Annexe S.15

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 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 <stdbool.h>
//#include "stm32f0xx.h"
#include "mesures.h"
#include "affichage.h"
#include "gestionBatterie.h"
#include "communication.h"
#include "sommeil.h"
#include "EPD_2in13_V4.h" // TEMP
/* 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;

I2C_HandleTypeDef hi2c2;

RTC_HandleTypeDef hrtc;

SPI_HandleTypeDef hspi1;

TIM_HandleTypeDef htim6;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */

EtatSTM etatSTM = INITIALISATION;

// Variables mesures
Mesures mesures;

// https://www.arduino.cc/reference/en/language/variables/data-types/array/
DefinitionValeur valeursServeur[] = {
  { "seuilTemperatureMin", 2 },
  { "seuilTemperatureMax", 8 },
  { "ecartTemperature", 1 },
  { "seuilHumiditeMin", 30 },
  { "seuilHumiditeMax", 60 },
  { "ecartHumidite", 5},
};
```

```
 78
 79  // Variables affichage
 80  bool rafraichirEPaper = false;
 81  bool receptionTramesEspTermine = false;
 82
 83  // Variables batterie
 84  InfoBatterie infoBatterie;
 85
 86  /* USER CODE END PV */
 87
 88  /* Private function prototypes -----------------------------------------------*/
 89  void SystemClock_Config(void);
 90  static void MX_GPIO_Init(void);
 91  static void MX_ADC_Init(void);
 92  static void MX_I2C2_Init(void);
 93  static void MX_SPI1_Init(void);
 94  static void MX_USART1_UART_Init(void);
 95  static void MX_TIM6_Init(void);
 96  static void MX_RTC_Init(void);
 97  /* USER CODE BEGIN PFP */
 98
 99  /* USER CODE END PFP */
100
101  /* Private user code ---------------------------------------------------------*/
102  /* USER CODE BEGIN 0 */
103
104
105
106  /* USER CODE END 0 */
107
108  /**
109    * @brief  The application entry point.
110    * @retval int
111    */
112  int main(void)
113  {
114
115    /* USER CODE BEGIN 1 */
116
117    /* USER CODE END 1 */
118
119    /* MCU Configuration--------------------------------------------------------*/
120
121    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
122    HAL_Init();
123
124    /* USER CODE BEGIN Init */
125
126    /* USER CODE END Init */
127
128    /* Configure the system clock */
129    SystemClock_Config();
130
131    /* USER CODE BEGIN SysInit */
132
133    /* USER CODE END SysInit */
134
135    /* Initialize all configured peripherals */
136    MX_GPIO_Init();
137    MX_ADC_Init();
138    MX_I2C2_Init();
139    MX_SPI1_Init();
140    MX_USART1_UART_Init();
141    MX_TIM6_Init();
142    MX_RTC_Init();
143    /* USER CODE BEGIN 2 */
144
145    /* USER CODE END 2 */
146
147    /* Infinite loop */
148    /* USER CODE BEGIN WHILE */
149    while (1)
150    {
151      /* USER CODE END WHILE */
152
153      /* USER CODE BEGIN 3 */
154
```

```
155       switch(etatSTM)
156       {
157         case INITIALISATION:
158           Initialisation(); // Initialisation au lancement du STM
159           etatSTM = RECEPTION; // Pret à recevoir les données de la part de l'ESP
160           break;
161         case RECEPTION:
162 
163           // Note: la fréquence d'exécution de l'état RECEPTION est lié au timeout de la réception UART (define UART_TIMEOUT)
164           ReceptionnerTrameUART(&huart1, valeursServeur);
165 
166           // On vérifie que receptionTramesEspTermine pour être sûr qu'on a bien été réveillé par l'ESP
167           // On vérifie que HAL_GPIO_ReadPin(REVEIL_GPIO_Port, REVEIL_Pin) est GPIO_PIN_RESET pour être sûr que l'ESP à fini d'envoyer ses trames UART
168           if(receptionTramesEspTermine == true && HAL_GPIO_ReadPin(REVEIL_GPIO_Port, REVEIL_Pin) == GPIO_PIN_RESET)
169           {
170             receptionTramesEspTermine = false;
171             etatSTM = VERIFIER_BATTERIE; // Passage à l'étape suivante
172           }
173           break;
174         case VERIFIER_BATTERIE:
175         {
176           bool besoinMesurerEtatBatterie = GestionCheckBatterie(&hadc, &htim6, &infoBatterie);
177           if(besoinMesurerEtatBatterie == true)
178           {
179             etatSTM = MESURER_BATTERIE; // Une mesure de l'état de la batterie est nécessaire
180           }
181           else
182           {
183             etatSTM = MESURER_SONDE;
184           }
185           break;
186         }
187         case MESURER_BATTERIE:
188         {
189           // Attente que le compteur décrémenté par l'interruption timer atteigne 0 (délai entre activation transistor et mesure)
190           if(infoBatterie.compteurCheckBatterie <= 0)
191           {
192             Etat nouvelEtatBatterie = MesurerEtatBatterie(&hadc, &htim6, &infoBatterie); // Procéder à la mesure de l'état de la batterie
193             if(nouvelEtatBatterie != infoBatterie.etatBatterie)
194             {
195               infoBatterie.etatBatterie = nouvelEtatBatterie;
196               EnvoyerTrameUART(&huart1, ETAT_BATTERIE, infoBatterie.etatBatterie); // Indiquer à l'ESP que l'état de la batterie à changé
197             }
198             etatSTM = MESURER_SONDE;
199           }
200           break;
201         }
202         case MESURER_SONDE:
203         {
204           Etat nouvelEtatSeuils = EffectuerMesuresSonde(&hi2c2, &mesures, valeursServeur); // Procéder aux mesures de la température et de l'humidité
205           if(nouvelEtatSeuils != mesures.etatSeuils)
206           {
207             mesures.etatSeuils = nouvelEtatSeuils;
208             EnvoyerTrameUART(&huart1, ETAT_SEUILS, mesures.etatSeuils); // Indiquer à l'ESP que l'état de l'alarme à changé
209           }
210 
211           // Indiquer à l'ESP qu'on a fini d'envoyer les trames UART pour qu'il puisse se rendormir (si ne l'est pas déjà)
212           HAL_GPIO_WritePin(ALARME_GPIO_Port, ALARME_Pin, GPIO_PIN_RESET);
213 
214           // Si un rafraichissement de l'écran est nécessaire
215           if(rafraichirEPaper == true)
216           {
217             etatSTM = RAFRAICHIR_EPAPER;
218           }
219           else
220           {
221             etatSTM = SOMMEIL;
222           }
```

```
223
224           break;
225         }
226         case RAFRAICHIR_EPAPER:
227           rafraichirEPaper = false;
228           AffichageEPaper(&mesures, valeursServeur); // Procéder au rafraichissement de l'écran
229           etatSTM = SOMMEIL;
230           break;
231         case SOMMEIL:
232           EntrerModeSommeil();
233           etatSTM = REVEIL;
234           break;
235         case REVEIL:
236           etatSTM = RECEPTION;
237           break;
238       }
239     }
240     /* USER CODE END 3 */
241   }
242
243   /**
244     * @brief System Clock Configuration
245     * @retval None
246     */
247   void SystemClock_Config(void)
248   {
249     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
250     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
251     RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
252
253     /** Initializes the RCC Oscillators according to the specified parameters
254     * in the RCC_OscInitTypeDef structure.
255     */
256     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI14|RCC_OSCILLATORTYPE_LSI
257                                       |RCC_OSCILLATORTYPE_HSE;
258     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
259     RCC_OscInitStruct.HSI14State = RCC_HSI14_ON;
260     RCC_OscInitStruct.HSI14CalibrationValue = 16;
261     RCC_OscInitStruct.LSIState = RCC_LSI_ON;
262     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
263     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
264     {
265       Error_Handler();
266     }
267
268     /** Initializes the CPU, AHB and APB buses clocks
269     */
270     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
271                                   |RCC_CLOCKTYPE_PCLK1;
272     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
273     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
274     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
275
276     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
277     {
278       Error_Handler();
279     }
280     PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_RTC;
281     PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK1;
282     PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
283     if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
284     {
285       Error_Handler();
286     }
287
288     /** Enables the Clock Security System
289     */
290     HAL_RCC_EnableCSS();
291   }
292
293   /**
294     * @brief ADC Initialization Function
295     * @param None
296     * @retval None
297     */
298   static void MX_ADC_Init(void)
299   {
```

```
300
301     /* USER CODE BEGIN ADC_Init 0 */
302
303     /* USER CODE END ADC_Init 0 */
304
305     ADC_ChannelConfTypeDef sConfig = {0};
306
307     /* USER CODE BEGIN ADC_Init 1 */
308
309     /* USER CODE END ADC_Init 1 */
310
311     /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of
    conversion)
312     */
313     hadc.Instance = ADC1;
314     hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
315     hadc.Init.Resolution = ADC_RESOLUTION_12B;
316     hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
317     hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
318     hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
319     hadc.Init.LowPowerAutoWait = DISABLE;
320     hadc.Init.LowPowerAutoPowerOff = DISABLE;
321     hadc.Init.ContinuousConvMode = DISABLE;
322     hadc.Init.DiscontinuousConvMode = DISABLE;
323     hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
324     hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
325     hadc.Init.DMAContinuousRequests = DISABLE;
326     hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
327     if (HAL_ADC_Init(&hadc) != HAL_OK)
328     {
329       Error_Handler();
330     }
331
332     /** Configure for the selected ADC regular channel to be converted.
333     */
334     sConfig.Channel = ADC_CHANNEL_6;
335     sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
336     sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
337     if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
338     {
339       Error_Handler();
340     }
341     /* USER CODE BEGIN ADC_Init 2 */
342
343     /* USER CODE END ADC_Init 2 */
344
345   }
346
347   /**
348     * @brief I2C2 Initialization Function
349     * @param None
350     * @retval None
351     */
352   static void MX_I2C2_Init(void)
353   {
354
355     /* USER CODE BEGIN I2C2_Init 0 */
356
357     /* USER CODE END I2C2_Init 0 */
358
359     /* USER CODE BEGIN I2C2_Init 1 */
360
361     /* USER CODE END I2C2_Init 1 */
362     hi2c2.Instance = I2C2;
363     hi2c2.Init.Timing = 0x2000090E;
364     hi2c2.Init.OwnAddress1 = 0;
365     hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
366     hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
367     hi2c2.Init.OwnAddress2 = 0;
368     hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
369     hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
370     hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
371     if (HAL_I2C_Init(&hi2c2) != HAL_OK)
372     {
373       Error_Handler();
374     }
375
```

```
376     /** Configure Analogue filter
377     */
378     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
379     {
380       Error_Handler();
381     }
382
383     /** Configure Digital filter
384     */
385     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK)
386     {
387       Error_Handler();
388     }
389     /* USER CODE BEGIN I2C2_Init 2 */
390
391     /* USER CODE END I2C2_Init 2 */
392
393   }
394
395   /**
396     * @brief RTC Initialization Function
397     * @param None
398     * @retval None
399     */
400   static void MX_RTC_Init(void)
401   {
402
403     /* USER CODE BEGIN RTC_Init 0 */
404
405     /* USER CODE END RTC_Init 0 */
406
407     /* USER CODE BEGIN RTC_Init 1 */
408
409     /* USER CODE END RTC_Init 1 */
410
411     /** Initialize RTC Only
412     */
413     hrtc.Instance = RTC;
414     hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
415     hrtc.Init.AsynchPrediv = 127;
416     hrtc.Init.SynchPrediv = 255;
417     hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
418     hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
419     hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
420     if (HAL_RTC_Init(&hrtc) != HAL_OK)
421     {
422       Error_Handler();
423     }
424     /* USER CODE BEGIN RTC_Init 2 */
425
426     /* USER CODE END RTC_Init 2 */
427
428   }
429
430   /**
431     * @brief SPI1 Initialization Function
432     * @param None
433     * @retval None
434     */
435   static void MX_SPI1_Init(void)
436   {
437
438     /* USER CODE BEGIN SPI1_Init 0 */
439
440     /* USER CODE END SPI1_Init 0 */
441
442     /* USER CODE BEGIN SPI1_Init 1 */
443
444     /* USER CODE END SPI1_Init 1 */
445     /* SPI1 parameter configuration*/
446     hspi1.Instance = SPI1;
447     hspi1.Init.Mode = SPI_MODE_MASTER;
448     hspi1.Init.Direction = SPI_DIRECTION_2LINES;
449     hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
450     hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
451     hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
452     hspi1.Init.NSS = SPI_NSS_SOFT;
```

```
453     hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
454     hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
455     hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
456     hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
457     hspi1.Init.CRCPolynomial = 7;
458     hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
459     hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
460     if (HAL_SPI_Init(&hspi1) != HAL_OK)
461     {
462       Error_Handler();
463     }
464     /* USER CODE BEGIN SPI1_Init 2 */
465
466     /* USER CODE END SPI1_Init 2 */
467
468   }
469
470   /**
471     * @brief TIM6 Initialization Function
472     * @param None
473     * @retval None
474     */
475   static void MX_TIM6_Init(void)
476   {
477
478     /* USER CODE BEGIN TIM6_Init 0 */
479
480     /* USER CODE END TIM6_Init 0 */
481
482     TIM_MasterConfigTypeDef sMasterConfig = {0};
483
484     /* USER CODE BEGIN TIM6_Init 1 */
485
486     /* USER CODE END TIM6_Init 1 */
487     htim6.Instance = TIM6;
488     htim6.Init.Prescaler = 1;
489     htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
490     htim6.Init.Period = 39999;
491     htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
492     if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
493     {
494       Error_Handler();
495     }
496     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
497     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
498     if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
499     {
500       Error_Handler();
501     }
502     /* USER CODE BEGIN TIM6_Init 2 */
503
504     /* USER CODE END TIM6_Init 2 */
505
506   }
507
508   /**
509     * @brief USART1 Initialization Function
510     * @param None
511     * @retval None
512     */
513   static void MX_USART1_UART_Init(void)
514   {
515
516     /* USER CODE BEGIN USART1_Init 0 */
517
518     /* USER CODE END USART1_Init 0 */
519
520     /* USER CODE BEGIN USART1_Init 1 */
521
522     /* USER CODE END USART1_Init 1 */
523     huart1.Instance = USART1;
524     huart1.Init.BaudRate = 115200;
525     huart1.Init.WordLength = UART_WORDLENGTH_8B;
526     huart1.Init.StopBits = UART_STOPBITS_1;
527     huart1.Init.Parity = UART_PARITY_NONE;
528     huart1.Init.Mode = UART_MODE_TX_RX;
529     huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
```

```
530     huart1.Init.OverSampling = UART_OVERSAMPLING_16;
531     huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
532     huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
533     if (HAL_UART_Init(&huart1) != HAL_OK)
534     {
535       Error_Handler();
536     }
537     /* USER CODE BEGIN USART1_Init 2 */
538
539     /* USER CODE END USART1_Init 2 */
540
541   }
542
543   /**
544     * @brief GPIO Initialization Function
545     * @param None
546     * @retval None
547     */
548   static void MX_GPIO_Init(void)
549   {
550     GPIO_InitTypeDef GPIO_InitStruct = {0};
551   /* USER CODE BEGIN MX_GPIO_Init_1 */
552   /* USER CODE END MX_GPIO_Init_1 */
553
554     /* GPIO Ports Clock Enable */
555     __HAL_RCC_GPIOC_CLK_ENABLE();
556     __HAL_RCC_GPIOF_CLK_ENABLE();
557     __HAL_RCC_GPIOA_CLK_ENABLE();
558     __HAL_RCC_GPIOB_CLK_ENABLE();
559
560     /*Configure GPIO pin Output Level */
561     HAL_GPIO_WritePin(WKUP_ESP_GPIO_Port, WKUP_ESP_Pin, GPIO_PIN_RESET);
562
563     /*Configure GPIO pin Output Level */
564     HAL_GPIO_WritePin(GPIOA, RST_Pin|DC_Pin|EN_VBAT_Pin, GPIO_PIN_RESET);
565
566     /*Configure GPIO pin Output Level */
567     HAL_GPIO_WritePin(SPI_CS_GPIO_Port, SPI_CS_Pin, GPIO_PIN_SET);
568
569     /*Configure GPIO pin Output Level */
570     HAL_GPIO_WritePin(GPIOB, LED_RED_Pin|ALARME_Pin, GPIO_PIN_RESET);
571
572     /*Configure GPIO pin Output Level */
573     HAL_GPIO_WritePin(ESP_EN_GPIO_Port, ESP_EN_Pin, GPIO_PIN_SET);
574
575     /*Configure GPIO pin : WKUP_ESP_Pin */
576     GPIO_InitStruct.Pin = WKUP_ESP_Pin;
577     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
578     GPIO_InitStruct.Pull = GPIO_NOPULL;
579     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
580     HAL_GPIO_Init(WKUP_ESP_GPIO_Port, &GPIO_InitStruct);
581
582     /*Configure GPIO pins : RST_Pin DC_Pin SPI_CS_Pin EN_VBAT_Pin */
583     GPIO_InitStruct.Pin = RST_Pin|DC_Pin|SPI_CS_Pin|EN_VBAT_Pin;
584     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
585     GPIO_InitStruct.Pull = GPIO_NOPULL;
586     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
587     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
588
589     /*Configure GPIO pin : BUSY_Pin */
590     GPIO_InitStruct.Pin = BUSY_Pin;
591     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
592     GPIO_InitStruct.Pull = GPIO_NOPULL;
593     HAL_GPIO_Init(BUSY_GPIO_Port, &GPIO_InitStruct);
594
595     /*Configure GPIO pins : LED_RED_Pin ALARME_Pin ESP_EN_Pin */
596     GPIO_InitStruct.Pin = LED_RED_Pin|ALARME_Pin|ESP_EN_Pin;
597     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
598     GPIO_InitStruct.Pull = GPIO_NOPULL;
599     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
600     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
601
602   /* USER CODE BEGIN MX_GPIO_Init_2 */
603   /* USER CODE END MX_GPIO_Init_2 */
604   }
605
606   /* USER CODE BEGIN 4 */
```

```
607
608   ////-------------------------------------------------------------------------------
609
610   //// Fonction Initialisation
611   //// Description: Logique utilisée pour initialiser le système (initialisation des variables,
      calibration ADC, initialisation de la gestion du mode sommeil)
612   //// Entrées: -
613   //// Sorties: -
614   void Initialisation()
615   {
616     mesures.etatSeuils = INDEFINI; // Pour s'assurer qu'un changement d'état de seuils aura bien lieu
      au lancement du programme
617
618     infoBatterie.etatBatterie = INDEFINI; // Pour s'assurer qu'un changement d'état de batterie aura
      bien lieu au lancement du programme
619     infoBatterie.compteurCheckBatterie = 0; // Pour s'assurer qu'une vérification de l'état de la
      batterie ait lieu au lancement du programme
620
621     HAL_ADCEx_Calibration_Start(&hadc); // Calibration ADC (utilisé pour vérifier l'état de la batterie)
622
623     InitialiserGestionSommeil();
624   }
625
626   //// Fonction HAL_TIM_PeriodElapsedCallback: interruption Timer
627   //// Description: fonction d'interruptioin du timer (délai entre activation transistor et mesure de
      l'état de la batterie)
628   //// Entrées: Pointeur:TIM_HandleTypeDef htim
629   //// Sorties: -
630   void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
631   {
632     // Decompte du compteur utilisé pour le délai entre l'activation du transistor et la mesure de
      l'état de la batterie
633     if(etatSTM == MESURER_BATTERIE)
634     {
635       if(infoBatterie.compteurCheckBatterie > 0)
636       {
637         infoBatterie.compteurCheckBatterie--;
638       }
639
640     // Quand le compteur vaudra 0
641     // Ce sera le moment de mesurer la tension de la batterie (à la prochaine exécution de la boucle
      main)
642     }
643   }
644
645   //// Fonction EXTI4_15_IRQHandler: interruption pins GPIO
646   //// Description: fonction d'interruptioin de la pin utilisée par l'ESP pour réveiller le STM
647   //// Entrées: -
648   //// Sorties: -
649   void EXTI4_15_IRQHandler(void)
650   {
651     if(HAL_GPIO_ReadPin(REVEIL_GPIO_Port, REVEIL_Pin) == GPIO_PIN_SET)
652     {
653       receptionTramesEspTermine = true; // Indique que l'ESP à fini d'envoyer ses éventuelles trames
      UART
654     }
655
656     // Pour s'assurer que l'état de l'interruption soit bien réinitialisé
657     HAL_NVIC_ClearPendingIRQ(EXTI4_15_IRQn);
658     __HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_PIN_5);
659   }
660
661   /* USER CODE END 4 */
662
663   /**
664     * @brief  This function is executed in case of error occurrence.
665     * @retval None
666     */
667   void Error_Handler(void)
668   {
669     /* USER CODE BEGIN Error_Handler_Debug */
670     /* User can add his own implementation to report the HAL error return state */
671     __disable_irq();
672     while (1)
673     {
674     }
675     /* USER CODE END Error_Handler_Debug */
```

```
676   }
677
678   #ifdef  USE_FULL_ASSERT
679   /**
680     * @brief  Reports the name of the source file and the source line number
681     *         where the assert_param error has occurred.
682     * @param  file: pointer to the source file name
683     * @param  line: assert_param error line source number
684     * @retval None
685     */
686   void assert_failed(uint8_t *file, uint32_t line)
687   {
688     /* USER CODE BEGIN 6 */
689     /* User can add his own implementation to report the file name and line number,
690        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
691     /* USER CODE END 6 */
692   }
693   #endif /* USE_FULL_ASSERT */
694
```