut = round(linspace(0, max_value, num_samples));

% Generate the triangular wave LUT
tri_lut = round([2 * (0:num_samples/2-1) * (max_value / (num_samples - 1)), 2 *
(num_samples/2-1:-1:0) * (max_value / (num_samples - 1))]);

% Plot the LUTs
figure;

% Plot Sinusoidal LUT
subplot(3,1,1);
plot(sin_lut, '-o');
title('Sinusoidal LUT');
xlabel('Index');
ylabel('Value');
grid on;

% Plot Sawtooth LUT
subplot(3,1,2);
plot(saw_lut, '-o');
title('Sawtooth LUT');
xlabel('Index');
ylabel('Value');
grid on;

% Plot Triangular LUT
subplot(3,1,3);
plot(tri_lut, '-o');
title('Triangular LUT');
xlabel('Index');
ylabel('Value');
grid on;

% Output LUTs in C array format
fprintf('Sinusoidal LUT:\n');
fprintf('const uint16_t sin_lut[%d] = {', num_samples);
fprintf('%d, ', sin_lut(1:end));

fprintf('Sawtooth LUT:\n');
fprintf('const uint16_t saw_lut[%d] = {', num_samples);
fprintf('%d, ', saw_lut(1:end));

fprintf('Triangular LUT:\n');
fprintf('const uint16_t tri_lut[%d] = {', num_samples);
fprintf('%d, ', tri_lut(1:end));
```

This MATLAB code generates the sine, saw and triangular look up tables we use to demonstrate the functionality of our DAC.

C. Experiment Procedure

The functionality of our DAC was reached by modulating the pulse width of a pulse signal. It does this by varying the duty cycle. It will set the duty cycle to a percentage. This percentage is the value at hand (x) from 0 - 1023 divided by 1023.
duty cycle = $\frac{x}{1023}$ This duty cycle will correspond to an analogue voltage. Example: If the value is 512 the duty cycle will be roughly 50%, this will result in an average analogue output over the time period specified by $\frac{1}{F\_SIGNAL}$ of 1.15V.

We did this to prove that even with the low resolution of the pulse width module we could still get DAC functionality

by passing a pulse width modulated square wave through a low pass filter.

III. Results

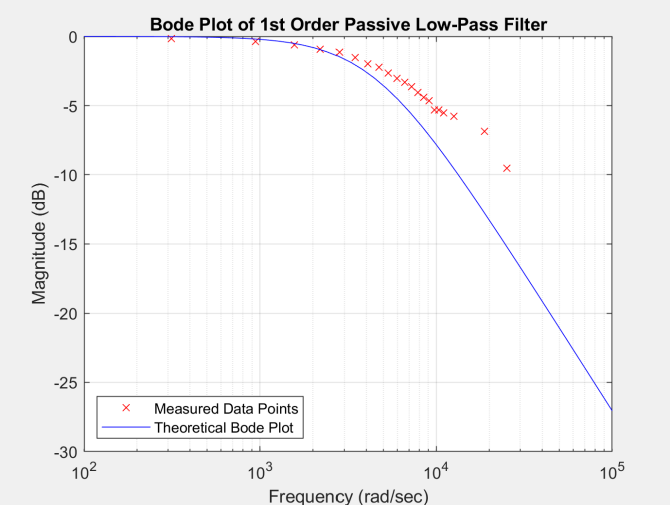

Fig. 1: Bode Plot of Low Pass Filter

The corner frequency is $\omega = 4444 \frac{rad}{sec}$

IV. Conclusion

We demonstrated that DAC functionality can be achieved by modulating the pulse width of a signal, varying the duty cycle based on a given value from 0 to 1023. This practical involved setting the duty cycle as a percentage of the maximum value, resulting in a analog voltage output corresponding to the unique input. Despite the low resolution of the pulse width module, we successfully proved that by passing the pulse width modulated square wave through a low pass filter, it is possible to obtain a usable analog signal.

minted

```

/* 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 */
// TODO: Add values for below variables
#define NS 128       // Number of samples in LUT
#define TIM2CLK  8000000 // STM Clock frequency
#define F_SIGNAL  3200 // Frequency of output analog signal
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
DMA_HandleTypeDef hdma_tim2_ch1;

/* USER CODE BEGIN PV */
```

```
// TODO: Add code for global variables, including LUTs

uint32_t Sin_LUT[NS] = {512, 537, 562, 587, 612, 637, 661, 685, 709, 732, 754, 776,
 ↪  798, 818, 838, 857, 875, 893, 909, 925, 939, 952, 965, 976, 986, 995, 1002, 1009,
 ↪  1014, 1018, 1021, 1023, 1023, 1022, 1020, 1016, 1012, 1006, 999, 990, 981, 970,
 ↪  959, 946, 932, 917, 901, 884, 866, 848, 828, 808, 787, 765, 743, 720, 697, 673,
 ↪  649, 624, 600, 575, 549, 524, 499, 474, 448, 423, 399, 374, 350, 326, 303, 280,
 ↪  258, 236, 215, 195, 175, 157, 139, 122, 106, 91, 77, 64, 53, 42, 33, 24, 17, 11,
 ↪  7, 3, 1, 0, 0, 2, 5, 9, 14, 21, 28, 37, 47, 58, 71, 84, 98, 114, 130, 148, 166,
 ↪  185, 205, 225, 247, 269, 291, 314, 338, 362, 386, 411, 436, 461, 486, 511};
uint32_t saw_LUT[NS] = {0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 81, 89, 97, 105, 113,
 ↪  121, 129, 137, 145, 153, 161, 169, 177, 185, 193, 201, 209, 217, 226, 234, 242,
 ↪  250, 258, 266, 274, 282, 290, 298, 306, 314, 322, 330, 338, 346, 354, 362, 371,
 ↪  379, 387, 395, 403, 411, 419, 427, 435, 443, 451, 459, 467, 475, 483, 491, 499,
 ↪  507, 516, 524, 532, 540, 548, 556, 564, 572, 580, 588, 596, 604, 612, 620, 628,
 ↪  636, 644, 652, 661, 669, 677, 685, 693, 701, 709, 717, 725, 733, 741, 749, 757,
 ↪  765, 773, 781, 789, 797, 806, 814, 822, 830, 838, 846, 854, 862, 870, 878, 886,
 ↪  894, 902, 910, 918, 926, 934, 942, 951, 959, 967, 975, 983, 991, 999, 1007, 1015,
 ↪  1023};
uint32_t triangle_LUT[NS] = {0, 16, 32, 48, 64, 81, 97, 113, 129, 145, 161, 177, 193,
 ↪  209, 226, 242, 258, 274, 290, 306, 322, 338, 354, 371, 387, 403, 419, 435, 451,
 ↪  467, 483, 499, 516, 532, 548, 564, 580, 596, 612, 628, 644, 661, 677, 693, 709,
 ↪  725, 741, 757, 773, 789, 806, 822, 838, 854, 870, 886, 902, 918, 934, 951, 967,
 ↪  983, 999, 1015, 1015, 999, 983, 967, 951, 934, 918, 902, 886, 870, 854, 838, 822,
 ↪  806, 789, 773, 757, 741, 725, 709, 693, 677, 661, 644, 628, 612, 596, 580, 564,
 ↪  548, 532, 516, 499, 483, 467, 451, 435, 419, 403, 387, 371, 354, 338, 322, 306,
 ↪  290, 274, 258, 242, 226, 209, 193, 177, 161, 145, 129, 113, 97, 81, 64, 48, 32,
 ↪  16, 0};

// TODO: Equation to calculate TIM2_Ticks

uint32_t TIM2_Ticks = TIM2CLK / (F_SIGNAL * NS); // How often to write new LUT value
uint32_t DestAddress = (uint32_t) &(TIM3->CCR3); // Write LUT TO TIM3->CCR3 to modify
 ↪  PWM duty cycle
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */
void EXTI0_1_IRQHandler(void);
/* 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)
```

```c
{

  /* 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_DMA_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */

  // TODO: Start TIM3 in PWM mode on channel 3
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);

  // TODO: Start TIM2 in Output Compare (OC) mode on channel 1.
  HAL_TIM_OC_Start(&htim2, TIM_CHANNEL_1);

  // TODO: Start DMA in IT mode on TIM2->CH1; Source is LUT and Dest is TIM3->CCR3;
   ↪  start with Sine LUT
  HAL_DMA_Start_IT(&hdma_tim2_ch1, (uint32_t)Sin_LUT, DestAddress, NS);

  // TODO: Write current waveform to LCD ("Sine")
  delay(3000);
  init_LCD();
  lcd_putstring("Sine");

  // TODO: Enable DMA (start transfer from LUT to CCR)
  __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

```
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_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
161   LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
162   LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSI);
163
164    /* Wait till System clock is ready */
165   while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSI)
166   {
167
168   }
169   LL_SetSystemCoreClock(8000000);
170
171    /* Update the time base */
172   if (HAL_InitTick (TICK_INT_PRIORITY) != HAL_OK)
173   {
174     Error_Handler();
175   }
176 }
177
178 /**
179   * @brief TIM2 Initialization Function
180   * @param None
181   * @retval None
182   */
183 static void MX_TIM2_Init(void)
184 {
185
186   /* USER CODE BEGIN TIM2_Init 0 */
187
188   /* USER CODE END TIM2_Init 0 */
189
190   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
191   TIM_MasterConfigTypeDef sMasterConfig = {0};
192   TIM_OC_InitTypeDef sConfigOC = {0};
193
194   /* USER CODE BEGIN TIM2_Init 1 */
195
196   /* USER CODE END TIM2_Init 1 */
```

```
197   htim2.Instance = TIM2;
198   htim2.Init.Prescaler = 0;
199   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
200   htim2.Init.Period = TIM2_Ticks - 1;
201   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
202   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
203   if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
204   {
205     Error_Handler();
206   }
207   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
208   if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
209   {
210     Error_Handler();
211   }
212   if (HAL_TIM_OC_Init(&htim2) != HAL_OK)
213   {
214     Error_Handler();
215   }
216   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
217   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
218   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
219   {
220     Error_Handler();
221   }
222   sConfigOC.OCMode = TIM_OCMODE_TIMING;
223   sConfigOC.Pulse = 0;
224   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
225   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
226   if (HAL_TIM_OC_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
227   {
228     Error_Handler();
229   }
230   /* USER CODE BEGIN TIM2_Init 2 */
231
232   /* USER CODE END TIM2_Init 2 */
233
234 }
235
236 /**
237   * @brief TIM3 Initialization Function
238   * @param None
239   * @retval None
240   */
241 static void MX_TIM3_Init(void)
242 {
243
244   /* USER CODE BEGIN TIM3_Init 0 */
245
246   /* USER CODE END TIM3_Init 0 */
247
248   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
249   TIM_MasterConfigTypeDef sMasterConfig = {0};
250   TIM_OC_InitTypeDef sConfigOC = {0};
251
252   /* USER CODE BEGIN TIM3_Init 1 */
```

```
253
254   /* USER CODE END TIM3_Init 1 */
255   htim3.Instance = TIM3;
256   htim3.Init.Prescaler = 0;
257   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
258   htim3.Init.Period = 1023;
259   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
260   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
261   if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
262   {
263     Error_Handler();
264   }
265   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
266   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
267   {
268     Error_Handler();
269   }
270   if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
271   {
272     Error_Handler();
273   }
274   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
275   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
276   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
277   {
278     Error_Handler();
279   }
280   sConfigOC.OCMode = TIM_OCMODE_PWM1;
281   sConfigOC.Pulse = 0;
282   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
283   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
284   if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
285   {
286     Error_Handler();
287   }
288   /* USER CODE BEGIN TIM3_Init 2 */
289
290   /* USER CODE END TIM3_Init 2 */
291   HAL_TIM_MspPostInit(&htim3);
292
293 }
294
295 /**
296   * Enable DMA controller clock
297   */
298 static void MX_DMA_Init(void)
299 {
300
301   /* DMA controller clock enable */
302   __HAL_RCC_DMA1_CLK_ENABLE();
303
304   /* DMA interrupt init */
305   /* DMA1_Channel4_5_IRQn interrupt configuration */
306   HAL_NVIC_SetPriority(DMA1_Channel4_5_IRQn, 0, 0);
307   HAL_NVIC_EnableIRQ(DMA1_Channel4_5_IRQn);
308
```

```
309 }
310
311 /**
312   * @brief GPIO Initialization Function
313   * @param None
314   * @retval None
315   */
316 static void MX_GPIO_Init(void)
317 {
318   LL_EXTI_InitTypeDef EXTI_InitStruct = {0};
319 /* USER CODE BEGIN MX_GPIO_Init_1 */
320 /* USER CODE END MX_GPIO_Init_1 */
321
322   /* GPIO Ports Clock Enable */
323   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOF);
324   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
325   LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
326
327   /**/
328   LL_SYSCFG_SetEXTISource(LL_SYSCFG_EXTI_PORTA, LL_SYSCFG_EXTI_LINE0);
329
330   /**/
331   LL_GPIO_SetPinPull(Button0_GPIO_Port, Button0_Pin, LL_GPIO_PULL_UP);
332
333   /**/
334   LL_GPIO_SetPinMode(Button0_GPIO_Port, Button0_Pin, LL_GPIO_MODE_INPUT);
335
336   /**/
337   EXTI_InitStruct.Line_0_31 = LL_EXTI_LINE_0;
338   EXTI_InitStruct.LineCommand = ENABLE;
339   EXTI_InitStruct.Mode = LL_EXTI_MODE_IT;
340   EXTI_InitStruct.Trigger = LL_EXTI_TRIGGER_RISING;
341   LL_EXTI_Init(&EXTI_InitStruct);
342
343 /* USER CODE BEGIN MX_GPIO_Init_2 */
344   HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
345   HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
346 /* USER CODE END MX_GPIO_Init_2 */
347 }
348
349 /* USER CODE BEGIN 4 */
350 void EXTI0_1_IRQHandler(void)
351 {
352         // TODO: Debounce using HAL_GetTick()
353         static uint32_t PrevPush = 0;
354         uint32_t NextPush = HAL_GetTick();
355
356
357
358         // TODO: Disable DMA transfer and abort IT, then start DMA in IT mode with new
        ↪  LUT and re-enable transfer
359         // HINT: Consider using C's "switch" function to handle LUT changes
360         if ((NextPush - PrevPush) > 200) {  // Debounce (200ms delay)
361                 static int currentWave = 0;
362                 __HAL_TIM_DISABLE_DMA(&htim2, TIM_DMA_CC1);
363                 HAL_DMA_Abort_IT(&hdma_tim2_ch1); // Stop DMA transfer
```

```
364
365
366                 // Cycle through the waveforms
367                 switch (currentWave) {
368                         case 0:
369                                 delay(3000);
370                                 lcd_command(CLEAR);
371                                 lcd_putstring("Sawtooth");
372                                 HAL_DMA_Start_IT(&hdma_tim2_ch1, (uint32_t)saw_LUT,
                                    ↪  DestAddress, NS);
373                                 __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
374                                 currentWave = 1;
375                                 break;
376
377                           case 1:
378                                   delay(3000);
379                                   lcd_command(CLEAR);
380                                   lcd_putstring("Triangle");
381                                   HAL_DMA_Start_IT(&hdma_tim2_ch1,
                                      ↪  (uint32_t)triangle_LUT, DestAddress, NS);
382                                   __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
383                                   currentWave = 2;
384                           break;
385
386       default:
387                                 delay(3000);
388                                 lcd_command(CLEAR);
389                                 lcd_putstring("Sine");
390                                 HAL_DMA_Start_IT(&hdma_tim2_ch1, (uint32_t)Sin_LUT,
                                    ↪  DestAddress, NS);
391                                 __HAL_TIM_ENABLE_DMA(&htim2, TIM_DMA_CC1);
392                                 currentWave = 0;
393                                 break;
394
395                 }
396         }
397
398         PrevPush = NextPush;
399
400
401         HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
402 }
403 /* USER CODE END 4 */
404
405 /**
406   * @brief  This function is executed in case of error occurrence.
407   * @retval None
408   */
409 void Error_Handler(void)
410 {
411   /* USER CODE BEGIN Error_Handler_Debug */
412   /* User can add his own implementation to report the HAL error return state */
413   __disable_irq();
414   while (1)
415   {
416   }
```

```
417   /* USER CODE END Error_Handler_Debug */
418 }
419
420 #ifdef  USE_FULL_ASSERT
421 /**
422   * @brief  Reports the name of the source file and the source line number
423   *         where the assert_param error has occurred.
424   * @param  file: pointer to the source file name
425   * @param  line: assert_param error line source number
426   * @retval None
427   */
428 void assert_failed(uint8_t *file, uint32_t line)
429 {
430   /* USER CODE BEGIN 6 */
431   /* User can add his own implementation to report the file name and line number,
432      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
433   /* USER CODE END 6 */
434 }
435 #endif /* USE_FULL_ASSERT */
436
```