UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Universitat Politècnica de Catalunya

Motion Control and Industrial Applications

Estudio, diseño e implementación de un sistema de monitorización basado en dron

Anexo

Autor:
Gorka Unzueta

Director:
Dr. Miguel Delgado

Co-Director:
Dr. Luis Romeral

Trabajo de final de grado para

Grau en Enginyeria en Vehicles Aeroespacials

10 de junio de 2019

Índice general

A | Anexos

A.1. main.c del microprocesador del nodo

```
  1
  2  /**
  3    ******************************************************************************
  4    * @file           : main.c
  5    * @brief          : Main program body
  6    ******************************************************************************
  7    * This notice applies to any and all portions of this file
  8    * that are not between comment pairs USER CODE BEGIN and
  9    * USER CODE END. Other portions of this file, whether
 10    * inserted by the user or by software development tools
 11    * are owned by their respective copyright owners.
 12    *
 13    * Copyright (c) 2019 STMicroelectronics International N.V.
 14    * All rights reserved.
 15    *
 16    * Redistribution and use in source and binary forms, with or without
 17    * modification, are permitted, provided that the following conditions are met:
 18    *
 19    * 1. Redistribution of source code must retain the above copyright notice,
 20    *    this list of conditions and the following disclaimer.
 21    * 2. Redistributions in binary form must reproduce the above copyright notice,
 22    *    this list of conditions and the following disclaimer in the documentation
 23    *    and/or other materials provided with the distribution.
 24    * 3. Neither the name of STMicroelectronics nor the names of other
 25    *    contributors to this software may be used to endorse or promote products
 26    *    derived from this software without specific written permission.
 27    * 4. This software, including modifications and/or derivative works of this
 28    *    software, must execute solely and exclusively on microcontroller or
 29    *    microprocessor devices manufactured by or for STMicroelectronics.
 30    * 5. Redistribution and use of this software other than as permitted under
 31    *    this license is void and will automatically terminate your rights under
 32    *    this license.
 33    *
 34    * THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS"
 35    * AND ANY EXPRESS, IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT
 36    * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 37    * PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY
 38    * RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO EVENT
 39    * SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 40    * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 41    * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
 42    * OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 43    * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 44    * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 45    * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 46    *
 47    ******************************************************************************
 48    */
 49  /* Includes ------------------------------------------------------------------*/
 50  #include "main.h"
 51  #include "stm32l4xx_hal.h"
 52  #include "usb_host.h"
 53
 54  /* USER CODE BEGIN Includes */
 55  //HCI Packet types
 56  #define COMMAND   0x01
 57  #define ACL_DATA  0x02
 58  #define SCO_DATA  0x03
 59  #define EVENT     0x04
 60
 61    /* Unit: ms                                                                   */
 62    #define ADC_CALIBRATION_TIMEOUT_MS       ((uint32_t)   1)
 63    #define ADC_ENABLE_TIMEOUT_MS            ((uint32_t)   1)
 64    #define ADC_DISABLE_TIMEOUT_MS           ((uint32_t)   1)
 65    #define ADC_STOP_CONVERSION_TIMEOUT_MS   ((uint32_t)   1)
 66    #define ADC_CONVERSION_TIMEOUT_MS        ((uint32_t) 500)
 67
 68    /* Delay between ADC end of calibration and ADC enable.                       */
 69    /* Delay estimation in CPU cycles: Case of ADC enable done                    */
 70    /* immediately after ADC calibration, ADC clock setting slow                  */
 71    /* (LL_ADC_CLOCK_ASYNC_DIV32). Use a higher delay if ratio                    */
 72    /* (CPU clock / ADC clock) is above 32.                                       */
 73    #define ADC_DELAY_CALIB_ENABLE_CPU_CYCLES  (LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES * 32)
 74
 75  /* Definitions of environment analog values */
 76    /* Value of analog reference voltage (Vref+), connected to analog voltage   */
 77    /* supply Vdda (unit: mV).                                                    */
 78    #define VDDA_APPLI                       ((uint32_t)3300)
```

```
 79
 80  /* Definitions of data related to this example */
 81    /* Init variable out of expected ADC conversion data range */
 82    #define VAR_CONVERTED_DATA_INIT_VALUE    (__LL_ADC_DIGITAL_SCALE(LL_ADC_RESOLUTION_12B) + 1)
 83
 84  /* USER CODE END Includes */
 85
 86  /* Private variables ---------------------------------------------------------*/
 87  ADC_HandleTypeDef hadc1;
 88  DMA_HandleTypeDef hdma_adc1;
 89
 90  LPTIM_HandleTypeDef hlptim1;
 91
 92  UART_HandleTypeDef hlpuart1;
 93
 94  /* USER CODE BEGIN PV */
 95  /* Private variables ---------------------------------------------------------*/
 96  uint8_t RxBuffer[256]= " ";
 97  uint8_t TxBuffer[256]= " ";
 98  uint8_t Data[256] = "";
 99  uint16_t dmabuffer[250];
100  uint16_t adcbuffer;
101  uint16_t mainstate = 0;
102  /* USER CODE END PV */
103
104  /* Private function prototypes -----------------------------------------------*/
105  void SystemClock_Config(void);
106  static void MX_GPIO_Init(void);
107  static void MX_LPUART1_UART_Init(void);
108
109  static void MX_LPTIM1_Init(void);
110  void MX_USB_HOST_Process(void);
111
112  /* USER CODE BEGIN PFP */
113  /* Private function prototypes -----------------------------------------------*/
114  void go_to_sleep(void);
115
116  void     Configure_ADC3(void);
117  void     Configure_DMA(void);
118  void     Activate_ADC3(void);
119
120  void floattostring(float medida, char *medidachar, uint16_t currentmed);
121  void reverse(char *str, int len);
122  int intToStr2(int x, char str[],uint16_t currentmed,int pos);
123  int intToStr(int x, char str[], int d);
124  /* USER CODE END PFP */
125
126  /* USER CODE BEGIN 0 */
127
128  /* USER CODE END 0 */
129
130  /**
131    * @brief  The application entry point.
132    *
133    * @retval None
134    */
135  int main(void)
136  {
137    /* USER CODE BEGIN 1 */
138
139    /* USER CODE END 1 */
140
141    /* MCU Configuration--------------------------------------------------------*/
142
143    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
144    HAL_Init();
145
146    /* USER CODE BEGIN Init */
147
148    /* USER CODE END Init */
149
150    /* Configure the system clock */
151    SystemClock_Config();
152
153    /* USER CODE BEGIN SysInit */
154
155    /* USER CODE END SysInit */
156
```

```
157     /* Initialize all configured peripherals */
158     MX_GPIO_Init();
159
160     MX_LPUART1_UART_Init();
161     MX_USB_HOST_Init();
162
163     MX_LPTIM1_Init();
164     /* USER CODE BEGIN 2 */
165
166
167     //Preparamos el comando para resetear el bluetooth
168     TxBuffer[0] = 0x01; //PACKET TYPE BYTE
169     TxBuffer[1] = 0x03;     //OPCODE BYTE 1
170     TxBuffer[2] = 0x0C;     //OPCODE BYTE 2
171     /* USER CODE END 2 */
172
173     /* Infinite loop */
174     /* USER CODE BEGIN WHILE */
175     while (1)
176     {
177       //Reseteamos el bluetooth
178     HAL_UART_Transmit(&hlpuart1,TxBuffer,4,HAL_MAX_DELAY);
179     HAL_UART_Receive(&hlpuart1,RxBuffer,7,HAL_MAX_DELAY);
180
181     if(RxBuffer[0] == EVENT && RxBuffer[1] == 0x0E && RxBuffer[6]== 0x00){
182       //Si se ha reseteado correctamente habilitamos los eventos LE
183       // SET EVENT MASK PARA HABILITAR LOS EVENTOS LE
184       //              OCF               OGF               LENGTH                  B1
      B2                  B3                  B4                  B5
185       TxBuffer[1] = 0x01; TxBuffer[2] = 0x0C; TxBuffer[3] = 0x08; TxBuffer[4] = 0x00; TxBuffer[5] =
      0x00; TxBuffer[6] = 0x00; TxBuffer[7] = 0x00; TxBuffer[8] = 0x00;
186       //              B6                  B7                  B8
187       TxBuffer[9] = 0x00; TxBuffer[10] = 0x00; TxBuffer[11] = 0x20;
188
189       HAL_UART_Transmit(&hlpuart1,TxBuffer,12,HAL_MAX_DELAY);
190       HAL_UART_Receive(&hlpuart1,RxBuffer,7,HAL_MAX_DELAY);
191
192       if(RxBuffer[0] == EVENT && RxBuffer[1] == 0x0E && RxBuffer[4] == 0x01 && RxBuffer[5] == 0x0C &&
      RxBuffer[6]== 0x00){
193         //si se ha habilitado correctamente empezamos el advertisement
194         TxBuffer[1] = 0x0A;
195         TxBuffer[2] = 0x20;
196         TxBuffer[3] = 0x01;
197         TxBuffer[4] = 0x01;
198
199         HAL_UART_Transmit(&hlpuart1,TxBuffer,5,HAL_MAX_DELAY);
200         HAL_UART_Receive(&hlpuart1,RxBuffer,7,HAL_MAX_DELAY);
201
202         if(RxBuffer[0] == EVENT && RxBuffer[1] == 0x0E && RxBuffer[4] == 0x0A && RxBuffer[5] == 0x20
      && RxBuffer[6]== 0x00) {
203           //Esperamos a recibir el evento de conexion completada
204           HAL_UART_Receive_IT(&hlpuart1,RxBuffer,22);
205           if (HAL_LPTIM_Counter_Start_IT(&hlptim1, 1000) != HAL_OK)
206     {
207       Error_Handler();
208     }
209           go_to_sleep();
210         }
211       }
212     }
213
214     /* USER CODE END WHILE */
215       MX_USB_HOST_Process();
216
217     /* USER CODE BEGIN 3 */
218
219     }
220     /* USER CODE END 3 */
221
222   }
223
224   /**
225     * @brief System Clock Configuration
226     * @retval None
227     */
228   void SystemClock_Config(void)
229   {
230
```

```
231     RCC_OscInitTypeDef RCC_OscInitStruct;
232     RCC_ClkInitTypeDef RCC_ClkInitStruct;
233     RCC_PeriphCLKInitTypeDef PeriphClkInit;
234
235       /**Configure LSE Drive Capability
236       */
237     HAL_PWR_EnableBkUpAccess();
238
239     __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
240
241       /**Initializes the CPU, AHB and APB busses clocks
242       */
243     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
244     RCC_OscInitStruct.LSEState = RCC_LSE_ON;
245     RCC_OscInitStruct.MSIState = RCC_MSI_ON;
246     RCC_OscInitStruct.MSICalibrationValue = 0;
247     RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_9;
248     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
249     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
250     RCC_OscInitStruct.PLL.PLLM = 5;
251     RCC_OscInitStruct.PLL.PLLN = 71;
252     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
253     RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
254     RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV6;
255     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
256     {
257       _Error_Handler(__FILE__, __LINE__);
258     }
259
260       /**Initializes the CPU, AHB and APB busses clocks
261       */
262     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
263                                   |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
264     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
265     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV4;
266     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
267     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
268
269     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
270     {
271       _Error_Handler(__FILE__, __LINE__);
272     }
273
274     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_LPUART1|RCC_PERIPHCLK_LPTIM1
275                                   |RCC_PERIPHCLK_USB|RCC_PERIPHCLK_ADC;
276     PeriphClkInit.Lpuart1ClockSelection = RCC_LPUART1CLKSOURCE_LSE;
277     PeriphClkInit.Lptim1ClockSelection = RCC_LPTIM1CLKSOURCE_LSE;
278     PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_PLLSAI1;
279     PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLLSAI1;
280     PeriphClkInit.PLLSAI1.PLLSAI1Source = RCC_PLLSOURCE_MSI;
281     PeriphClkInit.PLLSAI1.PLLSAI1M = 5;
282     PeriphClkInit.PLLSAI1.PLLSAI1N = 20;
283     PeriphClkInit.PLLSAI1.PLLSAI1P = RCC_PLLP_DIV2;
284     PeriphClkInit.PLLSAI1.PLLSAI1Q = RCC_PLLQ_DIV2;
285     PeriphClkInit.PLLSAI1.PLLSAI1R = RCC_PLLR_DIV2;
286     PeriphClkInit.PLLSAI1.PLLSAI1ClockOut = RCC_PLLSAI1_48M2CLK|RCC_PLLSAI1_ADC1CLK;
287     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
288     {
289       _Error_Handler(__FILE__, __LINE__);
290     }
291
292       /**Configure the main internal regulator output voltage
293       */
294     if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
295     {
296       _Error_Handler(__FILE__, __LINE__);
297     }
298
299       /**Configure the Systick interrupt time
300       */
301     HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
302
303       /**Configure the Systick
304       */
305     HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
306
307       /**Enable MSI Auto calibration
308       */
```

```
309     HAL_RCCEx_EnableMSIPLLMode();
310
311     /* SysTick_IRQn interrupt configuration */
312     HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
313   }
314
315   /* ADC1 init function */
316   void Configure_ADC3(void)
317   {
318     __IO uint32_t wait_loop_index = 0;
319
320     NVIC_SetPriority(ADC3_IRQn, 1); /* ADC IRQ greater priority than DMA IRQ */
321     NVIC_EnableIRQ(ADC3_IRQn);
322     LL_ADC_InitTypeDef ADC_InitStruct;
323     LL_ADC_REG_InitTypeDef ADC_REG_InitStruct;
324
325     /* Peripheral clock enable */
326     LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_ADC);
327
328     /* Enable GPIO Clock for temperature sensor */
329     LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOA);
330     //El sensor de temperatura se encuentra en el pin GPIO PA.4
331     LL_GPIO_SetPinMode(GPIOA, LL_GPIO_PIN_4, LL_GPIO_MODE_ANALOG);
332     //Hay que definir el caminto interno al sensor de temperatura
333     LL_ADC_SetCommonPathInternalCh(__LL_ADC_COMMON_INSTANCE(ADC3), LL_ADC_PATH_INTERNAL_TEMPSENSOR);
334     //Hay que esperar por el sensor de temperatura
335     wait_loop_index = ((LL_ADC_DELAY_TEMPSENSOR_STAB_US * (SystemCoreClock / (100000 * 2))) / 10);
336       while(wait_loop_index != 0)
337       {
338         wait_loop_index--;
339       }
340
341       /**Common config */
342     ADC_InitStruct.Resolution = LL_ADC_RESOLUTION_12B;
343     ADC_InitStruct.DataAlignment = LL_ADC_DATA_ALIGN_RIGHT;
344     ADC_InitStruct.LowPowerMode = LL_ADC_LP_MODE_NONE;
345     LL_ADC_Init(ADC3, &ADC_InitStruct);
346
347     ADC_REG_InitStruct.TriggerSource = LL_ADC_REG_TRIG_SOFTWARE;
348     ADC_REG_InitStruct.SequencerLength = LL_ADC_REG_SEQ_SCAN_DISABLE;
349     ADC_REG_InitStruct.SequencerDiscont = LL_ADC_REG_SEQ_DISCONT_DISABLE;
350     ADC_REG_InitStruct.ContinuousMode = LL_ADC_REG_CONV_CONTINUOUS;
351     ADC_REG_InitStruct.DMATransfer = LL_ADC_REG_DMA_TRANSFER_UNLIMITED;
352     ADC_REG_InitStruct.Overrun = LL_ADC_REG_OVR_DATA_OVERWRITTEN;
353     LL_ADC_REG_Init(ADC3, &ADC_REG_InitStruct);
354
355     LL_ADC_SetOverSamplingScope(ADC3, LL_ADC_OVS_DISABLE);
356
357     LL_ADC_EnableIT_EOC(ADC3);
358
359     LL_ADC_DisableIT_EOS(ADC3);
360
361       /**Configure Regular Channel */
362     LL_ADC_REG_SetSequencerRanks(ADC3, LL_ADC_REG_RANK_1, LL_ADC_CHANNEL_TEMPSENSOR);
363
364     LL_ADC_SetChannelSamplingTime(ADC3, LL_ADC_CHANNEL_TEMPSENSOR, LL_ADC_SAMPLINGTIME_640CYCLES_5);
365
366   }
367
368   void Activate_ADC3(void)
369   {
370     __IO uint32_t wait_loop_index = 0;
371     #if (USE_TIMEOUT == 1)
372     uint32_t Timeout = 0; /* Variable used for timeout management */
373     #endif /* USE_TIMEOUT */
374
375     /*## Operation on ADC hierarchical scope: ADC instance ####################*/
376
377     /* Note: Hardware constraint (refer to description of the functions         */
378     /*       below):                                                            */
379     /*       On this STM32 serie, setting of these features is conditioned to   */
380     /*       ADC state:                                                         */
381     /*       ADC must be disabled.                                              */
382     /* Note: In this example, all these checks are not necessary but are        */
383     /*       implemented anyway to show the best practice usages                */
384     /*       corresponding to reference manual procedure.                       */
385     /*       Software can be optimized by removing some of these checks, if     */
386     /*       they are not relevant considering previous settings and actions    */
```

```
387     /*       in user application.                                          */
388     if (LL_ADC_IsEnabled(ADC3) == 0)
389     {
390       /* Disable ADC deep power down (enabled by default after reset state) */
391       LL_ADC_DisableDeepPowerDown(ADC3);
392 
393       /* Enable ADC internal voltage regulator */
394       LL_ADC_EnableInternalRegulator(ADC3);
395 
396       /* Delay for ADC internal voltage regulator stabilization.               */
397       /* Compute number of CPU cycles to wait for, from delay in us.           */
398       /* Note: Variable divided by 2 to compensate partially                   */
399       /*       CPU processing cycles (depends on compilation optimization).    */
400       /* Note: If system core clock frequency is below 200kHz, wait time       */
401       /*       is only a few CPU processing cycles.                            */
402       wait_loop_index = ((LL_ADC_DELAY_INTERNAL_REGUL_STAB_US * (SystemCoreClock / (100000 * 2))) / 10);
403       while(wait_loop_index != 0)
404       {
405         wait_loop_index--;
406       }
407 
408       /* Run ADC self calibration */
409       LL_ADC_StartCalibration(ADC3, LL_ADC_SINGLE_ENDED);
410 
411       /* Poll for ADC effectively calibrated */
412       #if (USE_TIMEOUT == 1)
413       Timeout = ADC_CALIBRATION_TIMEOUT_MS;
414       #endif /* USE_TIMEOUT */
415 
416       while (LL_ADC_IsCalibrationOnGoing(ADC3) != 0)
417       {
418       #if (USE_TIMEOUT == 1)
419         /* Check Systick counter flag to decrement the time-out value */
420         if (LL_SYSTICK_IsActiveCounterFlag())
421         {
422           if(Timeout-- == 0)
423           {
424           /* Time-out occurred. Set LED to blinking mode */
425           LED_Blinking(LED_BLINK_ERROR);
426           }
427         }
428       #endif /* USE_TIMEOUT */
429       }
430 
431       /* Delay between ADC end of calibration and ADC enable.                  */
432       /* Note: Variable divided by 2 to compensate partially                   */
433       /*       CPU processing cycles (depends on compilation optimization).    */
434       wait_loop_index = (ADC_DELAY_CALIB_ENABLE_CPU_CYCLES >> 1);
435       while(wait_loop_index != 0)
436       {
437         wait_loop_index--;
438       }
439 
440       /* Enable ADC */
441       LL_ADC_Enable(ADC3);
442 
443       /* Poll for ADC ready to convert */
444       #if (USE_TIMEOUT == 1)
445       Timeout = ADC_ENABLE_TIMEOUT_MS;
446       #endif /* USE_TIMEOUT */
447 
448       while (LL_ADC_IsActiveFlag_ADRDY(ADC3) == 0)
449       {
450       #if (USE_TIMEOUT == 1)
451         /* Check Systick counter flag to decrement the time-out value */
452         if (LL_SYSTICK_IsActiveCounterFlag())
453         {
454           if(Timeout-- == 0)
455           {
456           /* Time-out occurred. Set LED to blinking mode */
457           LED_Blinking(LED_BLINK_ERROR);
458           }
459         }
460       #endif /* USE_TIMEOUT */
461       }
462 
463       /* Note: ADC flag ADRDY is not cleared here to be able to check ADC      */
464       /*       status afterwards.                                            */
```

```
465       /*       This flag should be cleared at ADC Deactivation, before a new    */
466       /*       ADC activation, using function "LL_ADC_ClearFlag_ADRDY()".       */
467     }
468
469     /*## Operation on ADC hierarchical scope: ADC group regular ################*/
470     /* Note: No operation on ADC group regular performed here.                   */
471     /*       ADC group regular conversions to be performed after this function  */
472     /*       using function:                                                     */
473     /*       "LL_ADC_REG_StartConversion();"                                     */
474
475     /*## Operation on ADC hierarchical scope: ADC group injected ################*/
476     /* Note: No operation on ADC group injected performed here.                  */
477     /*       ADC group injected conversions to be performed after this function */
478     /*       using function:                                                     */
479     /*       "LL_ADC_INJ_StartConversion();"                                     */
480
481   }
482
483
484   /* LPTIM1 init function */
485   static void MX_LPTIM1_Init(void)
486   {
487
488     hlptim1.Instance = LPTIM1;
489     hlptim1.Init.Clock.Source = LPTIM_CLOCKSOURCE_APBCLOCK_LPOSC;
490     hlptim1.Init.Clock.Prescaler = LPTIM_PRESCALER_DIV128;
491     hlptim1.Init.Trigger.Source = LPTIM_TRIGSOURCE_SOFTWARE;
492     hlptim1.Init.OutputPolarity = LPTIM_OUTPUTPOLARITY_HIGH;
493     hlptim1.Init.UpdateMode = LPTIM_UPDATE_IMMEDIATE;
494     hlptim1.Init.CounterSource = LPTIM_COUNTERSOURCE_INTERNAL;
495     hlptim1.Init.Input1Source = LPTIM_INPUT1SOURCE_GPIO;
496     hlptim1.Init.Input2Source = LPTIM_INPUT2SOURCE_GPIO;
497     if (HAL_LPTIM_Init(&hlptim1) != HAL_OK)
498     {
499       _Error_Handler(__FILE__, __LINE__);
500     }
501
502   }
503
504   /* LPUART1 init function */
505   static void MX_LPUART1_UART_Init(void)
506   {
507
508     hlpuart1.Instance = LPUART1;
509     hlpuart1.Init.BaudRate = 9600;
510     hlpuart1.Init.WordLength = UART_WORDLENGTH_8B;
511     hlpuart1.Init.StopBits = UART_STOPBITS_1;
512     hlpuart1.Init.Parity = UART_PARITY_NONE;
513     hlpuart1.Init.Mode = UART_MODE_TX_RX;
514     hlpuart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
515     hlpuart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
516     hlpuart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
517     if (HAL_UART_Init(&hlpuart1) != HAL_OK)
518     {
519       _Error_Handler(__FILE__, __LINE__);
520     }
521
522   }
523
524   /**
525     * Enable DMA controller clock
526     */
527   void Configure_DMA(void)
528   {
529     /*## Configuration of DMA ##################################################*/
530     /* Enable the peripheral clock of DMA */
531     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_DMA1);
532
533     /* Configure NVIC to enable DMA interruptions */
534     NVIC_SetPriority(DMA1_Channel3_IRQn, 3);  /* DMA IRQ lower priority than ADC IRQ */
535     NVIC_EnableIRQ(DMA1_Channel3_IRQn);
536       /* Configure the DMA transfer temperature */
537     /*  - DMA transfer in circular mode to match with ADC configuration:        */
538     /*    DMA unlimited requests.                                               */
539     /*  - DMA transfer from ADC without address increment.                      */
540     /*  - DMA transfer to memory with address increment.                        */
541     /*  - DMA transfer from ADC by word to match with ADC configuration:        */
542     /*    ADC resolution 12 bits and and multimode enabled,                     */
```

```
543     /*    ADC master and ADC slave conversion data are concatenated in          */
544     /*    a register of 32 bits.                                                */
545     /*  - DMA transfer to memory by word to match with ADC conversion data      */
546     /*    buffer variable type: word.                                           */
547     LL_DMA_ConfigTransfer(DMA1,
548                           LL_DMA_CHANNEL_3,
549                           LL_DMA_DIRECTION_PERIPH_TO_MEMORY |
550                           LL_DMA_MODE_CIRCULAR              |
551                           LL_DMA_PERIPH_NOINCREMENT         |
552                           LL_DMA_MEMORY_INCREMENT           |
553                           LL_DMA_PDATAALIGN_HALFWORD            |
554                           LL_DMA_MDATAALIGN_HALFWORD            |
555                           LL_DMA_PRIORITY_HIGH               );
556
557     /* Select ADC as DMA transfer request */
558     LL_DMA_SetPeriphRequest(DMA1,
559                             LL_DMA_CHANNEL_3,
560                             LL_DMA_REQUEST_0);
561
562     /* Set DMA transfer addresses of source and destination */
563     /* Note: On this STM32 device, in multimode, ADC conversion data with       */
564     /*       ADC master and ADC slave conversion data concatenated are located  */
565     /*       in a specific multimode data register.                             */
566     LL_DMA_ConfigAddresses(DMA1,
567                            LL_DMA_CHANNEL_3,
568                            LL_ADC_DMA_GetRegAddr(ADC3, LL_ADC_DMA_REG_REGULAR_DATA),
569                            (uint32_t)&dmabuffer[0],
570                            LL_DMA_DIRECTION_PERIPH_TO_MEMORY);
571
572     /* Set DMA transfer size */
573     LL_DMA_SetDataLength(DMA1,
574                          LL_DMA_CHANNEL_3,
575                          250);
576
577     //Activamos el DMA channel 3
578     /* Enable DMA transfer interruption: transfer complete */
579     LL_DMA_EnableIT_TC(DMA1,
580                        LL_DMA_CHANNEL_3);
581
582     /* Enable DMA transfer interruption: half transfer */
583     LL_DMA_EnableIT_HT(DMA1,
584                        LL_DMA_CHANNEL_3);
585
586     /* Enable DMA transfer interruption: transfer error */
587     LL_DMA_EnableIT_TE(DMA1,
588                        LL_DMA_CHANNEL_3);
589     /*## Activation of DMA #####################################################*/
590     /* Enable the DMA transfer */
591     LL_DMA_EnableChannel(DMA1,
592                          LL_DMA_CHANNEL_3);
593
594   }
595
596   /** Configure pins as
597           * Analog
598           * Input
599           * Output
600           * EVENT_OUT
601           * EXTI
602   */
603   static void MX_GPIO_Init(void)
604   {
605
606     GPIO_InitTypeDef GPIO_InitStruct;
607
608     /* GPIO Ports Clock Enable */
609     __HAL_RCC_GPIOC_CLK_ENABLE();
610     __HAL_RCC_GPIOH_CLK_ENABLE();
611     __HAL_RCC_GPIOB_CLK_ENABLE();
612     __HAL_RCC_GPIOG_CLK_ENABLE();
613     HAL_PWREx_EnableVddIO2();
614     __HAL_RCC_GPIOA_CLK_ENABLE();
615
616     /*Configure GPIO pin Output Level */
617     HAL_GPIO_WritePin(GPIOB, LD3_Pin|LD2_Pin, GPIO_PIN_RESET);
618
619     /*Configure GPIO pin Output Level */
620     HAL_GPIO_WritePin(GPIOG, USB_PowerSwitchOn_Pin|SMPS_V1_Pin|SMPS_EN_Pin|SMPS_SW_Pin, GPIO_PIN_RESET);
```

```
621
622     /*Configure GPIO pin : B1_Pin */
623     GPIO_InitStruct.Pin = B1_Pin;
624     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
625     GPIO_InitStruct.Pull = GPIO_NOPULL;
626     HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
627
628     /*Configure GPIO pins : LD3_Pin LD2_Pin */
629     GPIO_InitStruct.Pin = LD3_Pin|LD2_Pin;
630     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
631     GPIO_InitStruct.Pull = GPIO_NOPULL;
632     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
633     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
634
635     /*Configure GPIO pins : USB_OverCurrent_Pin SMPS_PG_Pin */
636     GPIO_InitStruct.Pin = USB_OverCurrent_Pin|SMPS_PG_Pin;
637     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
638     GPIO_InitStruct.Pull = GPIO_NOPULL;
639     HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
640
641     /*Configure GPIO pins : USB_PowerSwitchOn_Pin SMPS_V1_Pin SMPS_EN_Pin SMPS_SW_Pin */
642     GPIO_InitStruct.Pin = USB_PowerSwitchOn_Pin|SMPS_V1_Pin|SMPS_EN_Pin|SMPS_SW_Pin;
643     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
644     GPIO_InitStruct.Pull = GPIO_NOPULL;
645     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
646     HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
647
648   }
649
650   /* USER CODE BEGIN 4 */
651   void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
652     int i;
653     if(RxBuffer[0] == EVENT && RxBuffer[1] == 0x0F && RxBuffer[3] == 0x00 && RxBuffer[5] == 0x0D &&
      RxBuffer[6]== 0x20){
654
655       TxBuffer[0] = ACL_DATA;
656       TxBuffer[1] = RxBuffer[5];
657       TxBuffer[2] = RxBuffer[6];
658       TxBuffer[3] = 0xFB;
659       TxBuffer[4] = 0x00;
660       for(i = 5;i<256; i++){
661         TxBuffer[i] = Data[i-5];
662       }
663
664       HAL_UART_Transmit(&hlpuart1,TxBuffer,256,HAL_MAX_DELAY);
665     }
666   }
667
668   void go_to_sleep(void){
669
670     HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);
671
672   }
673
674   void HAL_LPTIM_AutoReloadMatchCallback(LPTIM_HandleTypeDef *hlptim){
675     HAL_GPIO_TogglePin(LD3_GPIO_Port,LD3_Pin);
676     uint8_t adquisicion = 0;
677     int k;
678     int j;
679     char medidaschar[11];
680
681     Configure_DMA();
682     Configure_ADC3();
683     Activate_ADC3();
684
685     k = 0;
686
687     while(adquisicion == 0){
688       switch(mainstate){
689         case 0:
690           //Si todo esta OK empezamos la conversión del ADC
691           if ((LL_ADC_IsEnabled(ADC3) == 1)                 &&
692           (LL_ADC_IsDisableOngoing(ADC3) == 0)        &&
693           (LL_ADC_REG_IsConversionOngoing(ADC3) == 0)    )
694           {
695             LL_ADC_REG_StartConversion(ADC3);
696             mainstate++;
697           }
```

```
698
699           break;
700
701         case 1:
702           //Esperamos que acabe la adquisición
703         break;
704
705         case 2:
706           //Convertimos los datos de integers a un string de chars
707         for(k = 0; k<250;k++){
708           floattostring(dmabuffer[k],medidaschar,00);
709             for (j=0;j<11;j++)
710               Data[k*j+j] = medidaschar[j];
711         }
712         adquisicion = 1;
713
714         break;
715
716       }
717     }
718
719
720     if (HAL_LPTIM_Counter_Start_IT(&hlptim1, 1000) != HAL_OK)
721     {
722       Error_Handler();
723     }
724     go_to_sleep();
725
726   }
727
728   void AdcDmaTransferComplete_Callback(){
729     HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
730     LL_ADC_REG_StopConversion(ADC3);
731     mainstate++;
732   }
733
734   //Inicio funciones conversion float to string
735   /**
736     * @brief Funcion para invertir un array
737     * @retval None
738     * INPUTS:
739     * *str: Puntero al array que invertir
740     * len: cantidad de posiciones que se invertiran
741     */
742   void reverse(char *str, int len) {
743       int i=0, j=len-1, temp;
744       while (i<j)
745       {
746           temp = str[i];
747           str[i] = str[j];
748           str[j] = temp;
749           i++; j--;
750       }
751   }
752
753
754   /**
755     * @brief Convierte un entero a string
756     * @retval Longitud del array
757     * INPUTS:
758     * x: Entero que se convertira a string
759     * str: array donde guardar el resultado
760     * d: 0s a la derecha de mas que se quieran poner
761     */
762   int intToStr(int x, char str[], int d) {
763       int i = 0;
764       while (x)
765       {
766           str[i++] = (x%10) + '0';
767           x = x/10;
768       }
769
770       // If number of digits required is more, then
771       // add 0s at the beginning
772       while (i < d)
773           str[i++] = '0';
774
775       reverse(str, i);
```

```
776       str[i] = '\0';
777       return i;
778   }
779
780   /**
781     * @brief Convierte un entero a string modificado
782     * @retval Longitud del array
783     * INPUTS:
784     * x: Entero que se convertira a string
785     * str: array donde guardar el resultado
786     * str2: array con el tipo de float para añadirlo al string
787     * pos: indica si el entero es positivo para poner un '+' o '-'
788     */
789   int intToStr2(int x, char str[],uint16_t currentmed,int pos) {
790       int i = 0;
791       int desena = currentmed/10;
792       int unitat = currentmed - desena*10;
793
794       if(x==0) str[i++] = '0';
795       while (x)
796       {
797           str[i++] = (x%10) + '0';
798           x = x/10;
799       }
800
801       if (pos)
802         str[i++] = '+';
803       else
804         str[i++] = '-';
805
806       str[i++] = '/';
807       str[i++] = '/';
808
809       reverse(str, i);
810       str[i] = '\0';
811       return i;
812   }
813
814   /**
815     * @brief Convierte un float a string
816     * @retval None
817     * INPUTS:
818     * medida: float que se convertira a string
819     * *medidachar: puntero al array donde guardar el string
820     * *tipus: puntero al array donde se encuentra el tipo de float
821     */
822   void floattostring(float medida, char *medidachar, uint16_t currentmed){
823
824     int l = 0;
825     int ipart = (int)medida;
826     int pos = 0;
827     double fpart;
828       if (medida>0){
829         pos = 1;
830         fpart = medida - (float)ipart;
831       }
832       else{
833         ipart=-ipart;
834         fpart = - medida - (float) ipart;
835       }
836
837     int i = intToStr2(ipart,medidachar,currentmed,pos);
838
839     medidachar[i] = '.';
840
841     int afterpoint = 10 - i;
842     if (afterpoint !=0 )
843     {
844       for (l = 1; l<=afterpoint; l++)
845         fpart = fpart*10;
846       intToStr((int)fpart, medidachar + i + 1, afterpoint);
847     }
848   }
849
850
851   /* USER CODE END 4 */
852
853   /**
```

```
854     * @brief  This function is executed in case of error occurrence.
855     * @param  file: The file name as string.
856     * @param  line: The line in file as a number.
857     * @retval None
858     */
859   void _Error_Handler(char *file, int line)
860   {
861     /* USER CODE BEGIN Error_Handler_Debug */
862     /* User can add his own implementation to report the HAL error return state */
863     while(1)
864     {
865     }
866     /* USER CODE END Error_Handler_Debug */
867   }
868
869   #ifdef  USE_FULL_ASSERT
870   /**
871     * @brief  Reports the name of the source file and the source line number
872     *         where the assert_param error has occurred.
873     * @param  file: pointer to the source file name
874     * @param  line: assert_param error line source number
875     * @retval None
876     */
877   void assert_failed(uint8_t* file, uint32_t line)
878   {
879     /* USER CODE BEGIN 6 */
880     /* User can add his own implementation to report the file name and line number,
881        tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
882     /* USER CODE END 6 */
883   }
884   #endif /* USE_FULL_ASSERT */
885
886   /**
887     * @}
888     */
889
890   /**
891     * @}
892     */
893
894   /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
895
```

A.2. main.c del microprocesador del dron

```
1
2   /**
3     ******************************************************************************
4     * @file           : main.c
5     * @brief          : Main program body
6     ******************************************************************************
7     * This notice applies to any and all portions of this file
8     * that are not between comment pairs USER CODE BEGIN and
9     * USER CODE END. Other portions of this file, whether
10    * inserted by the user or by software development tools
11    * are owned by their respective copyright owners.
12    *
13    * Copyright (c) 2019 STMicroelectronics International N.V.
14    * All rights reserved.
15    *
16    * Redistribution and use in source and binary forms, with or without
17    * modification, are permitted, provided that the following conditions are met:
18    *
19    * 1. Redistribution of source code must retain the above copyright notice,
20    *    this list of conditions and the following disclaimer.
21    * 2. Redistributions in binary form must reproduce the above copyright notice,
22    *    this list of conditions and the following disclaimer in the documentation
23    *    and/or other materials provided with the distribution.
24    * 3. Neither the name of STMicroelectronics nor the names of other
25    *    contributors to this software may be used to endorse or promote products
26    *    derived from this software without specific written permission.
27    * 4. This software, including modifications and/or derivative works of this
28    *    software, must execute solely and exclusively on microcontroller or
29    *    microprocessor devices manufactured by or for STMicroelectronics.
30    * 5. Redistribution and use of this software other than as permitted under
31    *    this license is void and will automatically terminate your rights under
32    *    this license.
33    *
34    * THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS"
35    * AND ANY EXPRESS, IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT
36    * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
37    * PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY
38    * RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO EVENT
39    * SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
40    * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
41    * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
42    * OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
43    * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
44    * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
45    * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
46    *
47    ******************************************************************************
48    */
49  /* Includes ------------------------------------------------------------------*/
50  #include "main.h"
51  #include "stm32l4xx_hal.h"
52  #include "usb_host.h"
53
54  /* USER CODE BEGIN Includes */
55  //HCI Packet types
56  #define COMMAND   0x01
57  #define ACL_DATA  0x02
58  #define SCO_DATA  0x03
59  #define EVENT     0x04
60  /* USER CODE END Includes */
61
62  /* Private variables ---------------------------------------------------------*/
63  UART_HandleTypeDef hlpuart1;
64
65  /* USER CODE BEGIN PV */
66  /* Private variables ---------------------------------------------------------*/
67  //HCI Packet types
68  __IO ITStatus UartReady = RESET;
69
70  uint8_t RxBuffer[256]= " ";
71  uint8_t TxBuffer[256]= " ";
72  uint8_t CommandTx = 0x0001;
73  /* USER CODE END PV */
74
75  /* Private function prototypes -----------------------------------------------*/
76  void SystemClock_Config(void);
77  static void MX_GPIO_Init(void);
78  static void MX_LPUART1_UART_Init(void);
```

```
  79  void MX_USB_HOST_Process(void);
  80  void go_to_sleep(void);
  81
  82  /* USER CODE BEGIN PFP */
  83  /* Private function prototypes -----------------------------------------------*/
  84
  85  /* USER CODE END PFP */
  86
  87  /* USER CODE BEGIN 0 */
  88
  89  /* USER CODE END 0 */
  90
  91  /**
  92    * @brief  The application entry point.
  93    *
  94    * @retval None
  95    */
  96  int main(void)
  97  {
  98    /* USER CODE BEGIN 1 */
  99
 100    /* USER CODE END 1 */
 101
 102    /* MCU Configuration--------------------------------------------------------*/
 103
 104    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 105    HAL_Init();
 106
 107    /* USER CODE BEGIN Init */
 108
 109    /* USER CODE END Init */
 110
 111    /* Configure the system clock */
 112    SystemClock_Config();
 113
 114    /* USER CODE BEGIN SysInit */
 115
 116    /* USER CODE END SysInit */
 117
 118    /* Initialize all configured peripherals */
 119    MX_GPIO_Init();
 120    MX_LPUART1_UART_Init();
 121    MX_USB_HOST_Init();
 122    /* USER CODE BEGIN 2 */
 123    //Preparamos el Tx para el comando reset
 124    TxBuffer[0] = 0x01; //PACKET TYPE BYTE
 125    TxBuffer[1] = 0x03;     //OPCODE BYTE 1
 126    TxBuffer[2] = 0x0C;     //OPCODE BYTE 2
 127
 128    /* USER CODE END 2 */
 129
 130    /* Infinite loop */
 131    /* USER CODE BEGIN WHILE */
 132    while (1)
 133    {
 134      //Enviamos el comando reset i comprovamos si recibimos el evento correcto
 135        HAL_UART_Transmit(&hlpuart1,TxBuffer,4,HAL_MAX_DELAY);
 136        HAL_UART_Receive(&hlpuart1,RxBuffer,7,HAL_MAX_DELAY);
 137
 138        if(RxBuffer[0] == EVENT && RxBuffer[1] == 0x0E && RxBuffer[6]== 0x00) {
 139            //Si recibimos el evento correcto habilitamos los eventos LE
 140            // SET EVENT MASK PARA HABILITAR LOS EVENTOS LE
 141          //              OCF               OGF               LENGTH
      B1                  B2                  B3                  B4                  B5
 142          TxBuffer[1] = 0x01; TxBuffer[2] = 0x0C; TxBuffer[3] = 0x08; TxBuffer[4] = 0x00; TxBuffer[5]
      = 0x00; TxBuffer[6] = 0x00; TxBuffer[7] = 0x00; TxBuffer[8] = 0x00;
 143          //              B6                  B7                  B8
 144          TxBuffer[9] = 0x00; TxBuffer[10] = 0x00; TxBuffer[11] = 0x20;
 145
 146          HAL_UART_Transmit(&hlpuart1,TxBuffer,12,HAL_MAX_DELAY);
 147          HAL_UART_Receive(&hlpuart1,RxBuffer,7,HAL_MAX_DELAY);
 148
 149          if(RxBuffer[0] == EVENT && RxBuffer[1] == 0x0E && RxBuffer[4] == 0x01 && RxBuffer[5] == 0x0C
      && RxBuffer[6]== 0x00){
 150            //Si se ha habilitado correctamente ya podemos entrar en el modo standby hasta que el dron
      indique que se ha llegado al nodo
 151            go_to_sleep();
 152          }
```

```
153         }
154     /* USER CODE END WHILE */
155       MX_USB_HOST_Process();
156
157     /* USER CODE BEGIN 3 */
158
159     }
160     /* USER CODE END 3 */
161
162   }
163
164   /**
165     * @brief System Clock Configuration
166     * @retval None
167     */
168   void SystemClock_Config(void)
169   {
170
171     RCC_OscInitTypeDef RCC_OscInitStruct;
172     RCC_ClkInitTypeDef RCC_ClkInitStruct;
173     RCC_PeriphCLKInitTypeDef PeriphClkInit;
174
175       /**Configure LSE Drive Capability
176       */
177     HAL_PWR_EnableBkUpAccess();
178
179     __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
180
181       /**Initializes the CPU, AHB and APB busses clocks
182       */
183     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
184     RCC_OscInitStruct.LSEState = RCC_LSE_ON;
185     RCC_OscInitStruct.MSIState = RCC_MSI_ON;
186     RCC_OscInitStruct.MSICalibrationValue = 0;
187     RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_9;
188     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
189     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
190     RCC_OscInitStruct.PLL.PLLM = 5;
191     RCC_OscInitStruct.PLL.PLLN = 71;
192     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
193     RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
194     RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV6;
195     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
196     {
197       _Error_Handler(__FILE__, __LINE__);
198     }
199
200       /**Initializes the CPU, AHB and APB busses clocks
201       */
202     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
203                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
204     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
205     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV4;
206     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
207     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
208
209     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
210     {
211       _Error_Handler(__FILE__, __LINE__);
212     }
213
214     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_LPUART1|RCC_PERIPHCLK_USB;
215     PeriphClkInit.Lpuart1ClockSelection = RCC_LPUART1CLKSOURCE_PCLK1;
216     PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLLSAI1;
217     PeriphClkInit.PLLSAI1.PLLSAI1Source = RCC_PLLSOURCE_MSI;
218     PeriphClkInit.PLLSAI1.PLLSAI1M = 5;
219     PeriphClkInit.PLLSAI1.PLLSAI1N = 20;
220     PeriphClkInit.PLLSAI1.PLLSAI1P = RCC_PLLP_DIV2;
221     PeriphClkInit.PLLSAI1.PLLSAI1Q = RCC_PLLQ_DIV2;
222     PeriphClkInit.PLLSAI1.PLLSAI1R = RCC_PLLR_DIV2;
223     PeriphClkInit.PLLSAI1.PLLSAI1ClockOut = RCC_PLLSAI1_48M2CLK;
224     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
225     {
226       _Error_Handler(__FILE__, __LINE__);
227     }
228
229       /**Configure the main internal regulator output voltage
230       */
```

```
231     if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
232     {
233       _Error_Handler(__FILE__, __LINE__);
234     }
235 
236       /**Configure the Systick interrupt time
237       */
238     HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
239 
240       /**Configure the Systick
241       */
242     HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
243 
244       /**Enable MSI Auto calibration
245       */
246     HAL_RCCEx_EnableMSIPLLMode();
247 
248     /* SysTick_IRQn interrupt configuration */
249     HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
250   }
251 
252   /* LPUART1 init function */
253   static void MX_LPUART1_UART_Init(void)
254   {
255 
256     hlpuart1.Instance = LPUART1;
257     hlpuart1.Init.BaudRate = 115200;
258     hlpuart1.Init.WordLength = UART_WORDLENGTH_8B;
259     hlpuart1.Init.StopBits = UART_STOPBITS_1;
260     hlpuart1.Init.Parity = UART_PARITY_NONE;
261     hlpuart1.Init.Mode = UART_MODE_TX_RX;
262     hlpuart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
263     hlpuart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
264     hlpuart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
265     if (HAL_UART_Init(&hlpuart1) != HAL_OK)
266     {
267       _Error_Handler(__FILE__, __LINE__);
268     }
269 
270   }
271 
272   /** Configure pins as
273           * Analog
274           * Input
275           * Output
276           * EVENT_OUT
277           * EXTI
278   */
279   static void MX_GPIO_Init(void)
280   {
281 
282     GPIO_InitTypeDef GPIO_InitStruct;
283 
284     /* GPIO Ports Clock Enable */
285     __HAL_RCC_GPIOC_CLK_ENABLE();
286     __HAL_RCC_GPIOH_CLK_ENABLE();
287     __HAL_RCC_GPIOB_CLK_ENABLE();
288     __HAL_RCC_GPIOG_CLK_ENABLE();
289     HAL_PWREx_EnableVddIO2();
290     __HAL_RCC_GPIOA_CLK_ENABLE();
291 
292     /*Configure GPIO pin Output Level */
293     HAL_GPIO_WritePin(GPIOB, LD3_Pin|LD2_Pin, GPIO_PIN_RESET);
294 
295     /*Configure GPIO pin Output Level */
296     HAL_GPIO_WritePin(GPIOG, USB_PowerSwitchOn_Pin|SMPS_V1_Pin|SMPS_EN_Pin|SMPS_SW_Pin, GPIO_PIN_RESET);
297 
298     /*Configure GPIO pin : B1_Pin */
299     GPIO_InitStruct.Pin = B1_Pin;
300     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
301     GPIO_InitStruct.Pull = GPIO_NOPULL;
302     HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
303 
304     /*Configure GPIO pins : LD3_Pin LD2_Pin */
305     GPIO_InitStruct.Pin = LD3_Pin|LD2_Pin;
306     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
307     GPIO_InitStruct.Pull = GPIO_NOPULL;
308     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
309     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
310
311     /*Configure GPIO pins : USB_OverCurrent_Pin SMPS_PG_Pin */
312     GPIO_InitStruct.Pin = USB_OverCurrent_Pin|SMPS_PG_Pin;
313     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
314     GPIO_InitStruct.Pull = GPIO_NOPULL;
315     HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
316
317     /*Configure GPIO pins : USB_PowerSwitchOn_Pin SMPS_V1_Pin SMPS_EN_Pin SMPS_SW_Pin */
318     GPIO_InitStruct.Pin = USB_PowerSwitchOn_Pin|SMPS_V1_Pin|SMPS_EN_Pin|SMPS_SW_Pin;
319     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
320     GPIO_InitStruct.Pull = GPIO_NOPULL;
321     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
322     HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
323
324     /*Configure GPIO pin : DRON_Pin */
325     GPIO_InitStruct.Pin = DRON_Pin;
326     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
327     GPIO_InitStruct.Pull = GPIO_NOPULL;
328     HAL_GPIO_Init(DRON_GPIO_Port, &GPIO_InitStruct);
329
330     /* EXTI interrupt init*/
331     HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
332     HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
333
334     HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
335     HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
336
337   }
338
339   /* USER CODE BEGIN 4 */
340   void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
341     //Este interrupt indica que ya hemos recibido los datos
342     go_to_sleep();
343   }
344
345   void go_to_sleep(void){
346
347     HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);
348
349   }
350
351   void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
352     uint8_t notconnected = 1;
353     uint16_t counter = 0;
354     if(GPIO_Pin == DRON_Pin){
355       //El dron indica que estamos cerca del nodo asi que creamos la conexion bluetooth
356        // Tx per crear una connexió
357           TxBuffer[1]  = 0x0D;  TxBuffer[2]  = 0x20;  TxBuffer[3]  = 0x19;  TxBuffer[4]  = 0x10;
   TxBuffer[5]  = 0x00;  TxBuffer[6]  = 0x10;  TxBuffer[7]  = 0x00;
358           TxBuffer[8]  = 0x00;  TxBuffer[9]  = 0x00;  TxBuffer[10] = 0x80;  TxBuffer[11] = 0x95;
   TxBuffer[12] = 0x19;  TxBuffer[13] = 0x29;  TxBuffer[14] = 0x49;
359           TxBuffer[15] = 0x80;  TxBuffer[16] = 0x00;  TxBuffer[17] = 0x06;  TxBuffer[18] = 0x00;
   TxBuffer[19] = 0x80;  TxBuffer[20] = 0x00;  TxBuffer[21] = 0x00;
360           TxBuffer[22] = 0x00;  TxBuffer[23] = 0x32;  TxBuffer[24] = 0x00;  TxBuffer[25] = 0x00;
   TxBuffer[26] = 0x00;  TxBuffer[27] = 0x14;  TxBuffer[28] = 0x00;
361
362           while(notconnected){
363             HAL_UART_Transmit(&hlpuart1,TxBuffer,29,HAL_MAX_DELAY);
364             HAL_UART_Receive(&hlpuart1,RxBuffer,7,HAL_MAX_DELAY);
365
366             if(RxBuffer[0] == EVENT && RxBuffer[1] == 0x0F && RxBuffer[3] == 0x00 && RxBuffer[5] ==
   0x0D && RxBuffer[6]== 0x20){
367               //Si la conexión se ha creado correctamente esperamos a recibir los datos
368               HAL_UART_Receive_IT(&hlpuart1,RxBuffer,255); notconnected = 0;
369             } else {
370               if(counter>500){
371                 notconnected = 0; go_to_sleep();
372               }else counter++;
373             }
374           }
375         } else go_to_sleep();
376   }
377   /* USER CODE END 4 */
378
379   /**
380     * @brief  This function is executed in case of error occurrence.
381     * @param  file: The file name as string.
```

```
382     * @param  line: The line in file as a number.
383     * @retval None
384     */
385   void _Error_Handler(char *file, int line)
386   {
387     /* USER CODE BEGIN Error_Handler_Debug */
388     /* User can add his own implementation to report the HAL error return state */
389     while(1)
390     {
391     }
392     /* USER CODE END Error_Handler_Debug */
393   }
394
395   #ifdef  USE_FULL_ASSERT
396   /**
397     * @brief  Reports the name of the source file and the source line number
398     *         where the assert_param error has occurred.
399     * @param  file: pointer to the source file name
400     * @param  line: assert_param error line source number
401     * @retval None
402     */
403   void assert_failed(uint8_t* file, uint32_t line)
404   {
405     /* USER CODE BEGIN 6 */
406     /* User can add his own implementation to report the file name and line number,
407        tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
408     /* USER CODE END 6 */
409   }
410   #endif /* USE_FULL_ASSERT */
411
412   /**
413     * @}
414     */
415
416   /**
417     * @}
418     */
419
420   /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
421
```

A.3. main.c del transceptor

```
/*******************************************************************************
* Project Name      : PSoC_4_BLE_DTM
* File Name         : main.c
* Version           : 1.0
* Device Used       : CY8C4247LQI-BL483
* Software Used     : PSoC Creator 4.2
* Compiler          : ARM GCC 5.4.1
* Related Hardware  : CY8CKIT-042-BLE Bluetooth Low Energy Pioneer Kit
********************************************************************************
* Copyright (2018), Cypress Semiconductor Corporation. All rights reserved.
********************************************************************************
* This software, including source code, documentation and related materials
* ("Software"), is owned by Cypress Semiconductor Corporation or one of its
* subsidiaries ("Cypress") and is protected by and subject to worldwide patent
* protection (United States and foreign), United States copyright laws and
* international treaty provisions. Therefore, you may use this Software only
* as provided in the license agreement accompanying the software package from
* which you obtained this Software ("EULA").
*
* If no EULA applies, Cypress hereby grants you a personal, nonexclusive,
* non-transferable license to copy, modify, and compile the Software source
* code solely for use in connection with Cypress's integrated circuit products.
* Any reproduction, modification, translation, compilation, or representation
* of this Software except as specified above is prohibited without the express
* written permission of Cypress.
*
* Disclaimer: THIS SOFTWARE IS PROVIDED AS-IS, WITH NO WARRANTY OF ANY KIND,
* EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, NONINFRINGEMENT, IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress
* reserves the right to make changes to the Software without notice. Cypress
* does not assume any liability arising out of the application or use of the
* Software or any product or circuit described in the Software. Cypress does
* not authorize its products for use in any products where a malfunction or
* failure of the Cypress product may reasonably be expected to result in
* significant property damage, injury or death ("High Risk Product"). By
* including Cypress's product in a High Risk Product, the manufacturer of such
* system or application assumes all risk of such use and in doing so agrees to
* indemnify Cypress against all liability.
********************************************************************************↵
/

/↵
********************************************************************************↵
**
*                              THEORY OF OPERATION
********************************************************************************↵
***
* This project operates in BLE specification defined HCI mode and executes ↵
the Direct
* Test Mode (DTM) cases. There is no application level activity except ↵
processing
* BLE events. This project requires the external CBT tester to operate.
```

```
* Refer CY8CKIT-042-BLE Pioneer Kit guide for more details.
********************************************************************************
***
* Hardware connection required for testing -
* UART RX   - P0[0] (Connect this pin from J3 header on BLE Pioneer Kit to
UART TX
                    line of external RS232 converter)
* UART TX   - P0[1] (Connect this pin from J3 header on BLE Pioneer Kit to
UART RX
                    line of external RS232 converter)
* RTS       - P0[2] (Connect this pin from J3 header on BLE Pioneer Kit to CTS
                    line of external RS232 converter; can be left unconnected)
* CTS       - P0[3] (Connect this pin from J3 header on BLE Pioneer Kit to RTS
                    line of external RS232 converter; can be left unconnected)
********************************************************************************
***/
#include <project.h>

#define ENABLE_LOW_POWER_MODE

/* BLE Callback Event Handler Function */
void ApplicationEventHandler(uint32 event, void * eventparam);
/* Handle Low Power Mode Function */
void HandleLowPowerMode(void);

/
*******************************************************************************
* Function Name: main
********************************************************************************
*
* Summary:
*        System entrance point. This calls the BLE start and processes BLE
Events
*
* Parameters:
*  void
*
* Return:
*  int
*

*******************************************************************************
/
int main()
{
    /* Enable global interrupts. */
    CyGlobalIntEnable;

    /* Start BLE Component and register the event callback function */
    CyBle_Start(ApplicationEventHandler);

    for(;;)
```

```
    {
        /* Process BLE events continuously */
        CyBle_ProcessEvents();

        /* Put CPU and BLESS to low power mode */
        HandleLowPowerMode();
    }
}

/
*******************************************************************************
* Function Name: ApplicationEventHandler
*******************************************************************************
*
* Summary:
*        Call back event function to handle various events from BLE stack
*
* Parameters:
*  uint32 event:        event returned
*  void * eventparam:   link to value of the events returned
*
* Return:
*  void
*
*******************************************************************************
/
void ApplicationEventHandler(uint32 event, void * eventparam)
{
    switch(event)
    {
        /* No application event is generated in HCI Mode. All commands
        * are processed by BLE stack */

        default:
            /* To prevent compiler warning */
            eventparam = eventparam;
        break;
    }
}

/
*******************************************************************************
* Function Name: HandleLowPowerMode
*******************************************************************************
*
* Summary:
*        This function puts the BLESS in deep sleep mode and CPU to sleep mode
.
*
* Parameters:
*  void
*
```

```
* Return:
*  void
*
*******************************************************************************
/
void HandleLowPowerMode(void)
{
    #ifdef ENABLE_LOW_POWER_MODE
        /* Local variable to store the status of BLESS Hardware block */
        CYBLE_LP_MODE_T sleepMode;
        CYBLE_BLESS_STATE_T blessState;

        /* Put BLESS into Deep Sleep and check the return status */
        sleepMode = CyBle_EnterLPM(CYBLE_BLESS_DEEPSLEEP);

        /* Disable global interrupt to prevent changes from any other
interrupt ISR */
        CyGlobalIntDisable;

        /* Check the Status of BLESS */
        blessState = CyBle_GetBleSsState();

        if(sleepMode == CYBLE_BLESS_DEEPSLEEP)
        {
            if(blessState == CYBLE_BLESS_STATE_ECO_ON || blessState ==
CYBLE_BLESS_STATE_DEEPSLEEP)
            {
                /* No action for DTM Project */
            }
        }
        else
        {
            if(blessState != CYBLE_BLESS_STATE_EVENT_CLOSE)
            {
                /* If the BLESS hardware block did not go to Deep Sleep and
BLE Event has not
                * closed yet, then place CPU to Sleep */
                CySysPmSleep();
            }
        }

        /* Re-enable global interrupt mask after wakeup */
        CyGlobalIntEnable;
    #endif
}

/* [] END OF FILE */
```