Internet de las cosas y conectividad de sistemas embebidos

2022

Trabajo práctico integrador

Alumnos :

Sensor Side

- BARUA Tomas
- KRUGER Jose Luis
- LENCI Juan

Lora Side

- FRUMENTO Leonel
- MARCILLA BONANNO Joaquin
- ROTA Franco

Docentes:

- ROMEO Marcelo
- BACIGALUPO Juan Ignacio

Universidad Nacional de San Martín

Índice

1. Objetivo

El objetivo del proyecto consiste en implementar los conceptos adquiridos en clase, por medio del uso de la placa de desarrollo STM32F401RE en dos partes separadas.

La primera, Sensor Side, comienza con la recepción por UART de una trama que contiene datos de un sensor, se extraen estos datos y se arma una nueva trama llenándola con dirección de origen y destino del mensaje, tamaño en bytes de los datos a enviar, los datos en sí y por último se calcula y se carga el CRC. Con la trama armada y utilizando módulos MAX-RS485 se habilita la transmisión de datos hacia la segunda parte, el Lora Side, por medio del protocolo RS485.

En el Lora Side, se recibe la trama completa y utilizando una máquina de estados se parsea la información extrayendo los datos (siempre y cuando la dirección destino del mensaje coincida con la dirección seteada), se calcula nuevamente el CRC, se lo compara con el enviado y se le envía al Sensor Side un mensaje de Ok (si el CRC enviado coincide con el calculado) o de Error (si no coinciden). Por último utilizando comandos AT, se envía la información recibida a un módulo emisor LORA.

2.Trama

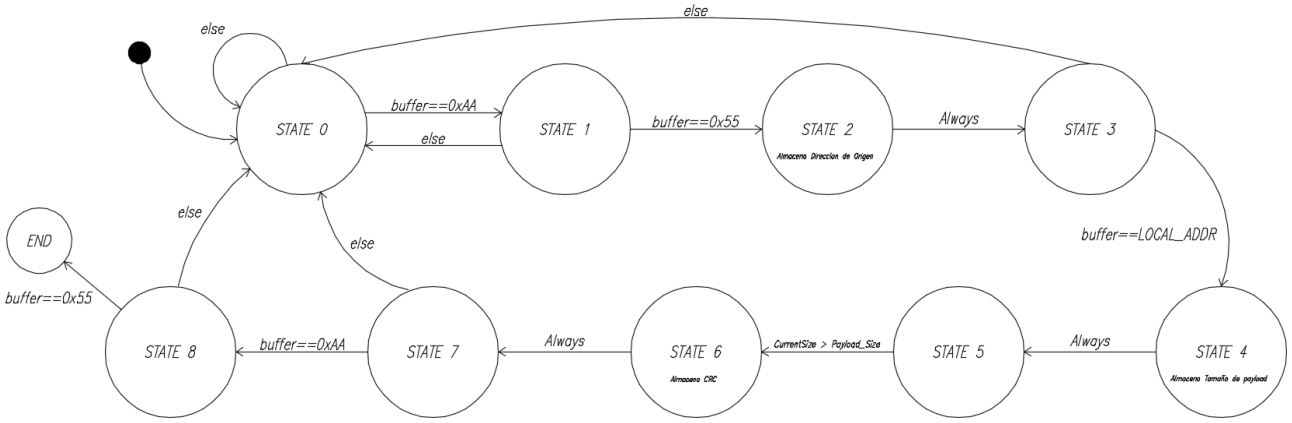
Para la comunicación entre ambos lados, es necesario formar una trama de comunicación, la misma es armada de la siguiente manera:

- **Señal de comienzo de trama:** Cuenta con 2 bytes, 0xAA seguido de 0x55. De no detectar esta secuencia, la máquina de estados de recepción descarta el mensaje.
- **Direcciones**: 2 bytes que contienen la dirección de origen y destino del mensaje.
  El primer byte contiene la dirección de origen del mensaje. El segundo byte contiene la dirección a la que se dirige el mensaje. De no coincidir la dirección de destino enviada con la establecida, la máquina de estados de recepción descarta el mensaje.
  La dirección del Sensor Side es 0x05 y la dirección del Lora Side es 0x06.
- **Tamaño del payload:** 1 byte que contiene el tamaño en bytes del payload.Ell tamaño máximo del payload es de 256 bytes.
- **Payload**: de 1 a 256 bytes, es la información útil del mensaje.
- **Verificación de trama**: Para la verificación de errores, se utilizan 4 bytes correspondientes al CRC32 implementado, que detecta cambios entre los datos de origen y de destino.
- **Señal de finalización de trama**: Cuenta con 2 bytes, 0x55 seguido de 0xAA.

Por lo que la trama tiene un tamaño entre 12 y 266 bytes.

3. Máquina de estados recepción de trama

Para la recepción de la trama se implementó una máquina de estados que permite detectar el mensaje mediante la señal de comienzo, reconocer si el mensaje está dirigido a este dispositivo y almacenar el tamaño del mensaje, el mensaje, CRC y dirección de origen.
La máquina de estados se puede representar de la siguiente manera:



4. Sensor Side

Sensor Side será el encargado de enviar la información útil (payload) hacia el LoRa Side. Una vez finalizada la recepción, quedaremos a la espera de la recepción de un mensaje de confirmación. En caso de recibir “OK” el ciclo finaliza. En caso de recibir “ER” se realizará una retransmisión del último mensaje. En caso de no recibir ninguna respuesta a los 2 segundos del último envío de mensaje se hará una retransmisión del mensaje, repitiendo éste último ciclo hasta recibir respuesta.

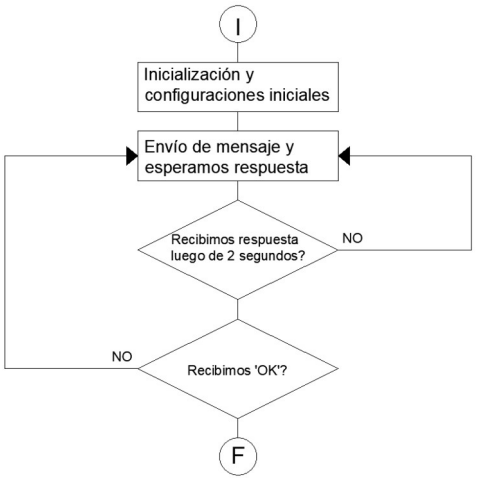


Diagrama de flujo Sensor Side

A continuación se detalla brevemente las funciones principales:

Función toma de datos del sensor

Sintaxis: **receive_sensor(sensor_data);**

Esta función recibe los datos del sensor cuyo payload es de 16 bytes. A diferencia de las demás funciones de comunicación del set-up, esta recibe el mensaje completo. El mismo cuenta con una condición de inicio y final que deben detectarse para extraer la carga útil. La condición de inicio es el valor de los dos primeros bytes como “AA”, y la de fin es “BB” en los dos últimos.

La función que lo implementa es:

```
357 void receive_sensor(uint8_t *data){
358     uint8_t payload[16]={0};
359     uint32_t i,j;
360
361     for(i=0;i<=sizeof(*data);i++){
362         if(*(data+i)=='A' && *(data+i+1)=='A'){
363             for(j=0;j<16;j++){
364                 if(*(data+j)!='B'&& *(data+j+1)!='B')
365                     *(payload + j) = *(data + i + j + 2);
366                 else {
367                     *(payload+j-1)=0;
368                     break;
369                 }
370             }
371         }
372
373     }
374
375 return;
376 }
```

Función Envío de Datos

Sintaxis: **send_data(payload,local_add, destino_add,sizeof(payload));**

La función requiere como dato el puntero del vector del mensaje a enviar, la dirección local, la dirección destino y el tamaño del mensaje.

Esta función se encarga de armar la trama para el envío del payload, donde incluirá los start frame, end frame, dirección de destino, dirección de origen, payload y CRC. Para el cálculo del CRC se necesita vector de variables de 32 bits. para ello debemos asegurarnos de que el largo de nuestra trama sea múltiplo de 32 bits. En caso de no serlo, en conjunto con LoRa Side, se decide agregar “0” al final de la trama.

Para el armado de la trama, la dirección de origen, dirección de destino, tamaño del payload y payload se obtienen del argumento de la función send_data. Los Start Frame y End Frame son valores constantes. Por último el CRC, se calcula una vez armada completamente la trama (las posiciones de la trama reservada para el envío del CRC se completan con ‘0’ antes del cálculo). El cálculo del CRC incluye los ‘0’ necesarios agregados al final de la trama para que el largo sea múltiplo de 32 bits. La API usada para el cálculo es HAL_CRC_Calculate que devuelve un número de 32 bits. Si queremos incluirlo en nuestra trama de variables de 8 bits, debemos truncar el resultado del CRC de 32 bits en 4 partes de 8 bits mediante enmascaramiento y desplazamiento de bits. Finalmente, luego de incluir el resultado del CRC en la trama, realizamos el envío completo mediante la API HAL_UART_Transmit.

```
uint8_t send_data(uint8_t *payload, uint8_t orAddr,uint8_t destAddr,uint8_t size){
    uint8_t START_FRAME[2]= {0xAA,0x55};
    uint8_t END_FRAME[2]= {0x55,0xAA};
    uint8_t maxSize=11+size;
    uint8_t crc[4]={0};
    uint32_t aux_crc=0;
    uint8_t Cant_Ceros_add=0;

    Cant_Ceros_add=4-(11+size)%4;   //determinamos la canitdad de cero a agregar para hacerlo multiplo de 4

    uint8_t trama[maxSize+Cant_Ceros_add];
    unsigned int i;
    unsigned int j;
    for(i=0;i< maxSize;i++){
        if(i==0)
            *(trama+i)= START_FRAME[i];
        if(i==1)
            *(trama+i)= START_FRAME[i];
        if(i==2)
            *(trama+i)= orAddr;
        if(i==3)
            *(trama+i)= destAddr;
        if(i==4)
            *(trama+i)= size;
        if(i==5){
            for(j=0;j<size;j++)
                *(trama+i+j)=*(payload+j);
        }
        if(i==6){
            for(j=0;j<4;j++){
                *(trama+i+j+size-1)=*(crc+j);       //completamos con '0' las posiciones del CRC
            }
        }
        if(i==7)
            *(trama+i+size+2)= END_FRAME[0];
        if(i==8){
                *(trama+i+size+2)= END_FRAME[1]; //
            for(j=0;j<Cant_Ceros_add;j++)
                *(trama+i+j+1+size+2)=0x00;     //agregamos los ceros adicionales al final de la trama para que sea multiplo de 32bits
        }
    }
    aux_crc= HAL_CRC_Calculate(&hcrc,(uint32_t *) trama, sizeof(trama)/sizeof(uint32_t));//calculamos el CRC

    *(trama+8+size)=aux_crc&0xFF000000>>24;         //dividimos los 32 bits del CRC...
    *(trama+7+size)=(aux_crc&0xFF000000>>16)>>8;    //...y lo colocamos en el vector Trama
    *(trama+6+size)=(aux_crc&0xFF000000>>8)>>16;
    *(trama+5+size)=(aux_crc&0xFF000000)>>24;

     HAL_GPIO_WritePin(TR_EN_GPIO_Port, TR_EN_Pin,GPIO_PIN_SET );   //transmision enable
     HAL_Delay(1);
     HAL_UART_Transmit(&huart1, trama, sizeof(trama), HAL_MAX_DELAY);
     HAL_GPIO_WritePin(TR_EN_GPIO_Port, TR_EN_Pin,GPIO_PIN_RESET ); //reception enable
    return 0;

}
```

Función Recepción de Datos:

Sintaxis: **receive_data(void)**

Esta función se encarga de la recepción de los datos enviados por LoRa Side, concretamente se espera recibir un mensaje de 'OK' o 'ER'. Dentro de la función receive_data, disponemos de la API HAL_UART_Receive que se encargará de la recepción de datos por UART. Dentro de dicha API tenemos definido como argumentos:

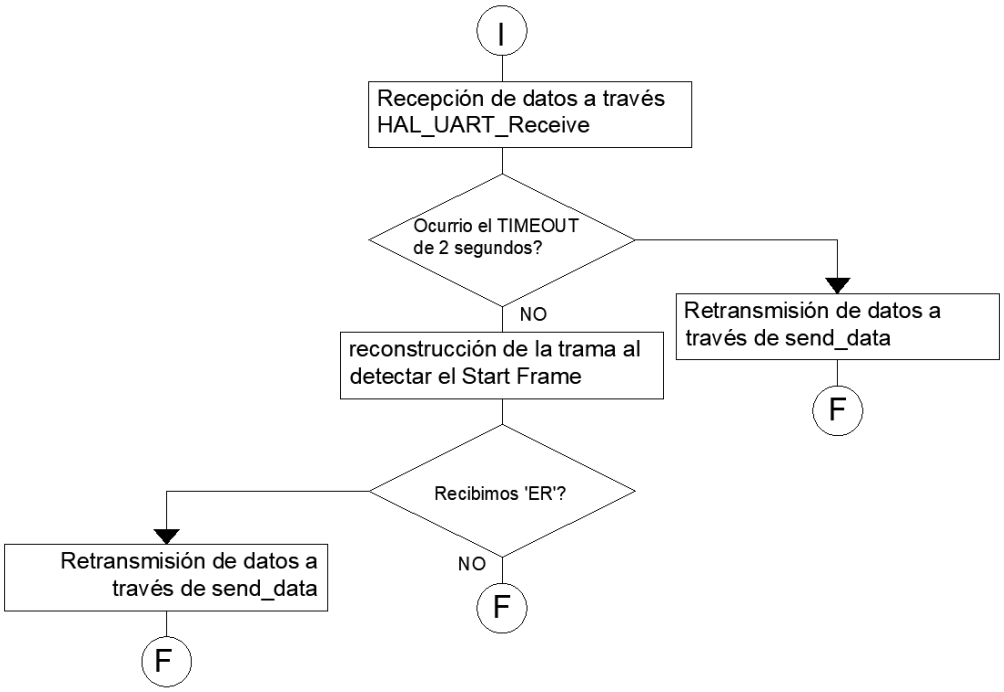
- La UART que usamos para la comunicación, la UART 1 para nuestro caso
- Variable que funcionará de buffer para almacenar los datos recibidos
- el tamaño del buffer (1 byte en este caso)
- TimeOut, es el tiempo máximo que esperaremos para la recepción de un mensaje. Hemos definido un tiempo máximo de 2 segundos.

HAL_UART_Receive devuelve como resultado, el estado de la recepción de datos, siendo posible los siguientes valores:

- HAL_OK
- HAL_ERROR

- HAL_BUSY
- HAL_TIMEOUT

Una vez recibido los datos, los valores comienzan a pasar por la máquinas de estado recepción de trama, en donde al detectar los bytes de start frame comienza con la reconstrucción de la trama enviada por LoRa Side. Una vez finalizada la reconstrucción de la trama, se procede a detectar si el mensaje recibido es 'ER' , ya que en caso de que así lo sea implicaría una retransmisión de los datos.



5. LoRa Side

El código diseñado para dar solución a los requerimientos, fue pensado con una función principal (Main) que se basa en una máquina de estados, donde cada estado llama a funciones distintas para dar solución adecuada a las entradas de diseño pautadas. La máquina de estado es la encargada de detectar y recibir la trama de información enviada por el Sensor Side, que, corrobora mediante un CRC la trama recibida y da aviso al Sensor Side que los datos recibidos son correctos o incorrectos. En el caso que la trama recibida sea correcta, continúa enviando el payload recibido al módulo LoRa.
A continuación incluimos las imágenes de la máquina de estados pensada para Lora Side, donde se observa el código que trabaja sobre la trama recibida.

```
428 while (1)
429  {
430
431    switch (ESTADO)
432    {
433        case ESTADO0:
434            info[0] = CSE_receive_data();
435          if(info[0] == 0xAA) // Compara byte 0xAA de inicio de trama
436              ESTADO = ESTADO2;
437          else
438              ESTADO = ESTADO0;
439          break;
440
441        case ESTADO2:
442            info[1] = CSE_receive_data();
443          if(info[1] == 0x55) // Compara byte 0x55 de inicio de trama
444              ESTADO = ESTADO3;
445          else
446              ESTADO = ESTADO0;
447          break;
448
449         case ESTADO3:
450             info[2] = CSE_receive_data();
451           if(info[2] == 0x05)// Compara byte dirección Origen
452              ESTADO = ESTADO4;
453           else
454               ESTADO = ESTADO0;
455         break;
456
457         case ESTADO4:
458            info[3] = CSE_receive_data();
459          if(info[3] == 0x06)// Compara byte dirección Destino
460             ESTADO = ESTADO5;
461           else
462               ESTADO = ESTADO0;
463          break;
464
465         case ESTADO5:
466            info[4] = CSE_receive_data();
467             size_rx = info[4];// Recibe tamaño de payload
468
469             for(i=0; i < size_rx; i++ ){
```

```
469             for(i=0; i < size_rx; i++ ){
470                 info[5+i] = CSE_receive_data();// Almacena payload
471                 payload_rx[i] = info[5+i];
472             }
473             ESTADO = ESTADO6;
474          break;
475
476         case ESTADO6:
477
478           for(j=0;j < 4; j++){
479               info[5 + size_rx + j] = CSE_receive_data();
480               CRC_rx[j] = info[5 + size_rx + j]; // Almacena CRC
481           }
482              ESTADO = ESTADO7;
483          break;
484
485         case ESTADO7:
486           info[5 + size_rx + 4] = CSE_receive_data();
487           if(info[5 + size_rx + 4] == 0x55) // Compara byte 0x55 de fin de trama
488              ESTADO = ESTADO8;
489           else
490               ESTADO = ESTADO0;
491           break;
492
```

Aquí, desde el Estado 0 al Estado 8 , la máquina de estados corrobora la recepción correcta de la trama, contemplando volver al estado inicial para comenzar nuevamente en caso de recibir un byte en el orden incorrecto pautado según el esquema de la trama.

```
493         case ESTADO8:
494           info[5 + size_rx + 4 + 1] = CSE_receive_data();
495           if(info[5 + size_rx + 4 + 1] == 0xAA) // Compara byte 0xAA de fin de trama
496              ESTADO = ESTADO9;
497           else
498               ESTADO = ESTADO0;
499           break;
500
501         case ESTADO9: // Calculo de CRC
502              crc_obt = Escritura_CRC(0x05, 0x06, size_rx, payload_rx);
503              crc_aux=0;
504              for (i=0 ; i<3 ; i++){
505                       crc_aux = CRC_rx[i] | crc_aux;
506                       crc_aux =  crc_aux << 8 ;
507                   }
508                   crc_aux = CRC_rx[3] | crc_aux;
509              HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, 1); // configuro el RS485 en modo transmision
510              HAL_Delay(1); // delay de 10ms para que tenga tiempo para poder transmitir paquetes
511
```

Luego, en el Estado 9 se calcula el CRC de la trama recibida de acuerdo a los parámetros pasados como argumentos (payload, el tamaño del mismo y las direcciones de origen y destino), mediante la función **Escritura_CRC**. Este CRC calculado se comparará con los 4 bytes de CRC recibidos dentro de la trama para corroborar la integridad de la información recibida.

Para ello, de la trama recibida se extraen los bytes correspondientes al CRC y se desplazan hacia la izquierda en 8 bytes, acumulando en la variable **crc_aux**, que luego servirá para verificar si los datos recibidos desde el medio de transmisión son correctos o no.

```
512             if(crc_obt == crc_aux)
513             {
514                 CSE_send_data("OK", 0x06 , 0x05 , sizeof("OK"));
515                 HAL_Delay(10); // delay de 10ms para que tenga tiempo para poder recibir paquetes
516                 Enviar_Comando_AT(payload_rx,size_rx);
517                 ESTADO=ESTADO0;
518             }
519             else
520             {
521                 CSE_send_data("ER", 0x06 , 0x05 , sizeof("ER"));
522                 HAL_Delay(10); // delay de 10ms para que tenga tiempo para poder recibir paquetes
523                 ESTADO=ESTADO0;
524             }
525             HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, 0); // configuro el RS485 en modo recepcion
526             HAL_Delay(10); // delay de 10ms para que tenga tiempo para poder recibir paquetes
527         break;
528     }
```

Se habilita el RS485 para transmitir un mensaje de “OK” como payload en caso de que los **crc_obt** y **crc_aux** coincidan, o un mensaje de “ER” en caso contrario.
Para transmitir con el módulo RS485 debe colocarse en valor 1 el pin PC8 (o en valor 0 para recepción)

Cuando ambos CRC coinciden se envía al transmisor LoRa, el payload recibido del Sensor Side mediante la utilización de comandos AT a través de la UART.

Funciones utilizadas en LoRa Side

Función CSE_receive_data:

Para recibir datos de la trama enviada por el Sensor Side, se utiliza la función **CSE_receive_data**, la cual nos ayudará también a verificar que el parsing de los bytes de la trama se ha efectuado de manera correcta.

```
298 //### Funcion que recibe por UART ###
299 uint8_t CSE_receive_data(){
300
301     HAL_UART_Receive_IT(&huart6, &data_rx, sizeof(data_rx));
302
303
304     while(RX_PENDING);
305
306     RX_PENDING = 1;
307
308     return data_rx;
309 }
```

En esta función, se llama a la API de **HAL_UART_Receive_IT** que se mantendrá recibiendo información de la UART en tamaño de enteros sin signo de 8 bits. Cuando se haya completado la recepción de un byte en la variable **data_rx**, se modificará el estado del flag **RX_PENDING** por medio del uso de una Callback y la función estará habilitada para retornar un valor.

```
359 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart6)
360 {
361     // Drivers > STM32F..HAL_Driver > Src > stm32f4xx_hal_uart.c
362
363     RX_PENDING = 0;
364 }
365
366 /* USER CODE END 0 */
```

La función **CSE_receive_data** está implementada a lo largo de la máquina de estados y los bytes retornados se guardan en el vector **info**.

Se comparan los elementos de este vector con los que debería tener la trama en caso de una recepción correcta y si la información coincide se avanza al próximo estado, caso contrario se vuelve al estado inicial.

Función Escritura_CRC:

Esta función toma como parámetros los bytes de origen, destino, tamaño y payload de la trama y tiene por objetivo utilizar la API **HAL_CRC_Calculate** para determinar el valor de CRC, esta API necesita que se le pase como parámetro un vector de variables de 32 bits.

En el caso que la cantidad de bytes totales de la trama no sea múltiplo de 32 bits, se debe rellenar la trama con una cantidad de bytes que lo hagan múltiplo. Se eligió, por convención, que el valor de dichos bytes sea “0”.

La función determina, de acuerdo al tamaño del payload ,si es múltiplo de 4 enteros sin signo de 8 bits (32 bits) y determina la cantidad de relleno a agregar.

Luego declara un vector denominado **Data_Input** del tamaño de la trama, adiciona el relleno y se le asigna a cada byte de dicho vector los bytes de la trama, inicializando los bytes correspondientes al CRC con el valor 0x00. Finalmente retornando el valor del CRC calculado con la API.

```
226 uint32_t Escritura_CRC(uint8_t origen, uint8_t destino, uint8_t size, uint8_t* payload_rx){
227
228     uint8_t i=0;
229     uint32_t CRC_;
230     uint8_t relleno=0;
231     uint8_t RESTOS=(11+size)%4;
232
233     switch (RESTOS)
234         {
235             case 0:
236                 relleno=0;
237               break;
238             case 1:
239                 relleno=3;
240               break;
241             case 2:
242                 relleno=2;
243               break;
244             case 3:
245                 relleno=1;
246               break;
247         }
248
249     uint8_t Data_Input [11+size+relleno];
250     Data_Input[0] = 0xAA;
251     Data_Input[1] = 0x55;
252     Data_Input[2] = origen;
253     Data_Input[3] = destino;
254     Data_Input[4] = size;
255
256     for (i=0 ; i<size ; i++)
257     {
258         Data_Input[5+i] = payload_rx[i];
259     }
260
261     Data_Input[5+size] = 0x00;
262     Data_Input[6+size] = 0x00;
263     Data_Input[7+size] = 0x00;
264     Data_Input[8+size] = 0x00;
265     Data_Input[9+size] = 0x55;
266     Data_Input[10+size] = 0xAA;
267
268     for (i=11+size ; i<=11+size+relleno ; i++)
269         {
270             Data_Input[i] = 0x00;
271         }
272
273     CRC_ = HAL_CRC_Calculate(&hcrc, (uint32_t*)Data_Input, sizeof(Data_Input) / sizeof(uint32_t));
274
275     return(CRC_);
276 }
```

Función CSE_send_data:

Esta función toma como parámetros el payload , su tamaño , dirección de origen y destino, y construye la trama para transmisión al Sensor Side de los mensajes de “OK” o “ER” en caso de que la verificación de la integridad de los bytes recibidos de la trama por medio de CRC, haya sido correcta o no. La transmisión de los datos se realiza por la UART 6 (la UART 2 está habilitada para verificación que quiera hacer el usuario por consola)

```
uint8_t CSE_send_data(uint8_t* vec, uint8_t origen , uint8_t destino , uint8_t size){
    uint8_t PACKAGE_BYTES = 11; // Cantidad de bytes adicionales al payload a transmitir.
    uint8_t i=0;
    uint8_t crc [4]= {0x00,0x00,0x00,0x00};
    uint8_t data[size + 11];
    uint32_t CRC_AUX=0;
    for(i=0; i < size + PACKAGE_BYTES; i++)
    {
      if(i==0)
          data[i] = 0xAA;
      if(i==1)
          data[i] = 0x55;
      if(i==2)
          data[i] = origen;
      if(i==3)
          data[i] = destino;
      if(i==4)
          data[i] = size; // 16
      if(i==(size+5))
          data[i] = crc[3];
      if(i==(size+6))
          data[i] = crc[2];
      if(i==(size+7))
          data[i] = crc[1];
      if(i==(size+8))
          data[i] = crc[0];
      if(i ==(size+9))
          data[i] = 0x55;
      if(i == (size+10)){
          data[i] = 0xAA;
      }

      if( (i>=5) && (i<=(5+size)))
          data[i] = vec[i-5];
    }

     CRC_AUX=Escritura_CRC(data[2], data[3], data[4], vec);
     data[size+5]=(CRC_AUX&0xFF000000)>>24;
     data[size+6]=(CRC_AUX&0x00FF0000)>>16;
     data[size+7]=(CRC_AUX&0x0000FF00)>>8;
     data[size+8]=CRC_AUX&0x000000FF;


    HAL_UART_Transmit_DMA(&huart2, data, sizeof(data));
    HAL_UART_Transmit_DMA(&huart6, data, sizeof(data));

    return 0;
}
```

Aquí también implementamos la función **Escritura_CRC** para calcular el CRC por medio del uso de API y también separamos el valor calculado con dicha función en 4 bytes para enviar dentro de la trama.

Enviar_Comando_AT:

Por último, esta función acondiciona la información recibida y con el CRC de la misma verificada, para transmitir por medio de la UART 1, a un módulo emisor LoRa mediante el comando **"AT+SEND+ <payload> "**(donde <payload> es el mensaje a transmitir):

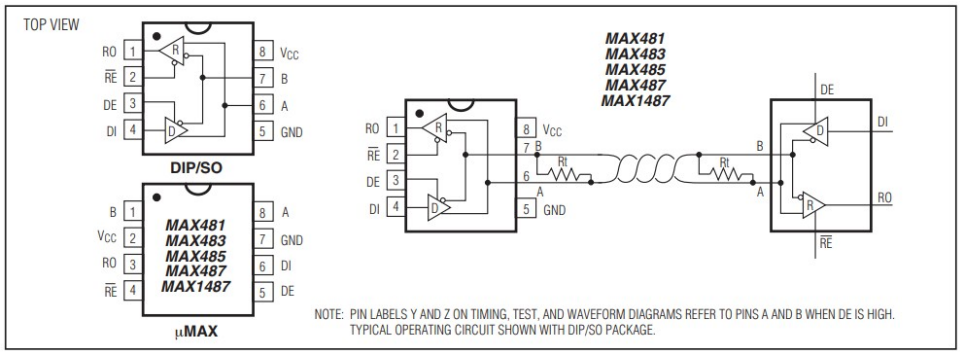
```
315 void Enviar_Comando_AT(uint8_t* payload, uint8_t size_rx){
316
317     uint8_t k=0;
318       uint8_t comando_AT[9] =  "AT+SEND+";
319       uint8_t size_AT = sizeof(comando_AT);
320       uint8_t mensaje_AT[size_AT+size_rx-1]; // el -1 es por el caracter nulo del vector char comando at
321     for(k=0; k < size_AT-1; k++){
322         mensaje_AT[k] =  comando_AT[k];
323     }
324
325     for(k=0; k < size_rx ;k++){
326         mensaje_AT[k+size_AT-1] =  payload[k];
327     }
328     HAL_UART_Transmit_DMA(&huart2, mensaje_AT, sizeof(mensaje_AT));
329     HAL_UART_Transmit_DMA(&huart1, mensaje_AT, sizeof(mensaje_AT));
330 }
```

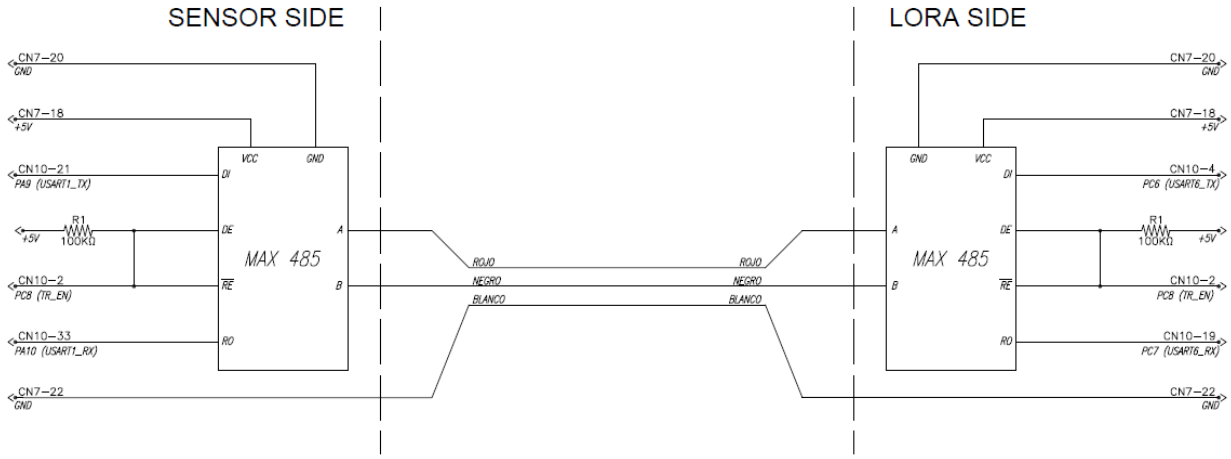
La UART 2 aquí también está habilitada para que el usuario pueda ver por consola el mensaje transmitido y verificar los datos enviados al transmisor LoRa.

Luego el módulo transmisor LoRa deberá encargarse de la transmisión de los datos a un gateway.

6. Conexionado

Para la comunicación entre ambos lados (Sensor y Lora) se utiliza comunicación serie cableada RS485. Para esto se utilizan dos módulos MAX-485 conectados según el siguiente diagrama:





Para el funcionamiento de estos módulos se alimentan con 5 volts y GND de las placas (debe ser la misma referencia para ambas placas).
El driver Input (DI) se conecta a la trasmisión de la UART.
El Receiver Output (RO) se conecta a la recepción de la UART.
El sensor side utiliza la UART 1 de la placa mientras que el lora side utiliza la UART6.
Ambas UART's se configuran de la siguiente forma:

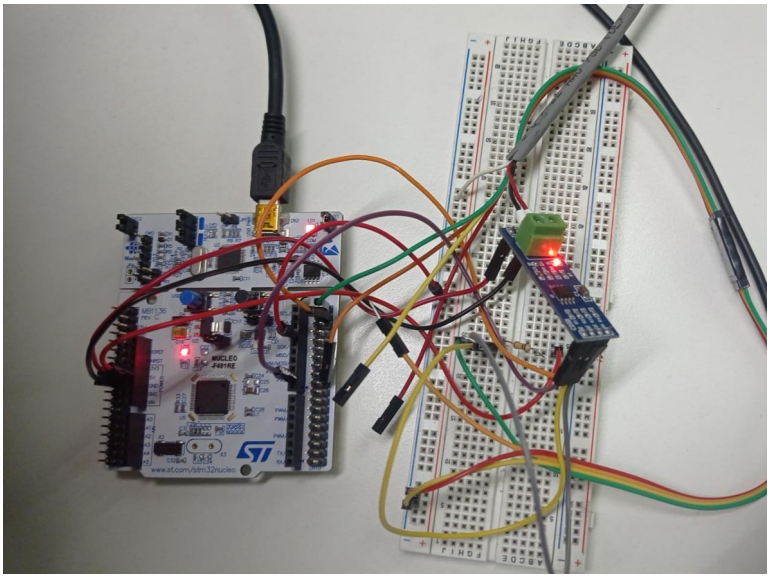
Baud Rate	9600
Data Bit	8
Paridad	SI
Bit Stop	1

Los habilitadores DE y RE cambian sus valores para recibir o transmitir datos según la siguiente tabla:

	EMISOR	RECEPTOR
DE	VCC	GND
~RE	VCC	GND

7. Resultados

Se logró el funcionamiento esperado de ambos bloques, LoRa Side y Sensor Side. Se realizaron sucesivas pruebas llevando a cabo los ajustes necesarios cada vez. Luego de presentar fallas repetitivas en el pin de habilitación de modo transmisión/recepción del módulo MAX485(se daño el transistor de salida del microcontrolador), se concluyó que sería más seguro colocar un transistor en modo inversor a la salida y controlar la misma de forma indirecta.



8. Conclusiones

A modo de conclusión queremos destacar que los resultados obtenidos cumplieron con nuestros objetivos pudiendo establecer comunicación entre las dos placas de desarrollo. Esto nos permite escalar el proyecto a un sistema más complejo en el que se permita medir mayor cantidad de sensores y que sea capaz de tomar decisiones en función a los datos obtenidos, donde, ya que pudimos establecer un sistema capaz de detectar errores en los datos de transmisión por cálculo de CRC, nos da la garantía de que el sistema poseerá un sistema de comunicación robusto y confiable. Cabe resaltar además que, al ser un proyecto donde se requiere la mutua cooperación de dos grupos de trabajo, se comprobó la importancia de establecer una buena comunicación entre ambos y así poder definir criterios que son decisivos a la hora de llevar adelante el proyecto.