```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"
#include "cmsis_os.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <string.h>
#include <stdio.h>
#include <stdint.h>
#include "stdlib.h"
#include "DS18B20.h"
#include "NMEA.h"
#include "UartRingbuffer_multi.h"
#include "MY_NRF24.h"
#include "flash.h"
#define FL_address 0x800FC00
/* 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 ---------------------------------------------------------*/
SPI_HandleTypeDef hspi1;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;

```

```
  59  UART_HandleTypeDef huart1;
  60  UART_HandleTypeDef huart3;
  61  DMA_HandleTypeDef hdma_usart3_rx;
  62
  63  osThreadId Task1Handle;
  64  osThreadId Task2Handle;
  65  osThreadId Task3Handle;
  66  /* USER CODE BEGIN PV */
  67  #define gps_uart &huart1 // gps_uart
  68  char GGA[100];
  69  char RMC[100];
  70  char GPS_Time[10];
  71  char GPS_Date[6];
  72  char Lat[9];
  73  char Long[9];
  74  GPSSTRUCT gpsData;
  75  /* USER CODE END PV */
  76
  77  /* Private function prototypes -----------------------------------------------*/
  78  void SystemClock_Config(void);
  79  static void MX_GPIO_Init(void);
  80  static void MX_DMA_Init(void);
  81  static void MX_TIM4_Init(void);
  82  static void MX_USART1_UART_Init(void);
  83  static void MX_SPI1_Init(void);
  84  static void MX_USART3_UART_Init(void);
  85  static void MX_TIM2_Init(void);
  86  static void MX_TIM3_Init(void);
  87  void StartTask1(void const * argument);
  88  void StartTask2(void const * argument);
  89  void StartTask3(void const * argument);
  90
  91  /* USER CODE BEGIN PFP */
  92  void SIMTransmit(char *cmd, uint16_t timeout);
  93  void serverPost(uint16_t timeout);
  94  void serverGet(uint16_t timeout);
  95  void GPStask(void);
  96  void Xuat_PWM(TIM_HandleTypeDef *htim, uint32_t Channel, float Duty_Cycle);
  97  void Dieukhiennhietdo(void);
  98  /* USER CODE END PFP */
  99
 100  /* Private user code ---------------------------------------------------------*/
 101  /* USER CODE BEGIN 0 */
 102  char GGA[100];
 103  GPSSTRUCT gpsData;
 104  DS18B20_Name DS1;
 105  float Temp;
 106  const char apn[]  = "v-internet";
 107  const char apiKey[]  = "YNBZKTS5NRS2YNU6";
 108  const uint32_t timeOut =10000;
 109  const char server[] = "184.106.153.149";
 110  const int  port = 80;
 111  const int lenght = 200;
 112  char ATcommand[100];
 113  uint8_t buffer[250];
 114  uint8_t ATisOK = 0;
 115  uint8_t CGREGisOK = 0;
 116  uint8_t CIPOPENisOK = 0;
```

```
117   uint8_t NETOPENisOK = 0;
118   uint32_t previousTick;
119   char postData[200];
120   char server_set_temp[2];
121
122   uint64_t addr = 0x0011223344;
123   uint8_t TXdata1[10];
124   uint8_t TXdata2[10];
125   uint8_t ACKpayload[5];
126   char setVal[5];
127   float t1,t3,t2;
128   int vt, i, dem=0;
129
130   volatile float giatrido_hientai, E = 0, E1 = 0, E2 = 0;
131   volatile static double alpha = 0, beta = 0,gamma = 0;
132   volatile float setpoint, Kp = 2, Ki = 10, Kd = 0.01;
133   volatile float Delta_T = 0.01;
134   volatile double output,Lastoutput;
135   /* USER CODE END 0 */
136
137   /**
138     * @brief  The application entry point.
139     * @retval int
140     */
141   int main(void)
142   {
143     /* USER CODE BEGIN 1 */
144
145     /* USER CODE END 1 */
146
147     /* MCU Configuration--------------------------------------------------------*/
148
149     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
150     HAL_Init();
151
152     /* USER CODE BEGIN Init */
153
154
155     /* USER CODE END Init */
156
157     /* Configure the system clock */
158     SystemClock_Config();
159
160     /* USER CODE BEGIN SysInit */
161
162     /* USER CODE END SysInit */
163
164     /* Initialize all configured peripherals */
165     MX_GPIO_Init();
166     MX_DMA_Init();
167     MX_TIM4_Init();
168     MX_USART1_UART_Init();
169     MX_SPI1_Init();
170     MX_USART3_UART_Init();
171     MX_TIM2_Init();
172     MX_TIM3_Init();
173     /* USER CODE BEGIN 2 */
174     DS18B20_Init(&DS1, &htim3, GPIOB, GPIO_PIN_15);
```

```
175     Ringbuf_init();
176     HAL_Delay(500);
177
178     NRF24_begin(GPIOA,GPIO_PIN_4,GPIO_PIN_3,hspi1);
179     nrf24_DebugUART_Init(huart1);
180     NRF24_stopListening();
181     NRF24_openWritingPipe(addr);
182     NRF24_setAutoAck(true);
183     NRF24_setChannel(1);
184     NRF24_setPayloadSize(10);
185
186     NRF24_enableDynamicPayloads();
187     NRF24_enableAckPayload();
188     TXdata2[0] = '*';
189     t1 = flashReadFloat(FL_address);
190     t3 = t1;
191     HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
192     HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
193     HAL_TIM_Base_Start(&htim2);
194     HAL_UART_Receive_DMA(&huart3,buffer,sizeof(buffer));
195   //  SIMTransmit("AT+IPR=9600",1000);
196   //  huart3.Init.BaudRate = 9600;
197     HAL_Delay(1000);
198     /* USER CODE END 2 */
199
200     /* USER CODE BEGIN RTOS_MUTEX */
201     /* add mutexes, ... */
202     /* USER CODE END RTOS_MUTEX */
203
204     /* USER CODE BEGIN RTOS_SEMAPHORES */
205     /* add semaphores, ... */
206     /* USER CODE END RTOS_SEMAPHORES */
207
208     /* USER CODE BEGIN RTOS_TIMERS */
209     /* start timers, add new ones, ... */
210     /* USER CODE END RTOS_TIMERS */
211
212     /* USER CODE BEGIN RTOS_QUEUES */
213     /* add queues, ... */
214     /* USER CODE END RTOS_QUEUES */
215
216     /* Create the thread(s) */
217     /* definition and creation of Task1 */
218     osThreadDef(Task1, StartTask1, osPriorityNormal, 0, 128);
219     Task1Handle = osThreadCreate(osThread(Task1), NULL);
220
221     /* definition and creation of Task2 */
222     osThreadDef(Task2, StartTask2, osPriorityBelowNormal, 0, 128);
223     Task2Handle = osThreadCreate(osThread(Task2), NULL);
224
225     /* definition and creation of Task3 */
226     osThreadDef(Task3, StartTask3, osPriorityLow, 0, 128);
227     Task3Handle = osThreadCreate(osThread(Task3), NULL);
228
229     /* USER CODE BEGIN RTOS_THREADS */
230     /* add threads, ... */
231     /* USER CODE END RTOS_THREADS */
232
```

```
233     /* Start scheduler */
234     osKernelStart();
235 
236     /* We should never get here as control is now taken by the scheduler */
237     /* Infinite loop */
238     /* USER CODE BEGIN WHILE */
239 
240     while (1)
241     {
242       /* USER CODE END WHILE */
243 
244       /* USER CODE BEGIN 3 */
245     }
246     /* USER CODE END 3 */
247   }
248 
249   /**
250     * @brief System Clock Configuration
251     * @retval None
252     */
253   void SystemClock_Config(void)
254   {
255     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
256     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
257 
258     /** Initializes the RCC Oscillators according to the specified parameters
259     * in the RCC_OscInitTypeDef structure.
260     */
261     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
262     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
263     RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
264     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
265     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
266     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
267     RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
268     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
269     {
270       Error_Handler();
271     }
272 
273     /** Initializes the CPU, AHB and APB buses clocks
274     */
275     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
276                                   |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
277     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
278     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
279     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
280     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
281 
282     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
283     {
284       Error_Handler();
285     }
286   }
287 
288   /**
289     * @brief SPI1 Initialization Function
290     * @param None
```

```
291     * @retval None
292     */
293   static void MX_SPI1_Init(void)
294   {
295
296     /* USER CODE BEGIN SPI1_Init 0 */
297
298     /* USER CODE END SPI1_Init 0 */
299
300     /* USER CODE BEGIN SPI1_Init 1 */
301
302     /* USER CODE END SPI1_Init 1 */
303     /* SPI1 parameter configuration*/
304     hspi1.Instance = SPI1;
305     hspi1.Init.Mode = SPI_MODE_MASTER;
306     hspi1.Init.Direction = SPI_DIRECTION_2LINES;
307     hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
308     hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
309     hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
310     hspi1.Init.NSS = SPI_NSS_SOFT;
311     hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
312     hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
313     hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
314     hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
315     hspi1.Init.CRCPolynomial = 10;
316     if (HAL_SPI_Init(&hspi1) != HAL_OK)
317     {
318       Error_Handler();
319     }
320     /* USER CODE BEGIN SPI1_Init 2 */
321
322     /* USER CODE END SPI1_Init 2 */
323
324   }
325
326   /**
327     * @brief TIM2 Initialization Function
328     * @param None
329     * @retval None
330     */
331   static void MX_TIM2_Init(void)
332   {
333
334     /* USER CODE BEGIN TIM2_Init 0 */
335
336     /* USER CODE END TIM2_Init 0 */
337
338     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
339     TIM_MasterConfigTypeDef sMasterConfig = {0};
340
341     /* USER CODE BEGIN TIM2_Init 1 */
342
343     /* USER CODE END TIM2_Init 1 */
344     htim2.Instance = TIM2;
345     htim2.Init.Prescaler = 71;
346     htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
347     htim2.Init.Period = 99;
348     htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
```

```
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 71;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 65534;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
```

```
407     {
408       Error_Handler();
409     }
410     /* USER CODE BEGIN TIM3_Init 2 */
411
412     /* USER CODE END TIM3_Init 2 */
413
414   }
415
416   /**
417     * @brief TIM4 Initialization Function
418     * @param None
419     * @retval None
420     */
421   static void MX_TIM4_Init(void)
422   {
423
424     /* USER CODE BEGIN TIM4_Init 0 */
425
426     /* USER CODE END TIM4_Init 0 */
427
428     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
429     TIM_MasterConfigTypeDef sMasterConfig = {0};
430     TIM_OC_InitTypeDef sConfigOC = {0};
431
432     /* USER CODE BEGIN TIM4_Init 1 */
433
434     /* USER CODE END TIM4_Init 1 */
435     htim4.Instance = TIM4;
436     htim4.Init.Prescaler = 71;
437     htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
438     htim4.Init.Period = 999;
439     htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
440     htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
441     if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
442     {
443       Error_Handler();
444     }
445     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
446     if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
447     {
448       Error_Handler();
449     }
450     if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
451     {
452       Error_Handler();
453     }
454     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
455     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
456     if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
457     {
458       Error_Handler();
459     }
460     sConfigOC.OCMode = TIM_OCMODE_PWM1;
461     sConfigOC.Pulse = 0;
462     sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
463     sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
464     if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
```

```
465     {
466       Error_Handler();
467     }
468     if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
469     {
470       Error_Handler();
471     }
472     /* USER CODE BEGIN TIM4_Init 2 */
473
474     /* USER CODE END TIM4_Init 2 */
475     HAL_TIM_MspPostInit(&htim4);
476
477   }
478
479   /**
480     * @brief USART1 Initialization Function
481     * @param None
482     * @retval None
483     */
484   static void MX_USART1_UART_Init(void)
485   {
486
487     /* USER CODE BEGIN USART1_Init 0 */
488
489     /* USER CODE END USART1_Init 0 */
490
491     /* USER CODE BEGIN USART1_Init 1 */
492
493     /* USER CODE END USART1_Init 1 */
494     huart1.Instance = USART1;
495     huart1.Init.BaudRate = 9600;
496     huart1.Init.WordLength = UART_WORDLENGTH_8B;
497     huart1.Init.StopBits = UART_STOPBITS_1;
498     huart1.Init.Parity = UART_PARITY_NONE;
499     huart1.Init.Mode = UART_MODE_TX_RX;
500     huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
501     huart1.Init.OverSampling = UART_OVERSAMPLING_16;
502     if (HAL_UART_Init(&huart1) != HAL_OK)
503     {
504       Error_Handler();
505     }
506     /* USER CODE BEGIN USART1_Init 2 */
507
508     /* USER CODE END USART1_Init 2 */
509
510   }
511
512   /**
513     * @brief USART3 Initialization Function
514     * @param None
515     * @retval None
516     */
517   static void MX_USART3_UART_Init(void)
518   {
519
520     /* USER CODE BEGIN USART3_Init 0 */
521
522     /* USER CODE END USART3_Init 0 */
```

```
523
524     /* USER CODE BEGIN USART3_Init 1 */
525
526     /* USER CODE END USART3_Init 1 */
527     huart3.Instance = USART3;
528     huart3.Init.BaudRate = 115200;
529     huart3.Init.WordLength = UART_WORDLENGTH_8B;
530     huart3.Init.StopBits = UART_STOPBITS_1;
531     huart3.Init.Parity = UART_PARITY_NONE;
532     huart3.Init.Mode = UART_MODE_TX_RX;
533     huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
534     huart3.Init.OverSampling = UART_OVERSAMPLING_16;
535     if (HAL_UART_Init(&huart3) != HAL_OK)
536     {
537       Error_Handler();
538     }
539     /* USER CODE BEGIN USART3_Init 2 */
540
541     /* USER CODE END USART3_Init 2 */
542
543   }
544
545   /**
546     * Enable DMA controller clock
547     */
548   static void MX_DMA_Init(void)
549   {
550
551     /* DMA controller clock enable */
552     __HAL_RCC_DMA1_CLK_ENABLE();
553
554     /* DMA interrupt init */
555     /* DMA1_Channel3_IRQn interrupt configuration */
556     HAL_NVIC_SetPriority(DMA1_Channel3_IRQn, 5, 0);
557     HAL_NVIC_EnableIRQ(DMA1_Channel3_IRQn);
558
559   }
560
561   /**
562     * @brief GPIO Initialization Function
563     * @param None
564     * @retval None
565     */
566   static void MX_GPIO_Init(void)
567   {
568     GPIO_InitTypeDef GPIO_InitStruct = {0};
569   /* USER CODE BEGIN MX_GPIO_Init_1 */
570   /* USER CODE END MX_GPIO_Init_1 */
571
572     /* GPIO Ports Clock Enable */
573     __HAL_RCC_GPIOC_CLK_ENABLE();
574     __HAL_RCC_GPIOD_CLK_ENABLE();
575     __HAL_RCC_GPIOA_CLK_ENABLE();
576     __HAL_RCC_GPIOB_CLK_ENABLE();
577
578     /*Configure GPIO pin Output Level */
579     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
580
```

```
581     /*Configure GPIO pin Output Level */
582     HAL_GPIO_WritePin(GPIOA, SPI1_CE_Pin|SPI1_CNS_Pin, GPIO_PIN_RESET);
583
584     /*Configure GPIO pin Output Level */
585     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, GPIO_PIN_SET);
586
587     /*Configure GPIO pin Output Level */
588     HAL_GPIO_WritePin(DS18B20_GPIO_Port, DS18B20_Pin, GPIO_PIN_RESET);
589
590     /*Configure GPIO pin : PC13 */
591     GPIO_InitStruct.Pin = GPIO_PIN_13;
592     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
593     GPIO_InitStruct.Pull = GPIO_NOPULL;
594     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
595     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
596
597     /*Configure GPIO pins : SPI1_CE_Pin SPI1_CNS_Pin */
598     GPIO_InitStruct.Pin = SPI1_CE_Pin|SPI1_CNS_Pin;
599     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
600     GPIO_InitStruct.Pull = GPIO_NOPULL;
601     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
602     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
603
604     /*Configure GPIO pins : PB12 DS18B20_Pin */
605     GPIO_InitStruct.Pin = GPIO_PIN_12|DS18B20_Pin;
606     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
607     GPIO_InitStruct.Pull = GPIO_NOPULL;
608     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
609     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
610
611   /* USER CODE BEGIN MX_GPIO_Init_2 */
612   /* USER CODE END MX_GPIO_Init_2 */
613   }
614
615   /* USER CODE BEGIN 4 */
616   //void SIMTransmit(char *cmd, uint16_t timeout)
617   // {
618   //  memset(buffer,0,sizeof(buffer));
619   //  HAL_UART_Transmit(&huart3,(uint8_t *)cmd,strlen(cmd),timeout);
620   //  HAL_UART_Receive(&huart3, buffer, 250, timeout);
621   //}
622   void SIMTransmit(char *cmd, uint16_t timeout)
623    {
624     HAL_UART_DMAStop(&huart3);
625     memset(buffer,0,sizeof(buffer));
626     HAL_UART_Transmit(&huart3,(uint8_t *)cmd,strlen(cmd),timeout);
627     HAL_UART_Receive_DMA(&huart3,buffer,sizeof(buffer));
628     HAL_Delay(1000);
629   }
630   void serverPost(uint16_t timeout)
631   {
632     ATisOK = 0;
633     CGREGisOK = 0;
634     NETOPENisOK = 0;
635     CIPOPENisOK = 0;
636     // Check for OK response for AT
637     previousTick =  HAL_GetTick();
638     while(!ATisOK && previousTick  + timeOut >  HAL_GetTick())
```

```
639     {
640       SIMTransmit("AT\r\n",timeout);
641       if(strstr((char *)buffer,"OK"))
642       {
643         ATisOK = 1;
644       }
645     }
646     // Check for network registration.
647     if(ATisOK)
648     {
649       previousTick =  HAL_GetTick();
650       while(!CGREGisOK  && previousTick  + timeOut >  HAL_GetTick())
651       {
652         SIMTransmit("AT+CGREG?\r\n",timeout);
653         if(strstr((char *)buffer,"+CGREG: 0,1"))
654         {
655           CGREGisOK = 1;
656         }
657       }
658     }
659     if(ATisOK && CGREGisOK)
660     {
661       previousTick =  HAL_GetTick();
662       while(!NETOPENisOK  && previousTick  + timeOut >  HAL_GetTick())
663       {
664         SIMTransmit("AT+NETCLOSE\r\n",timeout);
665         sprintf(ATcommand,"AT+CGDCONT=1,\"IP\",\"%s\",\"0.0.0.0\",0,0\r\n",apn);
666         SIMTransmit(ATcommand,timeout);
667         SIMTransmit("AT+NETOPEN\r\n",timeout);
668         SIMTransmit("AT+NETOPEN?\r\n",timeout);
669         if(strstr((char *)buffer,"+NETOPEN: 1"))
670         {
671           NETOPENisOK = 1;
672         }
673       }
674     }
675       // Check for TCP connection
676     if(ATisOK && CGREGisOK && NETOPENisOK)
677     {
678       previousTick =  HAL_GetTick();
679       while(!CIPOPENisOK  && previousTick  + timeOut >  HAL_GetTick())
680       {
681         SIMTransmit("AT+CIPCLOSE=0\r\n",timeout);
682         sprintf(ATcommand,"AT+CIPOPEN=0,\"TCP\",\"%s\",%d\r\n",server,port);
683         SIMTransmit(ATcommand,timeout);
684         if(strstr((char *)buffer,"+CIPOPEN: 0,0"))
685         {
686           CIPOPENisOK = 1;
687 
688         }
689       }
690     }
691     if(ATisOK && CGREGisOK && CIPOPENisOK && NETOPENisOK)
692     {
693       // Perform http request
694       sprintf(postData,"GET
    https://api.thingspeak.com/update?api_key=%s&field1=%.1f&field2=%s&field3=%s&field4=
    %s&field5=%s&field6=%f\r\n",apiKey,Temp,Lat,Long,GPS_Time,GPS_Date,setpoint);
```

```
695       sprintf(ATcommand,"AT+CIPSEND=0,%d\r\n",strlen(postData));
696       SIMTransmit(ATcommand,timeout);
697       if(strstr((char *)buffer,">"))
698       {
699         SIMTransmit(postData,timeout);
700       }
701       // Close connections
702       SIMTransmit("AT+CIPCLOSE=0\r\n",timeout);
703       SIMTransmit("AT+NETCLOSE\r\n",timeout);
704     }
705   }
706   void serverGet(uint16_t timeout)
707   {
708     ATisOK = 0;
709     CGREGisOK = 0;
710     NETOPENisOK = 0;
711     CIPOPENisOK = 0;
712     // Check for OK response for AT
713     previousTick =  HAL_GetTick();
714     while(!ATisOK && previousTick  + timeOut >  HAL_GetTick())
715     {
716       SIMTransmit("AT\r\n",timeout);
717       if(strstr((char *)buffer,"OK"))
718       {
719         ATisOK = 1;
720       }
721     }
722     // Check for network registration.
723     if(ATisOK)
724     {
725       previousTick =  HAL_GetTick();
726       while(!CGREGisOK  && previousTick  + timeOut >  HAL_GetTick())
727       {
728         SIMTransmit("AT+CGREG?\r\n",timeout);
729         if(strstr((char *)buffer,"+CGREG: 0,1"))
730         {
731           CGREGisOK = 1;
732         }
733       }
734     }
735     if(ATisOK && CGREGisOK)
736     {
737       previousTick =  HAL_GetTick();
738       while(!NETOPENisOK  && previousTick  + timeOut >  HAL_GetTick())
739       {
740         SIMTransmit("AT+NETCLOSE\r\n",timeout);
741         sprintf(ATcommand,"AT+CGDCONT=1,\"IP\",\"%s\",\"0.0.0.0\",0,0\r\n",apn);
742         SIMTransmit(ATcommand,timeout);
743         SIMTransmit("AT+NETOPEN\r\n",timeout);
744         SIMTransmit("AT+NETOPEN?\r\n",timeout);
745         if(strstr((char *)buffer,"+NETOPEN: 1"))
746         {
747           NETOPENisOK = 1;
748         }
749       }
750     }
751       // Check for TCP connection
752     if(ATisOK && CGREGisOK && NETOPENisOK)
```

```
753     {
754   //    SIMTransmit("AT+IPADDR\r\n");
755       previousTick =  HAL_GetTick();
756       while(!CIPOPENisOK  && previousTick  + timeOut >  HAL_GetTick())
757       {
758         SIMTransmit("AT+CIPCLOSE=0\r\n",timeout);
759         sprintf(ATcommand,"AT+CIPOPEN=0,\"TCP\",\"%s\",%d\r\n",server,port);
760         SIMTransmit(ATcommand,timeout);
761         if(strstr((char *)buffer,"+CIPOPEN: 0,0"))
762         {
763           CIPOPENisOK = 1;
764 
765         }
766       }
767     }
768     if(ATisOK && CGREGisOK && CIPOPENisOK && NETOPENisOK)
769     {
770       // Perform http request
771       sprintf(postData,"GET
      https://api.thingspeak.com/channels/2169158/fields/7/last?api_key=SLEEWW449CMWYSDI\r
      \n");
772       sprintf(ATcommand,"AT+CIPSEND=0,%d\r\n",strlen(postData));
773       SIMTransmit(ATcommand,timeout);
774       if(strstr((char *)buffer,">"))
775       {
776         SIMTransmit(postData,5000);
777       }
778       if(strstr((char *)buffer,"RECV FROM:"))
779       {
780         for(i=strlen((char *)buffer)-1;i>0;i--)
781         {
782           if(buffer[i]==0x0A) dem++;
783           if(dem==3)
784           {
785             vt = i;
786             dem =0;
787             break;
788           }
789         }
790         for(i=0;i<2;i++)
791           server_set_temp[i] = buffer[vt+i+1];
792         if(atof(server_set_temp)!=0)
793           t1 = atof(server_set_temp);
794       }
795       // Close connections
796       SIMTransmit("AT+CIPCLOSE=0\r\n",timeout);
797       SIMTransmit("AT+NETCLOSE\r\n",timeout);
798     }
799   }
800   void GPStask(void)
801   {
802       if (Wait_for("GGA",gps_uart) == 1) {
803         Copy_upto("*", GGA,gps_uart);
804         decodeGGA(GGA, &gpsData.ggastruct);
805         sprintf(Lat,"%.5f",gpsData.ggastruct.lcation.latitude);
806         sprintf(Long,"%.5f", gpsData.ggastruct.lcation.longtitude);
807       }
808       else
```

```
809       {
810         for(uint8_t i=0;i<9;i++) Lat[i] = '0';
811         for(uint8_t i=0;i<9;i++) Long[i] = '0';
812         for(uint8_t i=0;i<6;i++) GPS_Time[i] = '0';
813       }
814       if (Wait_for("RMC",gps_uart) == 1) {
815         Copy_upto("*", RMC,gps_uart);
816         decodeRMC(RMC, &gpsData.rmcstruct);
817       }
818 
      sprintf(GPS_Time,"%02d%02d%02d",gpsData.ggastruct.tim.hour,gpsData.ggastruct.tim.min
      ,gpsData.ggastruct.tim.sec);
819 
      sprintf(GPS_Date,"%02d%02d%02d",gpsData.rmcstruct.date.Day,gpsData.rmcstruct.date.Mo
      n,gpsData.rmcstruct.date.Yr);
820   }
821   void Xuat_PWM(TIM_HandleTypeDef *htim, uint32_t Channel, float Duty_Cycle)
822   {
823     Duty_Cycle = Duty_Cycle / 100 * htim->Instance->ARR;
824     __HAL_TIM_SET_COMPARE(htim, Channel, (uint16_t)Duty_Cycle);
825   }
826   void Dieukhiennhietdo()
827   {
828     if(output >= 0){
829 
830       Xuat_PWM(&htim4,TIM_CHANNEL_1,output);
831       Xuat_PWM(&htim4,TIM_CHANNEL_2,output);
832     }
833     if(output< 0){
834 
835       Xuat_PWM(&htim4,TIM_CHANNEL_1,-output);
836       Xuat_PWM(&htim4,TIM_CHANNEL_2,-output);
837     }
838   }
839   /* USER CODE END 4 */
840 
841   /* USER CODE BEGIN Header_StartTask1 */
842   /**
843     * @brief  Function implementing the Task1 thread.
844     * @param  argument: Not used
845     * @retval None
846     */
847   /* USER CODE END Header_StartTask1 */
848   void StartTask1(void const * argument)
849   {
850     /* USER CODE BEGIN 5 */
851     /* Infinite loop */
852     for(;;)
853     {
854       Temp = DS18B20_ReadTemp(&DS1);
855       sprintf((char *)TXdata1,"%.1f",Temp);
856       if(t1 != setpoint && t1 != t3)
857       {
858         t3 = t1;
859         flashErase(FL_address);
860         flashWriteFloat(FL_address,t1);
861         for(uint8_t i=0;i<5;i++)
862         {
```

```
863           TXdata2[i+1] = server_set_temp[i];
864         }
865         NRF24_write(TXdata2,10);
866       }
867       if(NRF24_write(TXdata1,10))
868       {
869         NRF24_read(ACKpayload,5);
870         HAL_GPIO_TogglePin(GPIOC,GPIO_PIN_13);
871         if(ACKpayload[0]!='s')
872         {
873           for(uint8_t i=0;i<5;i++)
874           setVal[i] = ACKpayload[i];
875           if(ACKpayload[0] == ' ')
876             setVal[0] = '+';
877         }
878         setpoint = atof(setVal);
879       }
880       else  setpoint = t1;
881       HAL_TIM_PeriodElapsedCallback(&htim2);
882       osDelay(1000);
883     }
884     /* USER CODE END 5 */
885   }
886
887   /* USER CODE BEGIN Header_StartTask2 */
888   /**
889   * @brief Function implementing the Task2 thread.
890   * @param argument: Not used
891   * @retval None
892   */
893   /* USER CODE END Header_StartTask2 */
894   void StartTask2(void const * argument)
895   {
896     /* USER CODE BEGIN StartTask2 */
897     /* Infinite loop */
898     for(;;)
899     {
900       GPStask();
901       osDelay(10000);
902     }
903     /* USER CODE END StartTask2 */
904   }
905
906   /* USER CODE BEGIN Header_StartTask3 */
907   /**
908   * @brief Function implementing the Task3 thread.
909   * @param argument: Not used
910   * @retval None
911   */
912   /* USER CODE END Header_StartTask3 */
913   void StartTask3(void const * argument)
914   {
915     /* USER CODE BEGIN StartTask3 */
916     /* Infinite loop */
917     for(;;)
918     {
919       serverPost(1000);
920       serverGet(1000);
```

```
921       //osDelay(2000);
922     }
923     /* USER CODE END StartTask3 */
924   }
925
926   /**
927     * @brief  Period elapsed callback in non blocking mode
928     * @note   This function is called  when TIM1 interrupt took place, inside
929     * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
930     * a global variable "uwTick" used as application time base.
931     * @param  htim : TIM handle
932     * @retval None
933     */
934   void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
935   {
936     /* USER CODE BEGIN Callback 0 */
937     if (htim == &htim2)
938     {
939       if(Temp < 50)
940       {
941       giatrido_hientai = Temp;
942       E = setpoint - giatrido_hientai;
943       alpha = (2*Delta_T*Kp) + (Ki*Delta_T*Delta_T)+(2*Kd);
944       beta = (Delta_T*Delta_T*Ki) - 4*Kd - (2*Delta_T*Kp);
945       gamma = 2*Kd;
946       output = (alpha*E + beta*E1 + gamma*E2 + 2*Delta_T*Lastoutput)/(2*Delta_T);
947       if (output > 100)
948       output = 100;
949       if (output < -100)
950       output = -100;
951       Lastoutput = output;
952       E1 = E;
953       E2 = E1;
954       Dieukhiennhietdo();
955       }
956
957     }
958     /* USER CODE END Callback 0 */
959     if (htim->Instance == TIM1) {
960       HAL_IncTick();
961     }
962     /* USER CODE BEGIN Callback 1 */
963
964     /* USER CODE END Callback 1 */
965   }
966
967   /**
968     * @brief  This function is executed in case of error occurrence.
969     * @retval None
970     */
971   void Error_Handler(void)
972   {
973     /* USER CODE BEGIN Error_Handler_Debug */
974     /* User can add his own implementation to report the HAL error return state */
975     __disable_irq();
976     while (1)
977     {
978     }
```

```
979     /* USER CODE END Error_Handler_Debug */
980   }
981
982   #ifdef  USE_FULL_ASSERT
983   /**
984     * @brief  Reports the name of the source file and the source line number
985     *         where the assert_param error has occurred.
986     * @param  file: pointer to the source file name
987     * @param  line: assert_param error line source number
988     * @retval None
989     */
990   void assert_failed(uint8_t *file, uint32_t line)
991   {
992     /* USER CODE BEGIN 6 */
993     /* User can add his own implementation to report the file name and line number,
994        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
995     /* USER CODE END 6 */
996   }
997   #endif /* USE_FULL_ASSERT */
998
```