Universidad Autónoma de Yucatán

Facultad de matemática

Sistemas Embebidos

Informe de entrega Tarea 5

Haendel Gabriel López Martínez

Julián Adrián Viana Palomo

Licenciatura en ingeniería en computación

Fecha de entrega: 24 de octubre de 2025

Informe de entrega tarea no. 5

al. Julián Adrián Viana Palomo y al. Haendel Gabriel López Martínez

Curso Sistemas Embebidos
Facultad de Matemáticas
UNIVERSIDAD AUTÓNOMA DE YUCATÁN

Available 24 de octubre de 2025

En este documento se presenta el informe o guía de la teoría junto a la implementación usada para la elaboración de la tarea no. 5. Haciendo énfasis del porque usar la configuración usada, la estructura del código. Esto para la implementación de comunicación UART atreves de los registros USART entre dos STM32F103C8.

I. INTRODUCCIÓN

El ejercicio consiste en usar la comunicación USART para comunicar dos STM32F103C8T6. Cada STM32F103C8T6 debía tener conectado dos botones y dos Leds de modo que si la en la primera STM32F103C8T6 se presionaba un botón en la otra STM32F103C8T6 se debía prender el LED correspondiente, y viceversa, ambas STM32F103C8T6 deben estar programadas para ser Transmisor y receptor al mismo tiempo.

II. MÉTODOS

Desarrollo teórico configuración del Reloj

La gestión del reloj, controlada por el módulo *Reset and Clock Control (`RCC`)[referencia],* es el primer paso en la inicialización de cualquier microcontrolador, ya que define la velocidad de operación de todos los periféricos. Para el ejercicio se selecciono el oscilador interno de alta velocidad (*`HSI`*) como fuente del reloj principal del sistema (*`SYSCLK`*).

- El HSI es un oscilador RC que proporciona una señal estable de 8Mhz por defecto, lo que simplifica la implementación al no requerir in cristal externo.
- Esta sección se realiza configurando el registro de configuración del reloj (*`RCC_CFGR`*).

Para que el microcontrolador pueda interactuar con los pines `GPIO` y el módulo `USART`, es obligatorio habilitar el reloj de cada periférico.
Los relojes se habilitan en los registros correspondiente del `RCC`:

- Puerto A (`GPIOA`), Puerto B (`GPIOB`) y `USART1`: Habilitados en el registro `RCC_APB2ENR`.

Desarrollo teórico configuración de pines GPIO

La configuración de los pines de entrada/salida (`GPIO`) es fundamental para asignar las funciones de comunicación serial a los pines físicos del microcontrolador.

El primer paso es habilitar el reloj del Puerto A (*`GPIOA`*) a través del periférico Reset and Clock Control (*`RCC`*). Esta activación se realiza en el registro *`RCC_APB2ENR`* para proporcionar energía al módulo *`GPIO`*.

Para la comunicación USART, los pines GPIO deben configurarse para su Función Alternativa (**AF**), lo que permite que el periférico USART tome el control de las líneas.

- Pin de Transmisión (TX – PA9): Se configura como Salida de Función Alternativa (*Alternate function output*). Esto se hace ajustando los bits de MODE y CNF correspondientes al Pin 9 en el registro GPIOx_CRH. Se define como una salida push/pull de modo que asegura una señal de salida robusta y definida para la transmisión de datos que va a realizar el pin. a velocidad de 50 MHz se selecciona para garantizar la integridad de la señal a la alta velocidad de *baud rate* elegida (*115200 baudios*) y para cualquier potencial de comunicación futura más rápida.
- Pin de Recepción (RX – PA10): Se configura como Entrada Flotante (*Input floating*). Esta configuración permite que el pin reciba datos sin una pull-up o pull-down interna específica, como se requiere para el receptor USART.

La asignación de PA9 como TX y PA10 como RX se utiliza para interconectar los dos microcontroladores STM32F103C8T6 (TX a RX y RX a TX). El diagrama de conexión utilizado se detalla visualmente en la Figure 1 y el Diagrama del circuito Figure 2 del reporte.

Desarrollo teórico configuración USART

El periférico Universal Synchronous Asynchronous Receiver Transmitter (*USART*) es el encargado de gestionar la comunicación serial asíncrona entre los dos microcontroladores.
Para asegurar una comunicación exitosa, ambos dispositivos deben operar con los mismos parámetros de protocolo:

- Baud Rate (Velocidad de Transmisión)

Se seleccionó una velocidad de 115200 baudios. Este valor se determina mediante el reloj del periférico y un divisor (USARTDIV).

El valor del divisor se establece en el registro *USART_BRR* y se calcula con la fórmula:

$$Baud\ rate = \frac{Fclk}{16 \times USARTDIV}$$

Referencia 17.2.5 Fractional baud rate generation page 388 [UM0306, p. 388]

Para una frecuencia de reloj del periférico (*Fclk*) de 8 MHz, el valor de USARTDIV es aproximadamente 4.34027, resultando en un valor hexadecimal de 0x45 para el registro USART_BRR (Mantisa 4h, Fracción 5h).

- Longitud de Palabra y Paridad

La longitud de palabra (8 bits) y la paridad (sin paridad) se configuran en el Registro de Control 1 (*USART_CR1*). La longitud se define con el bit M. Sin embargo, para ejercicio no se especificó este parámetro por lo que se decidió no utilizarlo.
Para poder controlar la paridad de la palabra hay que habilitar esta función en el Registro de Control 1 (*USART_CR1*) en el bit 10 llamado Parity Control Enable (*PCE*). Dejandolo deshabilitado dejando como la configuración de paridad deshabilitado.

- Bits de Parada

Se configuró 1 bit de parada en el Registro de Control 2 (USART_CR2).

Una vez configurado el protocolo, se debe habilitar el transmisor (TX) y el receptor (RX) a través de los bits específicos dentro de los registros de control del USART para que el módulo inicie su operación.

Referencia tipo de transmisión 17.2.3 Transmitter Figure 146. Configurable stop bits page 384 [UM0306, p. 384].

III. IMPLEMENTACIÓN

El código está estructurado en varios archivos para organizar mejor las responsabilidades:

configuración del reloj, configuración del USART y lógica principal.

- Address.h Figure 3.1

Este archivo funciona como "diccionario" de direcciones de memoria. En lugar de memorizar y escribir direcciones numéricas largas como 0x40021000 en el código, se usaron nombres como RCC_CR o GPIOA_ODR. Este archivo de cabecera define todos los punteros a los registros del hardware (GPIO, RCC, USART) que ocupamos para la tarea.

Referencias direcciones Memory map Figure 2. Memory map page 26, Table 1. Register boundary addresses page 27-28 [UM0306, p. 26]

- CLK_config.c Figure 3.2

La función de este archivo es inicializar el sistema de reloj. Activamos el oscilador interno de alta velocidad (HSI), como la fuente de reloj principal del sistema (SYSCLK) y habilitamos el reloj para los periféricos a usar: el Puerto A (para USART), el Puerto B (para LEDs/botones) y el módulo USART1.

- USART_config.c Figure 3.3

Este archivo contiene toda la lógica para configurar y operar la comunicación serial. Incluye:

- USARTports()

  Configura los pines PA9 (TX) y PA10 (RX) en sus modos de hardware correctos (Salida de Función Alternativa y Entrada).
- USARTconfig() Figure 3.3

  Establece los parámetros del protocolo UART, como el baud rate (115200), 8 bits de datos, 1 bit de parada y sin paridad. También habilita el transmisor y el receptor.

- transmmitt() Figure 3.4

  La línea `while(!(USART1_SR & (1<<7)));` hace que el programa se detenga y espere.

  TXE (Transmit Data Register Empty) es una bandera que se pone en 1 cuando el registro de *buffer* de transmisión (USART1_DR) está libre para recibir un nuevo dato. El bucle espera hasta que TXE sea 1, asegurando que el microcontrolador no sobrescriba un dato que aún no ha comenzado a enviarse.

  Una vez que el *buffer* está vacío, la línea `USART1_DR = value;` copia el valor que se desea enviar al Registro de Datos (USART1_DR).

  La línea `while(!(USART1_SR & (1<<6)));` detiene nuevamente el programa para esperar la finalización completa de la transmisión. Está comprobando el Bit 6 (TC) en el Registro de Estado (USART1_SR).

  Referencia de uso 17.2.3 Transmitter page 383 [UM0306, p. 383]

- receiver () Figure 3.4

  La línea `if (USART1_SR & (1<<5))` comprueba el Bit 5 (RXNE) en el Registro de Estado (USART1_SR).

  RXNE (Read Data Register Not Empty) es una bandera que se pone en 1 cuando el *hardware* ha terminado de recibir un *byte* completo y lo ha guardado en el *buffer* de recepción (USART1_DR).

  Si RXNE es 1 (hay dato): La línea `return (uint8_t) USART1_DR;` lee el valor del Registro de Datos (USART1_DR).

  Si RXNE es 0 (no hay dato): La función ejecuta el `else` y devuelve el valor 0xFF. Este es un valor de "no dato" o "error/vacío" que se utiliza para informar al programa principal que no había nada disponible para leer en ese momento.

Referencia 17.2.4 Receiver page 385 [UM0306, p. 385]

- Main.c Figure 3.5

1. Llama a las funciones de CLK_config.c y USART_config.c para inicializar el sistema.
2. Llama a PINconfig(), una función local que configura los pines del Puerto B como entradas (botones) o salidas (LEDs).

   Esta parte definimos los pines de entrada como pines input pull up/down para poder tener sus lecturas a través del registro `GPIOx_IDR` y poner saber el estado del pin. Adicionalmente con el registro de escritura `GPIOx_ODR` establecemos en estado bajo (activando pull down) para poder conectar nuestros botones a VCC 3.3 y leer 1 en `GPIO_IDR`. Respectando nuestro diagrama Figure 1.

   Nuestros pines de salida están configurados como salidas push/pull 50 Mhz. Esto con el afán que que en futuras configuraciones del reloj podamos usarlo para frecuencia de alta velocidad.
3. Entra en un bucle infinito `(while(1))` que constantemente:

- Lee el estado de mis botones (`PB0` y `PB1`).
- Envía un código (`0x00, 0x01, 0x02 o 0x03`) por el `TX` basado en qué botón se presiona.
- Revisa si ha llegado un dato por el `RX`.
- Usa un switch para encender o apagar los LEDs (`PB12 y PB13`) según el código recibido.

Figure 1. Diagrama esquemático de la actividad.

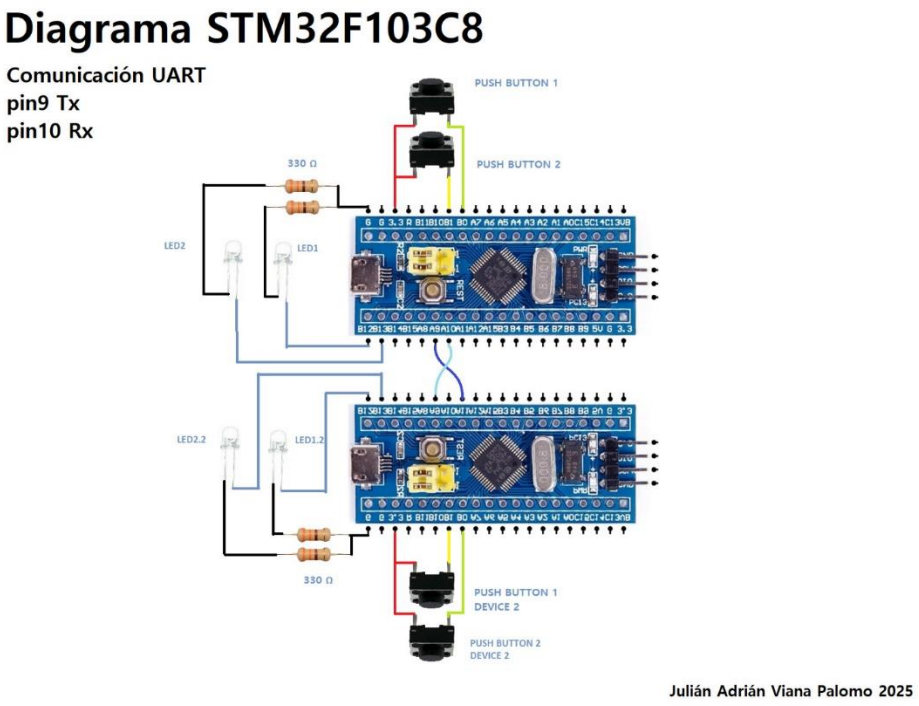

Figure 2. Diagrama esquemático de la actividad físico.

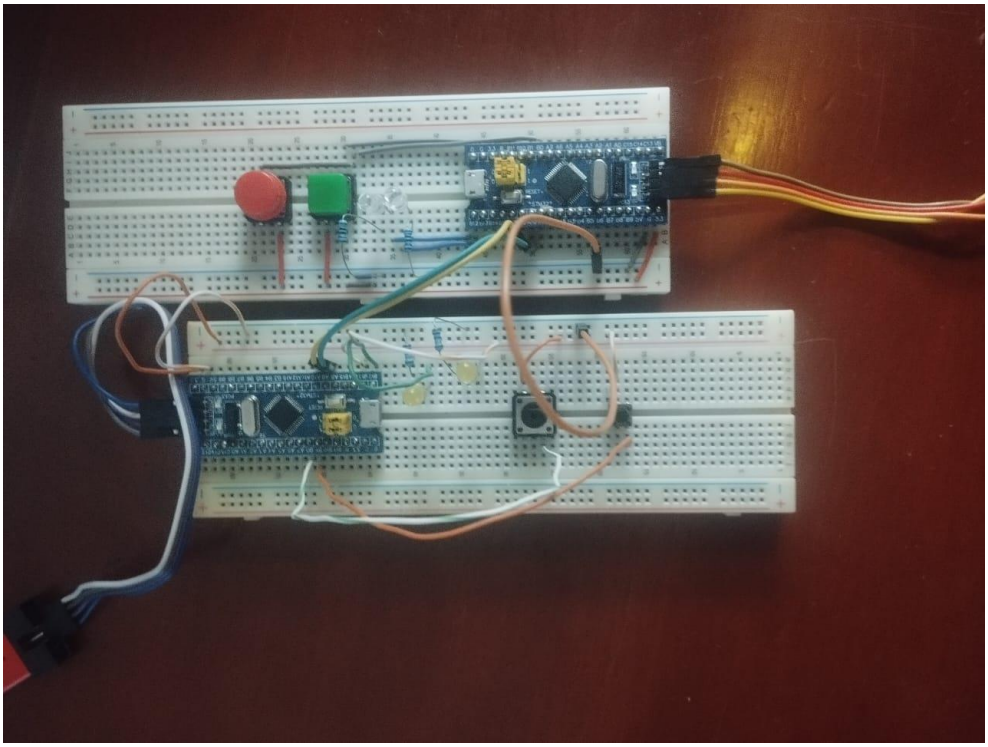

Figure 3.1 Código implementado address.h

address.h

```
//punteros a las memorias

//CLK
#define RCC_CR      (*(volatile uint32_t*) 0x40021000) //registro de encendido de osciladores
#define RCC_CFGR    (*(volatile uint32_t*) 0x40021004) //registro seleccion de  osciladores

//PERIFERICOS
#define RCC_APB2ENR    (*(volatile uint32_t*)  0x40021018)//registro para habilitar puertos/usart



//GPIOx registros

//puertoA
#define GPIOA_ODR    (*(volatile uint32_t*)0x4001080C)  // escritura de datos de datos (ODR)
#define GPIOA_IDR    (*(volatile uint32_t*)0x40010808)  // lectura de datos (IDR)
#define GPIOA_BSRR   (*(volatile uint32_t*)0x40010810)  // registro de set/reset rápido

#define GPIOA_CRL    (*(volatile uint32_t*)0x40010800)  // configuración de pines PA0..PA7
#define GPIOA_CRH    (*(volatile uint32_t*)0x40010804)  // configuración de pines PA8..PA15

//puerto B
#define GPIOB_ODR    (*(volatile uint32_t*)0x40010C0C)  // escritura de datos de datos (ODR)
#define GPIOB_IDR    (*(volatile uint32_t*)0x40010C08)  // lectura de datos (IDR)
#define GPIOB_BSRR   (*(volatile uint32_t*)0x40010C10)  // registro de set/reset rápido

#define GPIOB_CRL    (*(volatile uint32_t*)0x40010C00)  // configuración de pines PB0..PB7
#define GPIOB_CRH    (*(volatile uint32_t*)0x40010C04)  // configuración de pines PB8..PB15

//USART
#define USART1_SR     (*(volatile uint32_t*)  0x40013800) //registro para banderas de TXE and RXE
#define USART1_DR     (*(volatile uint32_t*)  0x40013804) //registro de transmision/recibimiento de datos
#define USART1_BRR    (*(volatile uint32_t*)  0x40013808) //registro para configurar el baud rate

#define USART1_CR1    (*(volatile uint32_t*)  0x4001380C) //registro de habilitacion y configuracion M, par/impar ..
#define USART1_CR2    (*(volatile uint32_t*)  0x40013810) //registro configuracion para bit stop

```

Figure 3.2 Código implementado CLK_config.c

CLK_config.c

```
//author: Julian Adrian Viana Palomo - Haendel Gabriel Lopez Martinez
#include "stm32f10x.h"                  // Device header
#include "address.h"                    //direcciones de memoria
#include "stdint.h"                     //C


void CLKconfig(void) {

    RCC_CR |= (1<<0);// HSI ON
    while (!(RCC_CR & (1<<1)));  // espera a que HSI esté listo

    // Selecciona HSI como SYSCLK
    RCC_CFGR &= ~(0x3); //limpia registro
    while ((RCC_CFGR & 0xC) != 0x0);  // espera SYSCLK = HSI


  //habilito perifericos
    RCC_APB2ENR |= (1 << 2);   // IOPAEN
    RCC_APB2ENR |= (1 << 3);   // IOPBEN
    RCC_APB2ENR |= (1 << 14);  // USART1EN
}

```

Figure 3.3 Código implementado USART_config.c

USART_config.c

```
1
2  //author: Julian Adrian Viana Palomo - Haendel Gabriel Lopez Martinez
3  #include "stm32f10x.h"                  // Device header
4  #include "address.h"                    //direcciones de memoria
5  #include "stdint.h"                     //C
6
7  //configuracion pines 9 & 10
8  void USARTports(void){
9
10   GPIOA_CRH &= ~(0xF << 4); //limpiamos bits para pin9 & pin10
11   GPIOA_CRH |= (0xB<<4); //pin9 -> output push/pull 50Mhz
12
13   GPIOA_CRH &= ~(0xF << 8);
14   GPIOA_CRH |= (0xB<<8); //pin10 -> Input Floating
15 }
16 //  USARTconfig -> configurar USART como UART
17 void USARTconfig(void){
18
19   USART1_CR1 &= ~(1<<13); //deshabilitamos USART para configurar
20
21   USART1_CR1 = 0; //limpiamos registro entero
22
23   USART1_BRR = (4<<4) | 5;
24
25   //con lo anterior aseguramos baud rate de 115200
26
27   USART1_CR1 |= (0<<12); //word leght -> 8bits
28   USART1_CR1 |= (0<<10); //PEN -> disabled -> decir que no queremos control de paridad
29                          //parity: none
30
31   USART1_CR2 = 0; //limpiamos registro entero
32   USART1_CR2 |= (0b00 << 12); //STOP BIT -> 1 -> 00
33
34   //habilitamos USART1
35   USART1_CR1 = (1<<13) | (1<<3) | (1<<2); // UE, TE, RE habilitados
36 }
```

Figure 3.4 Código implementado USART_config.c funciones transmmitt & receiver

USART_config.c

```
1
2  //author: Julian Adrian Viana Palomo - Haendel Gabriel Lopez Martinez
3  #include "stm32f10x.h"                  // Device header
4  #include "address.h"                    //direcciones de memoria
5  #include "stdint.h"                     //C
6
7  //configuracion pines 9 & 10
8  void USARTports(void){
9
10   GPIOA_CRH &= ~(0xF << 4); //limpiamos bits para pin9 & pin10
11   GPIOA_CRH |= (0xB<<4); //pin9 -> output push/pull 50Mhz
12
13   GPIOA_CRH &= ~(0xF << 8);
14   GPIOA_CRH |= (0xB<<8); //pin10 -> Input Floating
15 }
16 //  USARTconfig -> configurar USART como UART
17 void USARTconfig(void){
18
19   USART1_CR1 &= ~(1<<13); //deshabilitamos USART para configurar
20
21   USART1_CR1 = 0; //limpiamos registro entero
22
23   USART1_BRR = (4<<4) | 5;
24
25   //con lo anterior aseguramos baud rate de 115200
26
27   USART1_CR1 |= (0<<12); //word leght -> 8bits
28   USART1_CR1 |= (0<<10); //PEN -> disabled -> decir que no queremos control de paridad
29                          //parity: none
30
31   USART1_CR2 = 0; //limpiamos registro entero
32   USART1_CR2 |= (0b00 << 12); //STOP BIT -> 1 -> 00
33
34   //habilitamos USART1
35   USART1_CR1 = (1<<13) | (1<<3) | (1<<2); // UE, TE, RE habilitados
36 }
```

Figure 3.5 Código implementado main.c

USART_config.c

```c

//author: Julian Adrian Viana Palomo - Haendel Gabriel Lopez Martinez
#include "stm32f10x.h"                  // Device header
#include "address.h"                    //direcciones de memoria
#include "stdint.h"                     //C

//configuracion pines 9 & 10
void USARTports(void){

  GPIOA_CRH &= ~(0xF << 4); //limpiamos bits para pin9 & pin10
  GPIOA_CRH |= (0xB<<4); //pin9 -> output push/pull 50Mhz

  GPIOA_CRH &= ~(0xF << 8);
  GPIOA_CRH |= (0xB<<8); //pin10 -> Input Floating
}
//  USARTconfig -> configurar USART como UART
void USARTconfig(void){

  USART1_CR1 &= ~(1<<13); //deshabilitamos USART para configurar

  USART1_CR1 = 0; //limpiamos registro entero

  USART1_BRR = (4<<4) | 5;

  //con lo anterior aseguramos baud rate de 115200

  USART1_CR1 |= (0<<12); //word leght -> 8bits
  USART1_CR1 |= (0<<10); //PEN -> disabled -> decir que no queremos control de paridad
                         //parity: none

  USART1_CR2 = 0; //limpiamos registro entero
  USART1_CR2 |= (0b00 << 12); //STOP BIT -> 1 -> 00

  //habilitamos USART1
  USART1_CR1 = (1<<13) | (1<<3) | (1<<2); // UE, TE, RE habilitados
}
```

```c
int main (void){

  //configuramos el reloj HSI
  CLKconfig();

  USARTports();//configuramos los pines9 Tx y pin10 Rx

  //configuramos TX/RX
  USARTports();

  //cconfiguramos USART como protocolo UART
  USARTconfig();

  //configuramos LED/BOTONES
  PINconfig();

  while(1){//LOGICA PRINCIPAL

    //LECTURA DE1 BOTON (cuando transmite)
    int b0 = (GPIOB_IDR & (1 << 0)) ? 1 : 0; // Botón B0
    int b1 = (GPIOB_IDR & (1 << 1)) ? 1 : 0; // Botón B1

    uint8_t data_to_send = 0x00;

    // Detectar flanco de subida (presión)
    if (b0 && !last_state_b0 && !b1) {
        data_to_send = 0x01; // pinB0 presionadoh
    }
    else if (b1 && !last_state_b1 && !b0) {
        data_to_send = 0x02; // pinB1 presionado
    }
    else if (b0 && b1) {
        data_to_send = 0x03; //ambos presionados
    }
```

```
    //TRANSMISIÓN SOLO SI HAY CAMBIO
    if (data_to_send != 0x00) {
        transmmitt(data_to_send);
        for (volatile int i = 0; i < 50000; i++); // Pequeño retardo (debounce)
    }else{
        transmmitt(0x00);
        for (volatile int i = 0; i < 50000; i++);
    }

    //LECTURA DEL RX (cuando recibe)

    uint8_t receiverDATA = receiver();

    switch(receiverDATA) {
        case 0x00:
            GPIOB_ODR &= ~((1 << 12) | (1 << 13)); // Ambos OFF
            break;
        case 0x01:
            GPIOB_ODR |=  (1 << 12);  // LED B12 ON
            GPIOB_ODR &= ~(1 << 13);  // LED B13 OFF
            break;
        case 0x02:
            GPIOB_ODR &= ~(1 << 12);  // LED B12 OFF
            GPIOB_ODR |=  (1 << 13);  // LED B13 ON
            break;
        case 0x03:
            GPIOB_ODR |=  (1 << 12);  // LED B12 ON
            GPIOB_ODR |=  (1 << 13);  // LED B13 ON
            break;
        default:
            break;
    }
  }
}
```

IV. RESULTADOS

Link de evidencia de funcionamiento:

LINK DEL VIDEO DE FUNCIONAMIENTO

Link de códigos para observaciones:

https://github.com/Vianita17/SistemaEmbebidos-STM32/tree/464484baae0599c4ed59152054a2eebceb0f2678/15USART_UART

V. CONCLUSIONES

Como conclusión toca exponer un de las mayores dificultades del ejercicio. Si bien una vez leída la documentación, leer acerca de los registros nos da una gran información importante para poder emplear el uso de configuración para la implementación. La mayor dificultad final era identificar cual era el problema cuando desconectamos una de las líneas empatadas de Pin 9 a Pin 10. No sabíamos porque dejaba de funcionar la lógica de código. Y después de un rato logramos entender que habíamos implementado la lógica de la espera de bandera para recibir una palabra a través de un bucle while. Eso nos generaba que siempre cuando entraba en ese bucle si o si tenía que recibir la información. Quedándose atrapado en esa instrucción. La solución a este problema fue simple, en vez de un usar un bucle while, la condición de espera de la bandera de RXE. La implementamos en if de manera que una vez reciba información retorne esa información a nuestro main para su uso, dado el caso que no reciba información. Retorne un valor x sin función alguna.

REFERENCIAS

- ***STMicroelectronics.*** *(2007, junio). UM0306: Reference manual: STM32F101xx and STM32F103xX advanced ARM-based 32-bit MCUs (Rev 1). [Manual de referencia]. Recuperado de https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://www.bdtic.com/download/ST/UM0306.pdf&ved=2ahUKEwiVyp2dxL6QAxWfnWoFHYnEPIQQFnoECBgQAQ&usg=AOvVaw2YaSnIz91557ipcas0rExI*