HITOS 3 Y 4 TRABAJO FINAL – RTOS 1 – CESE
R. OLIVA – v 18-04-2019

CONTENIDOS:

1. Introducción
2. Avances logrados Hito 3
3. Avances Hito 4

1. **Introducción**: El proyecto prevé completar una parte de la integración de FreeRTOS al Proyecto Final iniciado en 2018 *"Registrador industrial con soporte de placas periféricas"*, y según se indicó en el documento previo de Alcances, se prevé lograr una decodificación del paquete remitido por el módulo METEO vía RS485, y su envío como datos formateados a través de una cola a la tarea Interfaz_Usuario(). La misma las envía al LCD y (no mostrado) a la UART principal. El presionado de las Teclas (conectadas a un ISR y tarea Debounce no mostrada) modifica el Buffer que es mostrado en el LCD.

   Los Hitos 3 y 4 indicados en la planilla de Trabajo Final RTOS1 son los siguientes:

   Hito 3: Definir colas de mensaje, espacios de memoria compartida y otros elementos de comunicación

   Hito 4: Implementar el uso de recursos de HW

2. **Hito 3 avances:** Se definió la siguiente distribución (Figura 1)

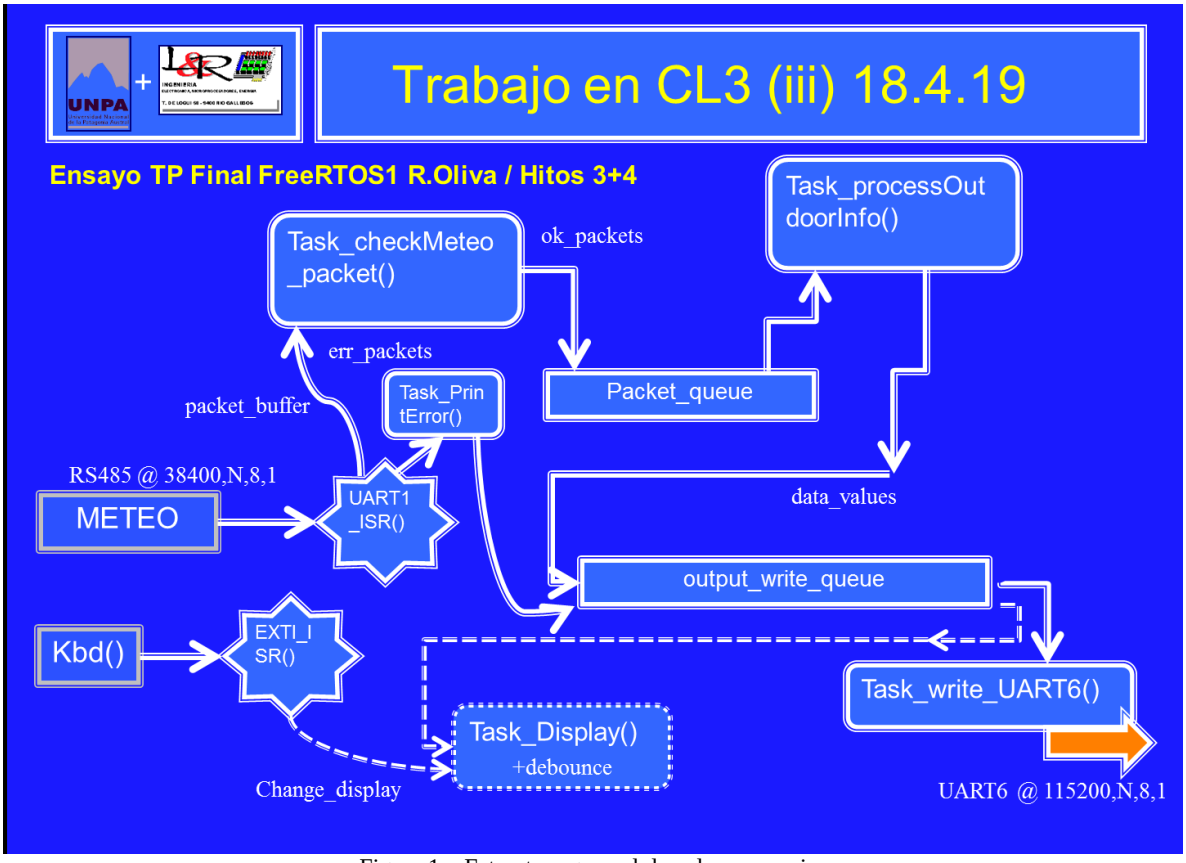

Figura 1 – Estructura general de colas y espacios

2.1 Componente METEO / UART1

En la implementación usual de estos equipos, el paquete de datos meteorológicos que se indicó (Figura 2) llega en forma autónoma aproximadamente a 1 Hz de frecuencia (1 paquete /segundo), y los equipos trabajan a 38400 baud, 8 bits, 1 bit de parada y sin paridad, con un simple checksum algebraico al final del paquete.

```
METEO PACKET STRUCTURE FROM METEO v20 (R.Oliva)
 / Thies uses higher freq, send 'SSSSS'
instead of 'SSS' :
->  (Temperature is Txd in 100*K = 100*(T°C+273.15) =
                    100*(273.15+20.1)=29335):
char    TestCStr[] = "UUU$29335.10156.2562.15100.125.1095*QQQ";
So the sscanf() function is %5d for the first.. (total 39c)

For v20 Thies- 2014 :
Maintains same packet "UUU$ttttt.bbbbb.dddd.sssss.vvv.CRCC*QQQ" but:
    ttttt is 00000 08191 Raw Temperature ADC reading,
                    can be 0-5V( Direct sensor with G=2) or 1-5V (4-20mA)
    bbbbb is 00000 08191 Raw BaroPressure ADC reading,
                    can be 0-5V( Direct sensor with G=1) or 1-5V (4-20mA)
    dddd  is 0000 to 3600, WDIR*10 in UWORD
    sssss   is 00000 to 99999, 0 to 9999.9Hz (no scaling) from Thies Anemometer.
   (15100 reading would be 1510Hz, or Typical: V= 0.04597*1510+0.21=69.6m/s)
    vvv   was voltage, not used.
    CRCC  is simple checksum
```

Figura 2 – Estructura del Paquete a ser recibido a una frecuecia de 1Hz.

2.2 UART1 Handler

Es el primer procesamiento de los datos que ingresan. En la versión de 8 bits se implementaba un buffer de 96 bytes, capaz de alojar 2 paquetes. En la presente versión se usan las rutinas LL de STM32, para chequear el flag "RX Not Empty" de la UART. Si hay datos, se leen y se copian a un buffer global packet_buff, con un puntero packet_len . La detección es por el último carácter '*' y la longitud adecuada, en cuyo caso se notifica a vTask_CheckMeteo_Packet() para que tome los datos y los copie, sinó se notifica al printError task. En ambos casos se resetea el puntero al inicio (buffer_len = 0).

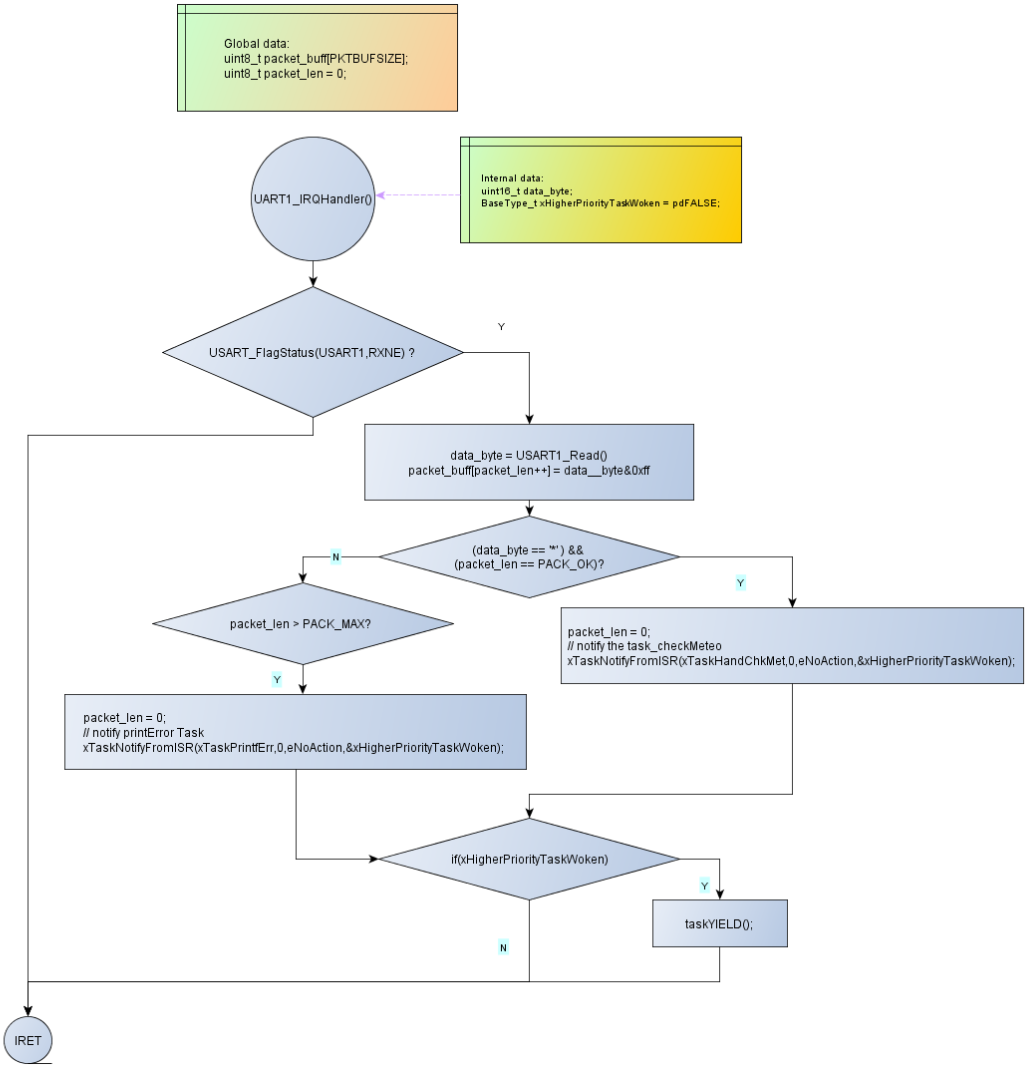

Figura 3 – Estructura del ISR UART1 Handler

2.2.1 Implementación del UART1_Handler:

```
// USART1_ISR R.Oliva 18.4.2019
// Recibe datos de METEO a 38400 por USART1
// Global data:
// uint8_t packet_buffer[PKTBUFSIZE];
// uint8_t packet_len = 0;

// Packet:
// v6 for use with METEO v20 uses packet "UUU$ttttt.bbbbb.dddd.sssss.vvv.CRCC*QQQ":  (WSpeed is 5 chars long)
//      UUU$   start identifier
//      ttttt is 00000 08191 Raw Temperature ADC reading, can be 0-5V( Direct sensor with G=2) or 1-5V (4-20mA)
//      bbbbb is 00000 08191 Raw BaroPressure ADC reading, can be 0-5V( Direct sensor with G=1) or 1-5V (4-20mA)
//      dddd  is 0000 to 3600, WDIR*10 in UWORD
//      sssss is 00000 to 99999 from Anemometer / Thies.
//      vvv   was voltage, not used.
//      CRCC  is simple checksum
//      *QQQ  end identifier
//      Total length from $ is: 5+1+5+1+4+1+5+1+3+1+4= then '*' =15+7+4+5 =31
//      Total length form first U is 31+4 = 35
//      TestCStr[] = "UUU$29335.10156.2562.15100.125.1095*QQQ";
//  Parsed originally (after $ detected) with
//      c = (char)sscanf(packet,"%5d.%5d.%4d.%5d.%3d.%4x",&out_t,
//              &out_b, &out_w_dir,&out_w_speed,
//              &out_vbat,&checksum);

void USART1_IRQHandler(void)
{
  /* USER CODE BEGIN USART1_IRQn 0 */
        uint16_t data_byte;
        BaseType_t xHigherPriorityTaskWoken = pdFALSE;

        if( USART_GetFlagStatus(USART1,USART_FLAG_RXNE) )
        {
                //a data byte is received from the user
                data_byte = USART_ReceiveData(USART1);

                packet_buffer[packet_len++] = (data_byte & 0xFF) ;

                if((data_byte == '*' ) && (packet_len == PACK_OK_LEN))
                {
                        //then packet is ok..
                        //reset the packet_len variable
                        packet_len = 0;

                        //notify the CheckMeteoHand task
                        xTaskNotifyFromISR(xTaskHandChkMet,0,eNoAction,&xHigherPriorityTaskWoken);
                }
                else if(packet_len > PACK_MAX)
                {
                        //then packet is corrupt..
                        //reset the packet_len variable
                        packet_len = 0;
                        // notify printError Task
                        xTaskNotifyFromISR(xTaskPrintfErr,0,eNoAction,&xHigherPriorityTaskWoken);
                }
                else{
                        // do nothing
                }

        }
        // if the above freertos apis wake up any higher priority task, then yield the processor to the
        //higher priority task which is just woken up.

        if(xHigherPriorityTaskWoken)
        {
                taskYIELD();
        }
  /* USER CODE END USART1_IRQn 0 */
  /* USER CODE BEGIN USART1_IRQn 1 */

  /* USER CODE END USART1_IRQn 1 */
}
```

2.3 Task_CheckMeteo()

El vTask_CheckMeteo_Packet() recibe los paquetes que están OK y copia el buffer de RX de la UART a un buffer "Packet_OK", que se asigna en memoria. Este nuevo paquete se pone en la cola de "paquetes_ok" para procesar por el vTask_Process_OutdoorInfo():

```
void vTask_Check_Meteo_packet(void *params)
{
	// typedef struct PACKET_OK
	// {
	//    uint8_t packet_content[PACK_OK_LEN+1];
	// } PACKET_OK_t;
	PACKET_OK_t *new_packet;


	while(1)
	{
		xTaskNotifyWait(0,0,NULL,portMAX_DELAY);
		//1. allocate space..
		new_packet = (PACKET_OK_t*) pvPortMalloc(sizeof(PACKET_OK_t));

		taskENTER_CRITICAL();
		// From WG Book - Minimal printf facility
		// requires miniprintf.c / .h pair
		// make a copy of the buffer_packet to new_packet
		#ifdef PACKET_COPIAR
		mini_snprintf(new_packet, PACK_OK_LEN,"%s",packet_buffer)
		#else
		// Copiamos uno de muestra
		// char okTestCStr[] = "UUU$29335.10156.2562.15100.125.1095";
		sprintf(new_packet,"%s",okTestCStr);
		#endif
		taskEXIT_CRITICAL();

		//send the packet to the packet queue
		xQueueSend(packet_queue,&new_packet,portMAX_DELAY);

	}

}
```

2.4 vTask_Process_OutdoorInfo()

El vTask_Process_OutdoorInfo () recibe un paquete de la cola de los OK y separa los valores recibidos en las variables enteras `int16_t out_t,out_b, out_w_speed,out_vbat, out_w_dir,checksum;` (todavía no escaladas), imprime en la cola de elementos de salida la cantidad de ítems recibidos, y si el checksum es correcto imprime (por ahora) el valor del viento en la misma cola de salida. Sinó imprime un error.

```
void vTask_Process_OutdoorInfo(void *params)
{
	PACKET_OK_t *new_packet;
	char task_msg[50];
	char out_id[6];
	uint16_t chk = 0;
	int16_t items = 0;
	int16_t out_t,out_b, out_w_speed,out_vbat;
	int16_t out_w_dir,checksum;  // temporary variables
	uint32_t toggle_duration = pdMS_TO_TICKS(500);

	while(1)
	{
		xQueueReceive(packet_queue,(void*)&new_packet,portMAX_DELAY);

		items = sscanf(new_packet,"%4s%5d.%5d.%4d.%5d.%3d.%4x",&out_id,&out_t,
		&out_b, &out_w_dir,&out_w_speed, &out_vbat,&checksum);

		print_items_message(task_msg, items);

		chk = out_t + out_b + out_w_dir;
		chk += out_w_speed + out_vbat;
```

```
            if((chk == checksum) && (items == 7))
                {
                print_Wind_Speed(task_msg, out_w_speed);
                }
                else
                {
                print_error_message(task_msg);
                }

                //lets free the allocated memory for the new packet
                vPortFree(new_packet);

        }
}
```

2.5 vTask_PrintError ()

Esta espera si ocurrió un error en el paquete recibido por el UART1_ISR, y lo reporta. Se podría condensar en otra tarea.

```
void vTask_PrintError(void *params)
{
        char pData[] = "Error en Recepcion"
        while(1)
        {
                xTaskNotifyWait(0,0,NULL,portMAX_DELAY);
                //1. allocate space..
                xQueueSend(output_write_queue,&pData,portMAX_DELAY);
        }

}
```

2.6 vTask_ Write_Uart6 ()

Esta tarea toma un elemento de la cola de salida `output_write_queue` y lo envía a la terminal de monitoreo en UART6 de la placa CL3

```
void vTask_Write_Uart6(void *params)
{
        char *pData = NULL;
        while(1)
        {
        xQueueReceive(output_write_queue,&pData,portMAX_DELAY);
                printmsg(pData);

        }
}
```

2.7 vTask_Display ()

Por ahora esta tarea sólo imprime un menú genérico,

```
//Menu por USART 6
char menu[]={"\
\r\nLectura USART METEO----> 1 \
\r\nEXIT_APP           ----> 0 \
\r\nType your option here : "};
```

y lo envía a la terminal de monitoreo en UART6. La idea es que esté controlado por Teclas y además envíe al display LCD alguna de las salidas.

```
// RTOSi - Implementacion de Task handlers
// TP Final R.Oliva 2019

void vTask_Display(void *params)
{

char *pData = menu;

        while(1)
        {
                xQueueSend(output_write_queue,&pData,portMAX_DELAY);
```

```
                //Esperar indefinidamente.
                xTaskNotifyWait(0,0,NULL,portMAX_DELAY);

        }
}
```

2.8 Funciones de salida auxiliares

Funciones que se utilizan para enviar ítems recibidos, velocidad de viento y error en el checksum o procesamiento del task "outdoor_info":

```
// Funciones de Salida al UART6

// Salida 1 - items recibidos de UART1
void print_items_message(char *task_msg, int16_t items)
{
        sprintf( task_msg,"\r\n Items leidos Outdoor %d\r\n", items);
        xQueueSend(output_write_queue,&task_msg,portMAX_DELAY);
}

// Salida 2 - viento en UART1
void print_Wind_Speed(char *task_msg, int16_t wspeed)
{
        sprintf( task_msg,"\r\n Velocidad de viento %d\r\n", wspeed);
        xQueueSend(output_write_queue,&task_msg,portMAX_DELAY);
}

// Salida 3 - Error en Outdoor Info
void print_error_message(char *task_msg)
{
        sprintf( task_msg,"\r\n Error en OutdoorInfo\r\n");
        xQueueSend(output_write_queue,&task_msg,portMAX_DELAY);
}
```

3. HITO 4: Implementación – versión UART1_5

Se hizo un primer ensayo UART1_5 utilizando las siguientes variables y luego la función main() que se indica a continuación. Todavía hay algunos problemas con las librerías LL_ utilizadas.

```
/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
//global space for some variable
char usr_msg[250]={0};

//task handles
TaskHandle_t xTaskHandleDisplay  = NULL;   // Task 1 Display Handle
TaskHandle_t xTaskHandleChkMeteo = NULL;   // Task 2 CheckMeteo Handle
TaskHandle_t xTaskHandleProcOutd = NULL;   // Task 3 ProcessOutdoorInfo Handle
TaskHandle_t xTaskHandleWrUart6  = NULL;   // Task 4 Write_Uart6 Handle
TaskHandle_t xTaskHandlePrintErr  = NULL;   // Task 5 PrintError

//Queue handle
QueueHandle_t packet_queue       = NULL;   // ok packets queue handle
QueueHandle_t output_write_queue = NULL;   // output write queue handle

//software timer handler
TimerHandle_t led_timer_handle = NULL;

// packet buffer
uint8_t packet_buffer[PKTBUFSIZE];
uint8_t packet_len = 0;
uint8_t command_buffer[20];
uint8_t command_len =0;

//Menu por USART 6
char menu[]={"\
\r\nLectura USART METEO----> 1 \
\r\nEXIT_APP           ----> 0 \
\r\nType your option here : "};


/**
  * @brief  The application entry point.
```

```
  * @retval int
  */
int main(void)
{
  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_USART6_UART_Init();
  MX_USART1_UART_Init();

  sprintf(usr_msg,"\r\n RTOS1 TP Final R.Oliva 2019 \r\n");
  printmsg(usr_msg);

  /* Call init function for freertos objects (in freertos.c) */
  MX_FREERTOS_Init();

  // Create packet queue (up to 10 stripped packets)
  packet_queue = xQueueCreate(10,sizeof(PACKET_OK_t*));

  // Create the write queue (up to 30 chars)
  output_write_queue = xQueueCreate(30,sizeof(char*));



if((packet_queue != NULL) && (output_write_queue != NULL))
        {
                // Create task-1 vTask_Display
                // TaskHandle_t xTaskHandleDisplay -  Task 1 Display Handle
                xTaskCreate(vTask_Display,"TASK_DISPLAY-1",500,NULL,1,&xTaskHandleDisplay);

                // Create task-2 vTask_Check_Meteo_packet
                // TaskHandle_t xTaskHandleChkMeteo - Task 2 CheckMeteo Handle
                xTaskCreate(vTask_Check_Meteo_packet,"TASK_CHK_MET_2",500,NULL,2,&xTaskHandleChkMeteo);

                // Create task-3 vTask_Process_OutdoorInfo
                // TaskHandle_t xTaskHandleProcOutd - Task 3 ProcessOutdoorInfo Handle
                xTaskCreate(vTask_Process_OutdoorInfo,"TASK_PROCESS_OUTD_3",500,NULL,2,&xTaskHandleProcOutd);

                // Create task-4 vTask_Write_Uart6
                // TaskHandle_t xTaskHandleWrUart6  - Task 4 Write_Uart6 Handle
                xTaskCreate(vTask_Write_Uart6,"TASK4-UART-WRITE",500,NULL,2,&xTaskHandleWrUart6);

                // Create task-5 vTask_PrintError
                // TaskHandle_t xPrintError  - Task 5
                xTaskCreate(vTask_PrintError,"TASK5-PRINTERROR",500,NULL,2,&xTaskHandlePrintErr);

            // start the scheduler
            vTaskStartScheduler();
        }  else
        {
                sprintf(usr_msg,"Fallo la creacion de las colas!\r\n");
                printmsg(usr_msg);

   }

  /* We should never get here as control is now taken by the scheduler */
  while (1)
  {
  }
}
```

ANEXO

Figura A0 –Ensayo RS485 con un convertidor RS485 a RS232, y luego a USB - TPFinal RTOSi

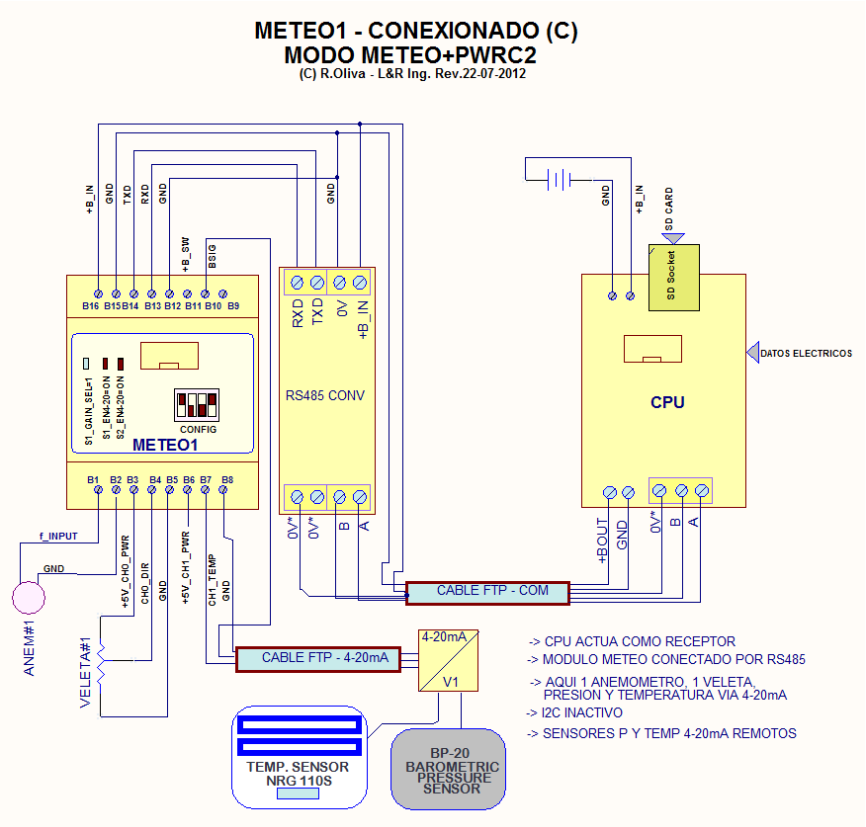

Figura A1 – METEO y módulo de conversión a RS485 – Conexionado a CPU

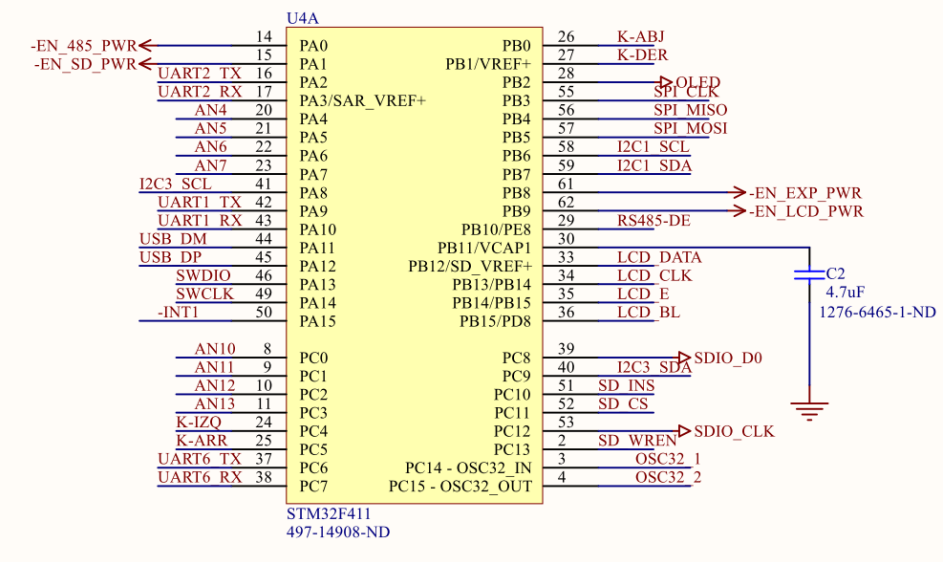

Figura A2 – Asignación de funciones en el STM32F411 de CL3

-0-