UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

MICROCONTROLADORES

REPORTE
PRÁCTICA 3
UART

SONIA PAOLA AGUILLÓN OLAMENDI
261520

PRÁCTICA UART

1. OBJETIVO

El alumno conocerá y aprenderá a implementar el código de programación para la configuración y uso del UART utilizando el microcontrolador Tiva TM4C123GH6PM.

2. MARCO TEÓRICO

UART

UART significa **receptor/transmisor asíncrono universal** y define un protocolo, o conjunto de reglas, para intercambiar datos en serie entre dos dispositivos, este solo usa dos cables entre el transmisor y el receptor para transmitir y recibir en ambas direcciones. Ambos extremos también tienen una conexión a tierra.
La comunicación en UART puede ser:

- **Simple**: Donde los datos se envían en una sola dirección
- **Semidúplex:** Donde cada lado envía datos, pero solo uno a la vez
- **Dúplex completo:** Donde ambos lados pueden transmitir simultáneamente.

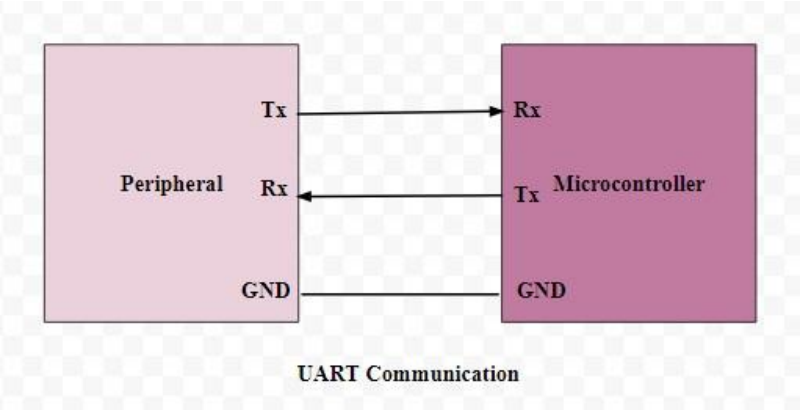


Fig1. *Comunicación UART*

Una de las grandes ventajas del UART es que es asíncrono: el transmisor y el receptor no comparten una señal de reloj común. Aunque esto puede simplificar el protocolo, pone ciertos requisitos al transmisor y al receptor. Dado que no comparten un reloj, ambos terminales deben transmitir a la misma velocidad preestablecida para que tengan el mismo tono de bits. Además de tener la misma velocidad en baudios, ambos lados de una conexión UART también deben utilizar los mismos parámetros y estructura de trama.

Bits de inicio y de parada

Ya que el UART es asíncrono, el transmisor necesita indicar que los bits de datos están llegando. Esto se logra utilizando el bit de inicio. El bit de inicio es una transición del estado alto de reposo a un estado bajo, y seguido inmediatamente por bits de datos de usuario.

Después de que se terminen los bits de datos, el bit de parada indica el fin de datos de usuario. El bit de parada es una transición de regreso al estado alto o de reposo o permanece en el estado alto por un tiempo de bit adicional.

3. ANTEDECENTES

Estudiando la datasheet se extrajeron los puntos más importantes para lograr llevar a cabo una configuración correcta del UART del microcontrolador Tiva TM4C123GH6PM.

Características UART

La Tiva TM4C123GH6PM cuenta con 8 módulos UART, que van desde el UART0 hasta el UART7. Cada uno de ellos cuenta con un FIFO receptor de 16x12, el cual es el Rx y un FIFO de transmisión de 16x8 el cual es el Tx. El UART tiene una velocidad programable, puede ser configurada la trama a 5, 6, 7 y 8 bits por carácter, este cuenta con todos los bits de paridad N, E, O, M y S, además de los bits de paro 1 o 2.

Nota: Solo el UART1 cuenta con control de flujo por hardware

Todos los UART tienen soporte para el estándar para el IrDA, cuentan con modo Loopback para conectar internamente la línea de transmisión de un UART a su línea de recepción para poder realizar diagnósticos o depuración. Tiene un soporte para el estándar RS-485 y para el LIN.

Configuración UART

El microcontrolador cuenta con puertos de entrada y salida, los cuales son denominados como módulos GPIO, los cuales controlan un grupo de 5 a 8 PINES.

Cuando se emplea cualquier módulo de entrada/salida que no sea GPIO y se quiere que se tenga acceso a un pin de entrada/salida, se necesita configurar necesariamente el módulo GPIO que controla ese PIN.

Por ejemplo:
Si se quiere hacer la conexión en los pines PD7 y PD6, se necesita configurar el módulo GPIO del puerto D.

Función alternativa: La configuración de un pin de un puerto para que sea controlado por otro módulo de entrada/salida.

Cada módulo UART cuenta con 31 registros de entrada y salida, de los cuales solo se utilizan 6

4	Configuración de la trama
1	Velocidad de comunicación
1	Habilitación

Tabla1. *Registros de entrada y salida UART*

UARTDR	Mediante el cual el CPU envía datos al UART o lee los datos recibidos
UARTFR	Permite conocer el estado de la comunicación
FIFO TRANSMICIÓN (Rx)	
FIFO RECEPCIÓN (Tx)	

Tabla2. *Módulos del UART*

Para llevar a cabo la configuración, se requiere realizar los siguientes 6 pasos.

1. Activar la señal del reloj
    - Módulo UART. (El que se emplea)
    - Módulos GPIO.
2. Desactivar el módulo UART. (No la señal de reloj)
3. Configurar los pines GPIO para Tx y Rx.
4. Configurar UART velocidad de comunicación. (Velocidad de transmisión y recepción)
5. Configurar trama y buffers.
6. Habilitar UART.

4. METODOLOGÍA

Llevar a cabo la práctica utilizando el UART7( );

Para comenzar a llevar a cabo la práctica, se inició configurando el UART7.

Inicialmente se reviso en la datasheet los pines que pueden ser conectados con el UART7.

14.2 SIGNAL DESCRIPTION PAG.894

U7Rx	9	PE0 (1)	I	TTL	UART module 7 receive.
U7Tx	8	PE1 (1)	O	TTL	UART module 7 transmit.

Fig2. *Pines UART 7*

Podemos observar como el UART7 sólo puede conectarse a los **pines PE0** y **PE1**, los cuales pertenecen al **puerto E**.

Línea transmisión (Tx)	PE1
Línea recepción (Rx)	PE0

Tabla3. *Pines Tx y Rx – Puerto E*

Es importante tener en cuenta, que además del UART7, es necesario activar la señal del reloj del módulo GPIOE, el cual es conocido como puerto E.

*NOTA TTL indica: La configuración eléctrica del PIN

- Pin digital
- Sin resistencia de pull-up
- Sin resistencia de pull-down
- No es función analógica

Posteriormente se llevo a cabo la activación del reloj UART7, el cual es llamado RCGCUART, esto se configura con los siguientes registros

RCGC1	UART0 a UART2
RCGCUART	Para todos los módulos UART

Tabla4. *Activación reloj UART*

NOTA: Si se emplea RGCC1 para algún periférico (no necesariamente un UART) entonces:

Configurar: UART0 UART1 UART2	Con RCGC1
Configurar: UART3 a UART7	Con RCGCUART

Tabla5. *Configuración RCGC1 y RCGCUART*

VALOR BIT	**ESTADO**
0	Reloj del módulo UART desactivado No se tiene acceso a sus registros de E/S
1	Reloj del módulo E/S activado

Tabla6. *Valores para estado reloj*

Continuando con la configuración, el siguiente paso fue activar el UART 7. (Línea 5 del código)

Cuando se habilita la señal de reloj de un módulo, no se genera de inmediato. Para cada tipo de modulo entrada/salida, hay un registro de solo lectura en donde cada bit indica si un módulo tiene su señal de reloj activa, a continuación, se muestran los valores para la configuración del estado E/S del módulo

VALOR BIT	**ESTADO DE MÓDULO E/S**
0	No está activa aún la señal de reloj del módulo E/S
1	Ya está activa la señal de reloj del módulo y se podrá configurar y utilizar

Tabla7. *Valor de bit para estado de módulo E/S*

Posteriormente se activó el reloj de los módulos GPIO. Estos módulos GPIO también tienen su registro de activación de reloj de nombre RCGCGPIO. Para ello, se tomo en cuenta el siguiente registro de la datasheet.

4	R4	RW	0	GPIO Port E Run Mode Clock Gating Control

Value	Description
0	GPIO Port E is disabled.
1	Enable and provide a clock to GPIO Port E in Run mode.

Fig3. *Registro Puerto E módulo GPIO*

Me corresponde el **puerto E**, por ello debo poner en 1 el puerto E, el cual es el **bit 4**

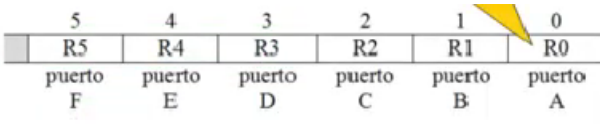


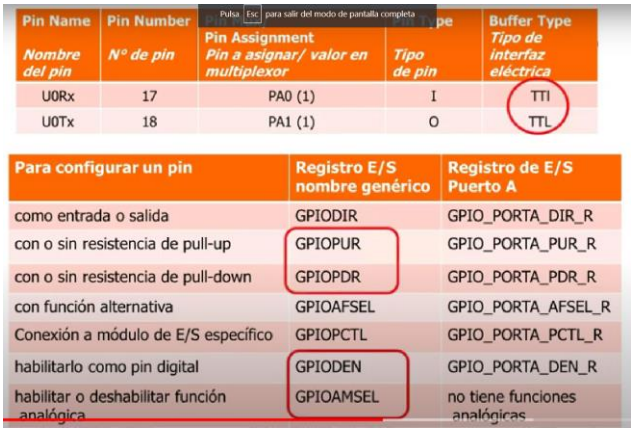
Fig4. *Configuración puerto E*

El siguiente paso realizado fue configurar los pines en modo TLL como se indicó anteriormente. Los registros de entrada salida que permiten configurar los pines son GPIOAFSEL y GPIOPCTL.

GPIOAFSEL GPIOPCTL	Permiten configurar los pines

Tabla8. *AFSEL y PCTL*

Para esta configuración, se utilizo un video de YouTube como recurso extra, del cual se extrajo la siguiente imagen que muestra un resumen de lo que es esta configuración.



Pin Name *Nombre del pin*	Pin Number *N° de pin*	Pin Assignment *Pin a asignar/ valor en multiplexor*	Tipo *Tipo de pin*	Buffer Type *Tipo de interfaz eléctrica*
U0Rx	17	PA0 (1)	I	TTl
U0Tx	18	PA1 (1)	O	TTL

Para configurar un pin	Registro E/S nombre genérico	Registro de E/S Puerto A
como entrada o salida	GPIODIR	GPIO_PORTA_DIR_R
con o sin resistencia de pull-up	GPIOPUR	GPIO_PORTA_PUR_R
con o sin resistencia de pull-down	GPIOPDR	GPIO_PORTA_PDR_R
con función alternativa	GPIOAFSEL	GPIO_PORTA_AFSEL_R
Conexión a módulo de E/S específico	GPIOPCTL	GPIO_PORTA_PCTL_R
habilitarlo como pin digital	GPIODEN	GPIO_PORTA_DEN_R
habilitar o deshabilitar función analógica	GPIOAMSEL	no tiene funciones analógicas

Fig5. *Resumen – Configuración pines*

Con esta información, se comenzó la configuración.

AFSEL

|Register 10: GPIO Alternate Function Select (GPIOAFSEL) PAG. 672

VALORES REQUERIDOS PARA SU CONFIGURACIÓN:

Como podemos notar anteriormente mis pines son el 0 y el 1 del puerto E, por ello se ponen en 1 esos dos pines. (LINEA 8 CÓDIGO)

U7Rx	9	PE0 (1)	I	TTL	UART module 7 receive.
U7Tx	8	PE1 (1)	O	TTL	UART module 7 transmit.

Fig6. *GPIO Alternate Function Select (GPIOAFSEL)*

```
7    //(GPIOAFSEL) pag.671 Enable alternate function        *Se habilitan funcion alternativa del GPIO para configurar los pines*
8    GPIOE->AFSEL |= (1<<1) | (1<<0);   //*Corresponde puerto E, pines 0 y 1
```

Fig7. *Configuración AFSEL en código C.*

DEN

|Register 18: GPIO Digital Enable (GPIODEN) PAG. 682

VALORES REQUERIDOS PARA SU CONFIGURACIÓN:

Se desea que los pines PE0 y PE1 sean TTL, por ello es necesario que sean pines digitales, siendo así se pone 1 en el pin 0 y 1 en DEN.

Recapitulando un poco:

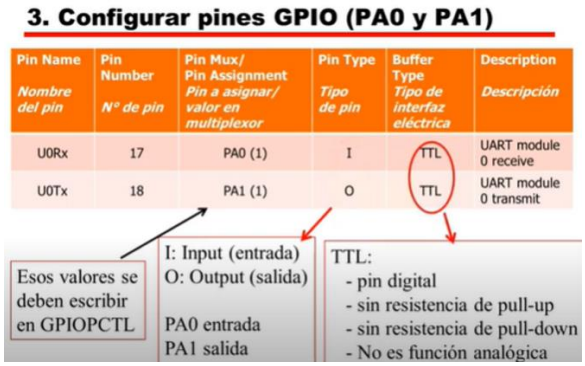


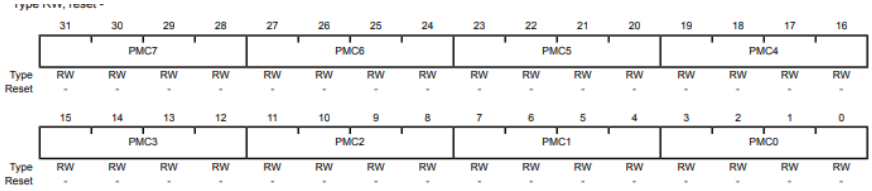
Fig8. *Configuración pines GPIO*

Por ello, para la configuración de GPIOPCTL se debe escribir el valor de los pines que nos corresponden. En este caso corresponden el pin PE0 y pin PE1, donde:

PE0	Entrada
PE1	Salida

Tabla9. *Pines entrada y salida UART7*

GPIOPCTL es utilizada junto con el registro del AFSEL y se configura para la conexión a módulo de entrada o salida especifico, es decir, se utiliza para configurar cual va como entrada o salida.



Bit/Field	Name	Type	Reset	Description
31:28	PMC7	RW	-	Port Mux Control 7 This field controls the configuration for GPIO pin 7.
27:24	PMC6	RW	-	Port Mux Control 6 This field controls the configuration for GPIO pin 6.
23:20	PMC5	RW	-	Port Mux Control 5 This field controls the configuration for GPIO pin 5.
19:16	PMC4	RW	-	Port Mux Control 4 This field controls the configuration for GPIO pin 4.
15:12	PMC3	RW	-	Port Mux Control 3 This field controls the configuration for GPIO pin 3.
11:8	PMC2	RW	-	Port Mux Control 2 This field controls the configuration for GPIO pin 2.
7:4	PMC1	RW	-	Port Mux Control 1 This field controls the configuration for GPIO pin 1.
3:0	PMC0	RW	-	Port Mux Control 0 This field controls the configuration for GPIO pin 0.

Fig9. *Configuración GPIOPCTL*

Para la configuración del GPIOPCTL, el pin que utilizamos es el pin PE0 y PE1, es decir, pin 0 y 1, por ello se utiliza el Port Mux 0 y 1.

En visual se pone la asignación que corresponde, es decir, poner en 1 el número de bit que corresponde (Poner el 1 en el 0 y 4)

El siguiente paso es configurar GPIODIR, el cual nos indica si es entrada o salida, por ello, los valores serán de la siguiente manera:

PE0	Entrada	0
PE1	Salida	1

Fig10. *Configuración GPIODIR*

Para poder continuar con la configuración es necesario desactivar el UART hasta terminar su configuración. El bit que lo activa es **UARTCTL**

Fig11. *Configuración UARTCTL*

- El bit0 UARTEN es el habilitador, por eso se debe poner en 0
- El bit9 es para RXE
- El bit8 para TXE, recibir y transmitir, también se ponen en 0 para desactivar

```
UART7->CTL = (0<<9) | (0<<8) | (0<<0);
```

Fig12. *Configuración UARTCTL -Código C.*

Posteriormente, se continúa configurando la velocidad de comunicación. Esta se obtiene a partir de la señal de reloj que recibe el UART, en este caso es el reloj de sistema es de 16MHz, el cual se divide entre dos factores que es el BRD y ClkDiv

La velocidad se obtiene con la siguiente formula:

$$Velocidad = \frac{Frec.Reloj}{ClkDiv * BRD}$$

Fig13. *Formula – cálculo velocidad*

ClkDiv

ClkDiv	Puede tener los valores 8 o 16	Se configura como un bit llamado HSE		
		Si HSE:	HSE=1	Divide entre 8
			HSE=0	Divide entre 16

Tabla10. *Configuración ClkDiv*

BRD se obtiene:

$$BRD = \frac{\text{Frec. Reloj}}{\text{ClkDiv*Velocidad}}$$

Fig14. *Formula – Cálculo BRD*

El UART cuenta con 2 registros para almacenar el valor de BRD, es decir, utiliza 1 para la parte entera y otro para la parte decimal.

UARTIBRD	Almacena parte entera	104
UARTFBRD	Almacena la parte fraccionaria, pero en base 2 y de 16bits	Redondear

Tabla11. *Configuración UARTIBRD y UARTFBRD*

Calculo ***UARTFBRD*** (para el redondeo):
Se multiplica el decimal por *64 y se redondea al entero más cercano

Por ejemplo:

$$Velocidad = \frac{Frec.Reloj}{ClkDiv * BRD}$$

$$BRD = \frac{\text{Frec. Reloj}}{\text{ClkDiv*Velocidad}} = \frac{16 * 10^6}{16*9600} = 104.16667$$

UARTIBRD = 104

10.67

UARTFBRD = redondear (0.16667 *64) = 11

Fig15. *Ejemplo calculo Velocidad*

Para escribirlo, es necesario saber que para el **UARTIBRD** solo se emplean los 16 primeros bits menos significativos**:**

Fig16. *Escritura UARTIBRD*

Por ejemplo, con BRD = 104 en binario

kmnñopqrstuvwxyz0000000001101000

Fig17. *104 BINARIO*

Otro ejemplo sería: **130** = FFFFFFFFFFFFFFFF0000000010000010

Para el **UARTFBRD**

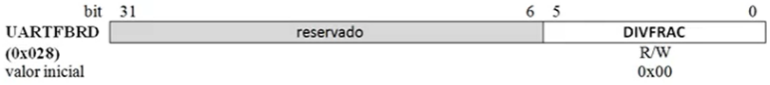


Fig18. *Escritura UARTFBRD*

Solo se emplean los 6 bits menos significativos
Por ejemplo: **14** = FFFFFFFFFFFFFFFFFFFFFFFFFF001110

Una vez sabiendo esto, se realizaron los cálculos correspondientes para la configuración que me toco.

Datos:

Fclk	50MHz	50,000,000
Baud-rate	57,600	

Tabla12. *Datos proporcionados por profesor*

$$BRD = \frac{50{,}000{,}000}{16*57600} = 54.2534$$

UARTIBRD	54
UARTFBRD	$(.2534 * 64) + 0.5 = 16.7176 = 17$

Tabla13. *Calculos correspondientes a datos*

Por lo tanto, daría un resultado de **54.17**

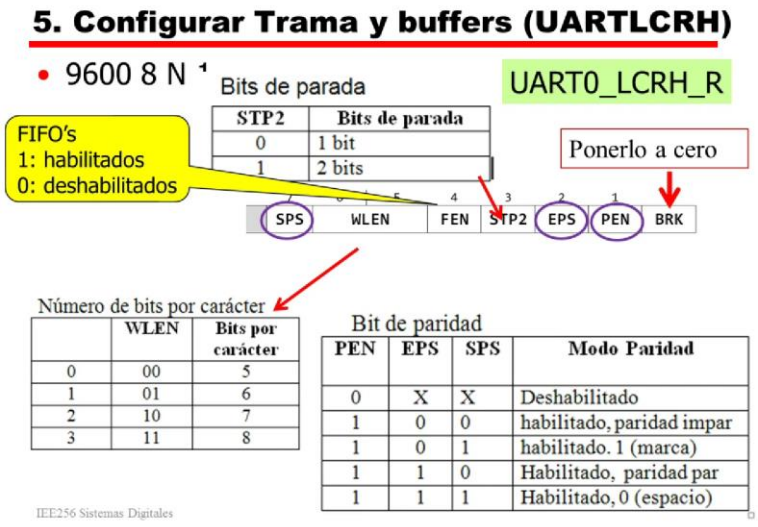
Se continuo con la configuración del trama y buffer. Esta configuración se hace con el registro de nombre genérico **UARTLCRH**. Siendo 8 los bits a configurar:

WLEN	Configura los bits por carácter
SPS **EPS** **PEN**	Configuran el bit de paridad

STP2	Configura el bit de parada
BRK	Se usa para enviar un break, que consiste en transmitir toda una trama en ceros, incluyendo el bit de parada.
FEN	Habilita si es que esta en 1 o deshabilita si es que esta en 0 los FIFOS de recepción y transmisión *Se recomienda siempre habilitarlos para tener más espacio de almacenamiento de datos

Tabla14. *Configuración UARTCLRH*

Para esta configuración, se utilizó un video de YouTube como recurso extra, del cual se extrajo la siguiente imagen que muestra un resumen de lo que es esta configuración.



STP2	Bits de parada
0	1 bit
1	2 bits

	WLEN	Bits por carácter
0	00	5
1	01	6
2	10	7
3	11	8

PEN	EPS	SPS	Modo Paridad
0	X	X	Deshabilitado
1	0	0	habilitado, paridad impar
1	0	1	habilitado. 1 (marca)
1	1	0	Habilitado, paridad par
1	1	1	Habilitado, 0 (espacio)

Fig19. *Configuración UARTLCRH*

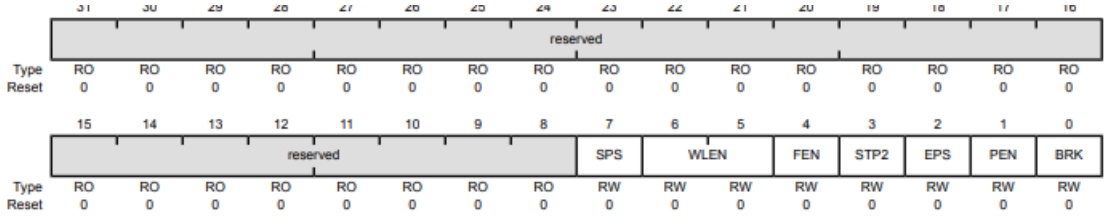


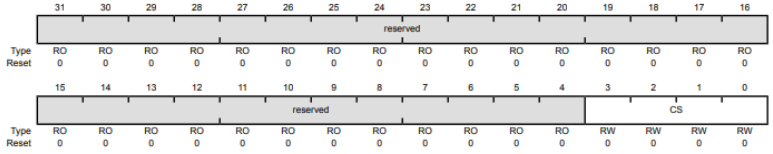
Fig20. *Bits UARTLCRH*

Los bits que nos van a interesar en **UARTLCRH** son:

WLEN	Longitud de palabra UART Los bits indican el número de bits de datos transmitidos o recibidos
FEN	Habilita o deshabilita los FIFOS
TODO LO DEMÁS	EN CERO

Tabla15. *Bits de interés UARTLCRH*

Posteriormente se modificó el registro UARTCC con ayuda de la datasheet, este controla la fuente de reloj en baudios para el módulo UART, en este caso, se le puso un 0 para indicar que se utilizara el reloj del sistema (basado en la fuente del reloj y el factor divisor)

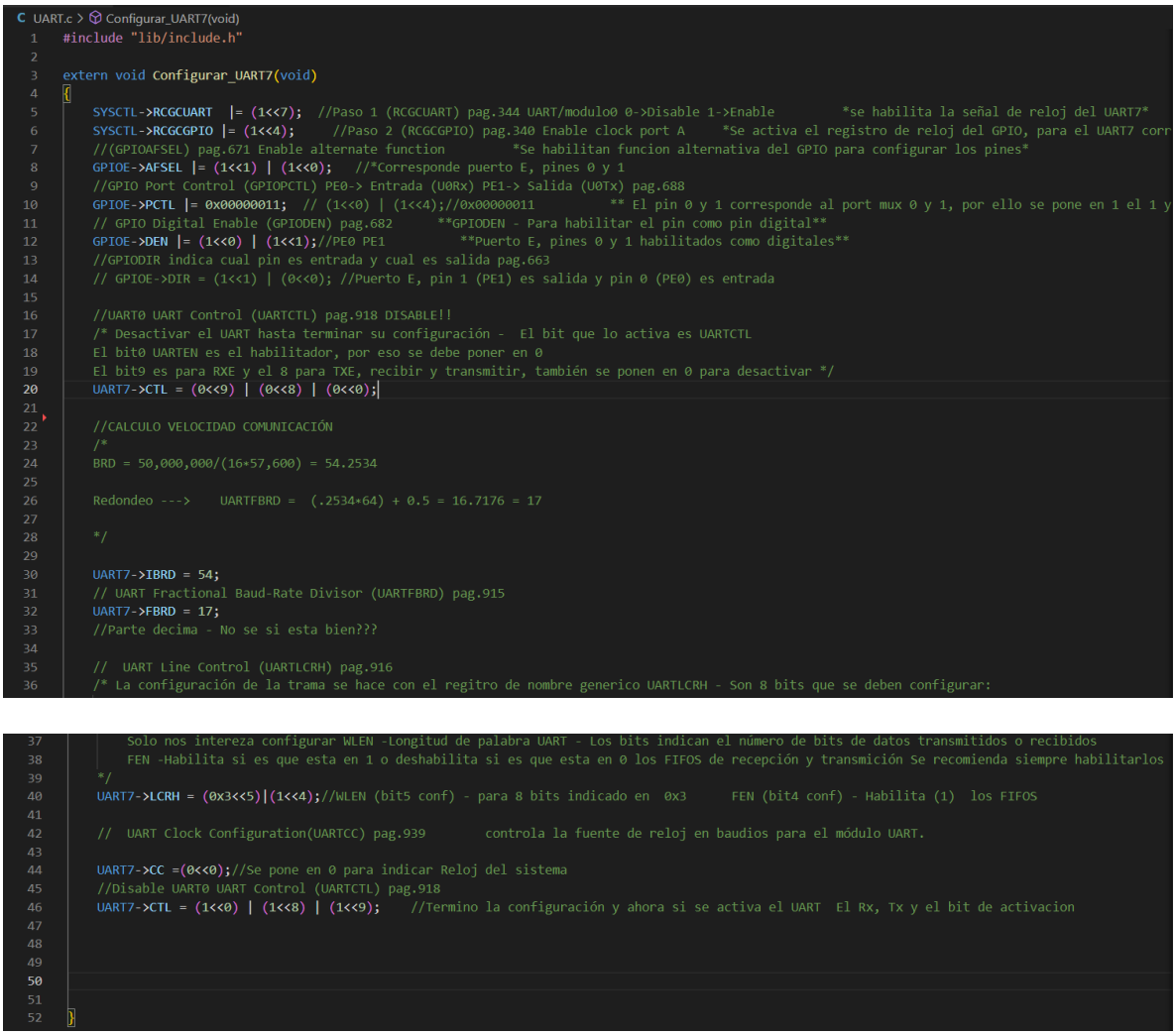


Bit/Field	Name	Type	Reset	Description
31:4	reserved	RO	0x0000.000	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
3:0	CS	RW	0	UART Baud Clock Source The following table specifies the source that generates for the UART baud clock:

Value	Description
0x0	System clock (based on clock source and divisor factor)
0x1-0x4	reserved
0x5	PIOSC
0x5-0xF	Reserved

Fig21. *Configuración UARTCC*

Quedando la configuración del UART.c de la siguiente manera:

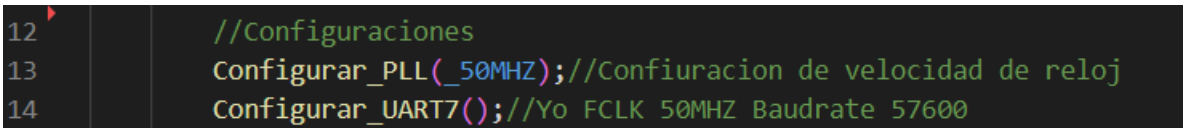


```c
C UART.c > Configurar_UART7(void)
#include "lib/include.h"

extern void Configurar_UART7(void)
{
    SYSCTL->RCGCUART  |= (1<<7);  //Paso 1 (RCGCUART) pag.344 UART/modulo0 0->Disable 1->Enable        *se habilita la señal de reloj del UART7*
    SYSCTL->RCGCGPIO |= (1<<4);     //Paso 2 (RCGCGPIO) pag.340 Enable clock port A     *Se activa el registro de reloj del GPIO, para el UART7 corr
    //(GPIOAFSEL) pag.671 Enable alternate function         *Se habilitan funcion alternativa del GPIO para configurar los pines*
    GPIOE->AFSEL |= (1<<1) | (1<<0);   //*Corresponde puerto E, pines 0 y 1
    //GPIO Port Control (GPIOPCTL) PE0-> Entrada (U0Rx) PE1-> Salida (U0Tx) pag.688
    GPIOE->PCTL |= 0x00000011;  // (1<<0) | (1<<4);//0x00000011         ** El pin 0 y 1 corresponde al port mux 0 y 1, por ello se pone en 1 el 1 y
    // GPIO Digital Enable (GPIODEN) pag.682      **GPIODEN - Para habilitar el pin como pin digital**
    GPIOE->DEN |= (1<<0) | (1<<1);//PE0 PE1          **Puerto E, pines 0 y 1 habilitados como digitales**
    //GPIODIR indica cual pin es entrada y cual es salida pag.663
    // GPIOE->DIR = (1<<1) | (0<<0); //Puerto E, pin 1 (PE1) es salida y pin 0 (PE0) es entrada

    //UART0 UART Control (UARTCTL) pag.918 DISABLE!!
    /* Desactivar el UART hasta terminar su configuración -  El bit que lo activa es UARTCTL
    El bit0 UARTEN es el habilitador, por eso se debe poner en 0
    El bit9 es para RXE y el 8 para TXE, recibir y transmitir, también se ponen en 0 para desactivar */
    UART7->CTL = (0<<9) | (0<<8) | (0<<0);

    //CALCULO VELOCIDAD COMUNICACIÓN
    /*
    BRD = 50,000,000/(16*57,600) = 54.2534

    Redondeo --->    UARTFBRD =  (.2534*64) + 0.5 = 16.7176 = 17

    */

    UART7->IBRD = 54;
    // UART Fractional Baud-Rate Divisor (UARTFBRD) pag.915
    UART7->FBRD = 17;
    //Parte decima - No se si esta bien???

    //  UART Line Control (UARTLCRH) pag.916
    /* La configuración de la trama se hace con el regitro de nombre generico UARTLCRH - Son 8 bits que se deben configurar:
       Solo nos interesa configurar WLEN -Longitud de palabra UART - Los bits indican el número de bits de datos transmitidos o recibidos
       FEN -Habilita si es que esta en 1 o deshabilita si es que esta en 0 los FIFOS de recepción y transmición Se recomienda siempre habilitarlos
    */
    UART7->LCRH = (0x3<<5)|(1<<4);//WLEN (bit5 conf) - para 8 bits indicado en  0x3      FEN (bit4 conf) - Habilita (1)  los FIFOS

    //  UART Clock Configuration(UARTCC) pag.939        controla la fuente de reloj en baudios para el módulo UART.

    UART7->CC =(0<<0);//Se pone en 0 para indicar Reloj del sistema
    //Disable UART0 UART Control (UARTCTL) pag.918
    UART7->CTL = (1<<0) | (1<<8) | (1<<9);    //Termino la configuración y ahora si se activa el UART  El Rx, Tx y el bit de activacion



}
```

Fig22. *Código configuración UART en C*

Una vez terminada la configuración, se editó el MAIN.c y UART.h.

En el MAIN.c se modificó la configuración de la velocidad a 50MHz y se indicó que el UART utilizado es el 7



```c
    //Configuraciones
    Configurar_PLL(_50MHZ);//Confiuracion de velocidad de reloj
    Configurar_UART7();//Yo FCLK 50MHZ Baudrate 57600
```

Fig23. *Modificación MAIN.c*

En el UART.h se módico el UART para indicar que el UART utilizado es el 7.

```
4    extern void Configurar_UART7(void);
```

Fig24. *Modificación UART.h*

Una vez hecho esto, se verifico que la comunicación se estuviera haciendo correctamente.

Se conecto el pin PE0 de la tiva al Tx y el PE1 de la tiva al Rx, además las tierras fueron conectadas una con otra.

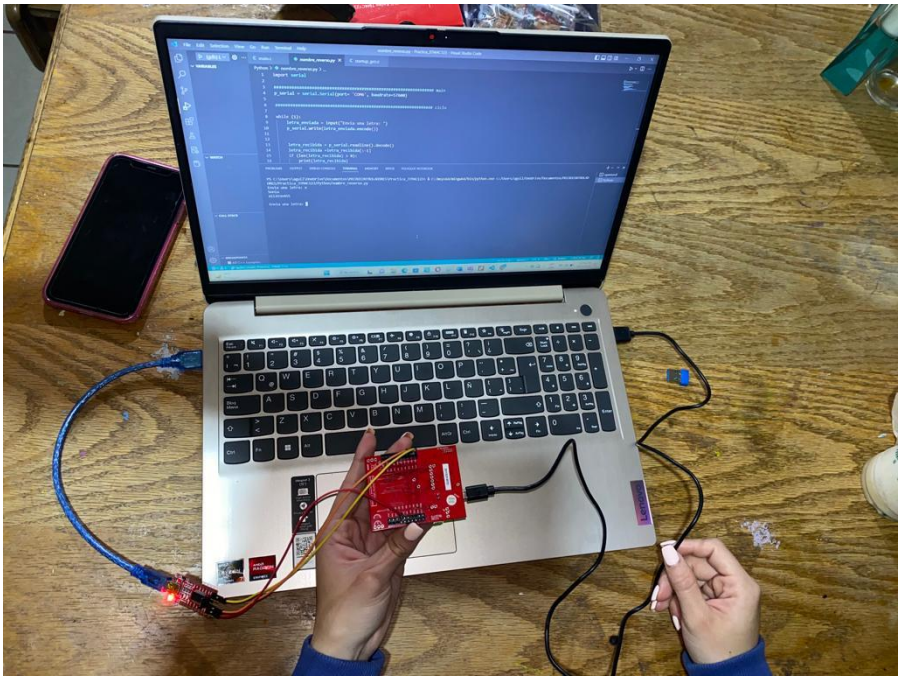


Fig25. *Conexión TIVA y RS-232*

Inicialmente, se buscaba que al enviar una letra por la terminal de python, se encendiera el led del color correspondiente y además enviara una respuesta por esta misma terminal.

El código principal se llevó a cabo de la siguiente manera, primero, se declaró una variable llamada c de tipo char, el cual se usa **para almacenar el valor entero,** a este se le dio el valor de '5', además se declaró una cadena de caracteres de tipo char, de longitud 32.

```
7       char c = '5';
8       char data_str[32] = " ";
```

Fig26. *Declaración variables*

Seguido de esto, están las configuraciones del PLL, UART y GPIO.

```
11
12      //Configuraciones
13      Configurar_PLL(_50MHZ);//Confiuracion de velocidad de reloj
14      Configurar_UART7();//Yo FCLK 50MHZ Baudrate 57600
15      Configurar_GPIO(); //Se configura GPIO
16
```

Fig27. *Configuraciones – MAIN.c*

Para esto, se llevo a cabo un código en MAIN.c, el cual está dentro de la función principal, se declaró que c va a leer el carácter que se enviara por la terminal, posteriormente se utilizó la función *switch,* para establecer distintos casos. Los casos son los siguientes, si se recibe una r, se encenderá el led rojo, el cual esta ubicado en el puerto F, en del pin 1, y además, devolverá una letra a; en el caso de enviar una b, se encenderá el red azul, que esta ubicado en el puerto F en el pin 2 y devolverá la letra b; en el caso de enviar una g, se encenderá el led verde, el cual esta ubicado en el pin 3 del puerto F, y se devolverá una letra c; en el caso de enviar una y, se encenderá el led amarillo, ya que el led que la tiva incluye es de tipo RGB, la combinación de colores que crean el color amarillo es verde y rojo, por ello, en este caso, se enciende el pin 1 (rojo) y el pin 3 (verde) los cuales están ubicados en el puerto F y se devolverá una d.

```
22
23          while(1)
24          {
25              c = readChar(); //lee el caracter enviado por terminal
26              switch(c)      //se establecen if para diferentes casos
27              {
28                  case 'r':
29                      //GPIODATA port F 662
30                      GPIOF->DATA = (1<<1); sprintf(data_str, "a\n");
31                      break;
32                  case 'b':
33                      //GPIODATA port F 662
34                      GPIOF->DATA = (1<<2); sprintf(data_str, "b\n");
35                      break;
36                  case 'g':
37                      //GPIODATA port F 662
38                      GPIOF->DATA = (1<<3); sprintf(data_str, "c\n");
39                      break;
40                  case 'y':
41                      //GPIODATA port F 662
42                      GPIOF->DATA = (1<<3) | (1<<1); sprintf(data_str, "d\n");
43                      break;
44
45
46
47                  default:
48                     GPIOF->DATA = (0<<1) | (0<<2) | (0<<3); sprintf(data_str, "\n");
49                   break;
50              }
51
52              printString(data_str);
53
```

Fig28. *Código MAIN.c -Experimento 1*

Experimento 2

```
//El envió es su nombre (rave)
// Invertirlo y regresarlo con números consecutivos entre letras (e1v2a3r)
```

Fig29. *Instrucciones experimento 2*

Por lo tanto, se buscaba que al enviar mi nombre **Sonia**, este, fuera devuelto de la siguiente manera **a1i2n3o4S5.** Para ello, se creo una carpeta la cual se llamó “Python”, aquí se creo un código en Python, el cual inicialmente se importó la librería necesaria para que pudiera haber comunicación serial entre la tiva y el RS-232; después se declaro una variable llamada Puerto_serial en la cual se declara el puerto y se ponen los parámetros ‘COM6’, velocidad y timeout = 0, siendo este parámetro el que permite seleccionar el tiempo máximo (en segundos) para que se complete la solicitud, nosotros por eso elegimos 0.

Se pone el ciclo while porque le timeout no espera y lo que se hace, es que el while siga rebobinando, hasta obtener dato. Es un ciclo infinito y nunca se va a cerrar. En cuanto se reciba una dato, entra en este ciclo y en la variable nombre se le da la instrucción readline, esta instrucción El método readlines() devuelve una lista que contiene cada línea del archivo como un elemento de lista. Una vez guardado, se da la instrucción de que nombre sea igual a la longitud de nombre menos 1.

Para enviar se utiliza la instrucción "**For i in range(0,LEN(data)):"**, donde el rango va desde 0 hasta la longitud de data**.** Con el ciclo for, se hace un arreglo, va a ir de 0 a el número de datos que se enviaron. De esta manera, el puerto serial va a escribir el arreglo de datos, donde será el número de datos menos el último dato menos i (i va a ir yendo en incremento cada ciclo).

Además, el puerto serial va a escribir el carácter (para eso se usa str) donde i va a ir aumentando en cada iteración, y el 1 se pone para que inicie desde 1 (que sea el primer carácter que se escribe.
Para finalizar, se agrego un caso en el código de MAIN.c, donde en caso de enviar la letra x, la respuesta será el nombre enviado (Sonia).

Una vez terminado el código se probó y después de varios arreglos de obtuvieron los siguientes resultados.

nombre_reverso.py > ...

```
1  Import serial as s
2  Puerto_serial=s.Serial('COM6',57600,TIMEOUT=0)
3
4  While(1):
5  {
6      if(Puerto_serial.inwaiting>0):
7      {
8          nombre=Puerto_serial.readline()
9
10         nombre = nombre[:-1]
11
12             for i in range(0,LEN(DATA)):
13             {
14                 Puerto_serial.write(nombre(LEN(nombre)-1-i])
15                 Puerto_serial.write(str(i+1))
16             }
17     }
18
19 }
20
21
```

Fig30. *Código Python – Experimento 2*

```
23          while(1)
24          {
25              c = readChar(); //lee el caracter enviado por terminal
26              switch(c)      //se establecen if para diferentes casos
27              {
28                  case 'r':
29                      //GPIODATA port F 662
30                      GPIOF->DATA = (1<<1); sprintf(data_str, "a\n");
31                      break;
32                  case 'b':
33                      //GPIODATA port F 662
34                      GPIOF->DATA = (1<<2); sprintf(data_str, "b\n");
35                      break;
36                  case 'g':
37                      //GPIODATA port F 662
38                      GPIOF->DATA = (1<<3); sprintf(data_str, "c\n");
39                      break;
40                  case 'y':
41                      //GPIODATA port F 662
42                      GPIOF->DATA = (1<<3) | (1<<1); sprintf(data_str, "d\n");
43                      break;
44
45                  case 'x': sprintf(data_str, "Sonia\n"); break;
46
47
48                  default:
49                     GPIOF->DATA = (0<<1) | (0<<2) | (0<<3); sprintf(data_str, "\n");
50                   break;
51              }
52
53              printString(data_str);
54
```

Fig31. *Código MAIN.c – Experimento 2*

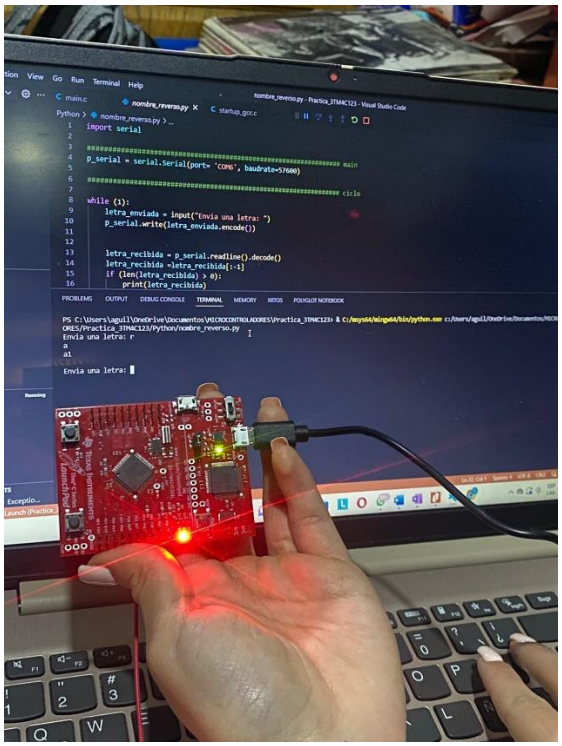


Fig32. *Evidencia funcionamiento – Respuesta led rojo*

Fig33. *Evidencia funcionamiento – Respuesta led azul*

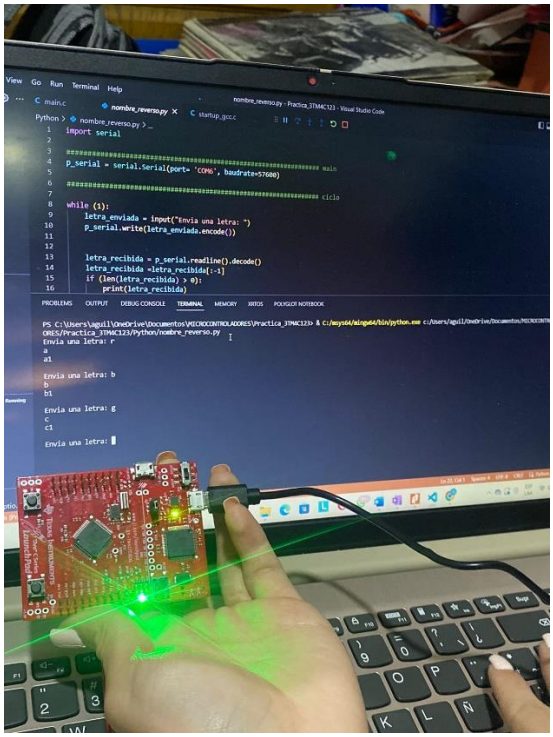


Fig34. *Evidencia funcionamiento – Respuesta led verde*

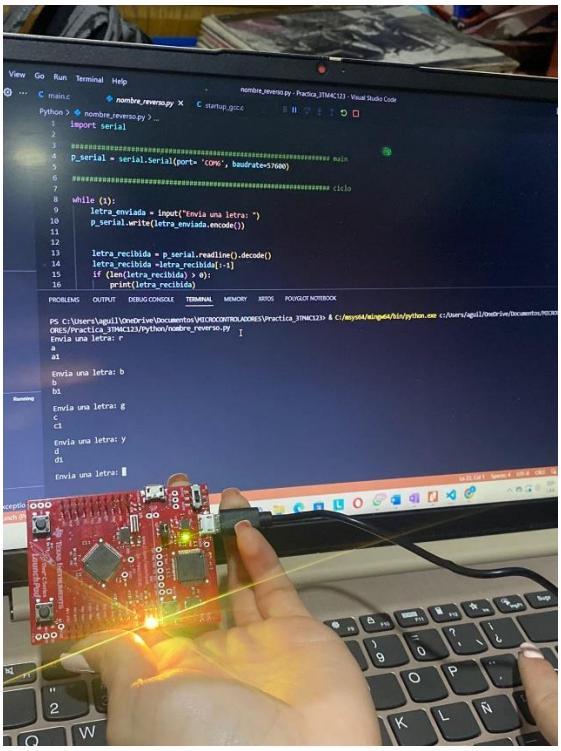


Fig45. *Evidencia funcionamiento – Respuesta led amarillo*

Fig40. *Evidencia funcionamiento – Respuesta nombre al revés con números*

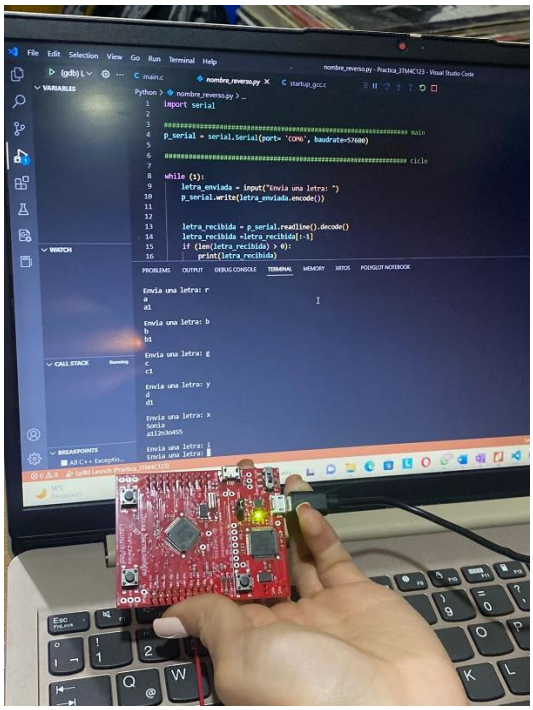


Fig41. *Evidencia funcionamiento – Respuesta caso default*

***EVIDENCIA: Video subido en el git**