PRÁCTICA 3

Comunicación UART con TM4C123C (tiva chica).

Universidad Autónoma de Querétaro

Microcontroladores.

Nieto Guerrero Andrea

29 de Enero del 2023

Comunicación UART.

UART significa receptor/transmisor asíncrono universal, y define un protocolo para el intercambio de datos bit a bit en serie entre dos dispositivos. UART es muy simple y 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. Los datos en UART se transmiten en forma de tramas.

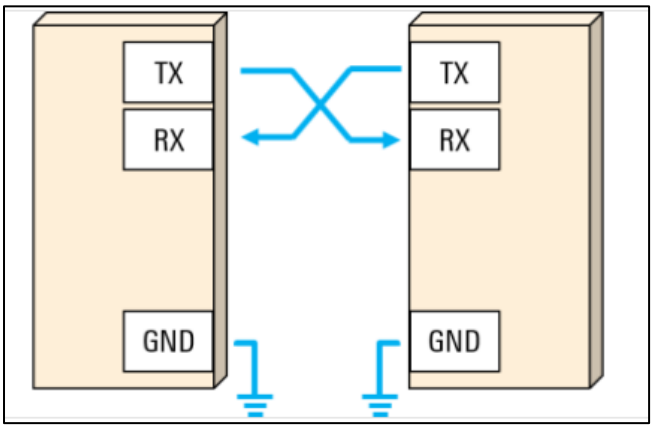


Figura 1. Comunicación entre dos dispositivos.

Una de las grandes ventajas del UART es que es asíncrono, por lo que el transmisor y el receptor no comparte una señal de reloj común. Para que exista una sincronización de bits es necesario que ambas terminales transmitan a la misma velocidad preestablecida, de no ser así, su comunicación no va a ser posible. Las velocidades de baudios del UART más comunes que se utilizan son: 4800, 9600, 19200, 57600 y 115200.

Metodología práctica 3.

Experimento 1.

Encender led rojo, azul y verde mediante comunicación UART, haciendo uso del UART2, con una frecuencia de reloj de 50Mhz y Baud-rate de 57600.

Para conocer los pines asociados al módulo 2, es necesario buscar en la página 895 de la datasheet.

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
U2Rx	53	PD6 (1)	I	TTL	UART module 2 receive.
U2Tx	10	PD7 (1)	O	TTL	UART module 2 transmit.

Figura 2. Señales UART 2.

Como lo indica la figura 2, el puerto asociado al módulo 2 es el D, en los pines D6 y D7. Datos que se utilizarán más adelante.

Para la configuración, comenzamos con una función que va a incluir los registros necesarios para el buen funcionamiento del UART2.

Lo primero que vamos hacer es habilitar y deshabilitar el módulo UART2, el registro correspondiente para esta función es RCGCUART y se encuentra en la página 344, como se muestra en la figura 2.

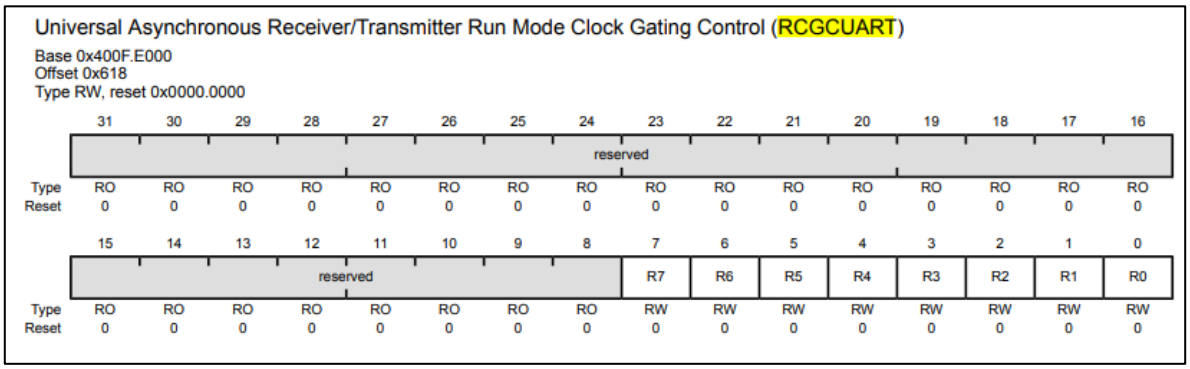


Figura 2. Registro RCGCUART

El bit 2 es el que se encarga de habilitar el módulo 2, por lo que escribimos un 1 en el bit 2.

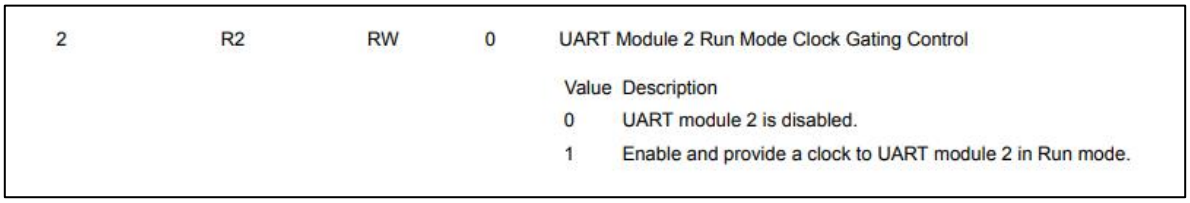
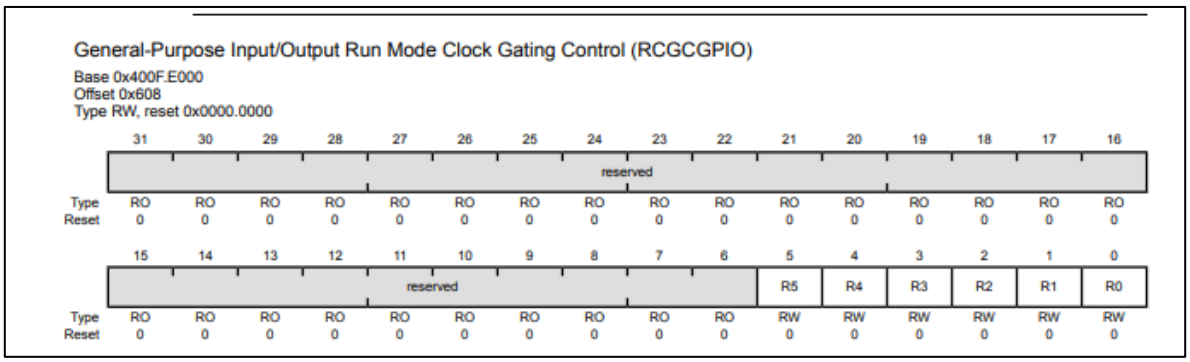


Figura 3. Bit 2 del registro RCGCUART

```
SYSCTL -> RCGCUART |= 0x4;
```

Figura 4. Asignación del registro RCGCGPIO en Visual Studio

El registro que activa el reloj de los puertos es RCCGPIO, lo encontramos en la página 340 de la datasheet, como se muestra en la imagen



General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO)

Base 0x400F.E000
Offset 0x608
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved										R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 5. Registro RCGCGPIO

2	R2	RW	0	UART Module 2 Run Mode Clock Gating Control

Value	Description
0	UART module 2 is disabled.
1	Enable and provide a clock to UART module 2 in Run mode.

Figura 6. Bit 2 del registro RCGCGPIO

Como se muestra en la figura 6, para habilitar el reloj en el puerto D, asignamos un 1 al bit 2.

```
SYSCTL -> RCGCGPIO |= 0x8;
```

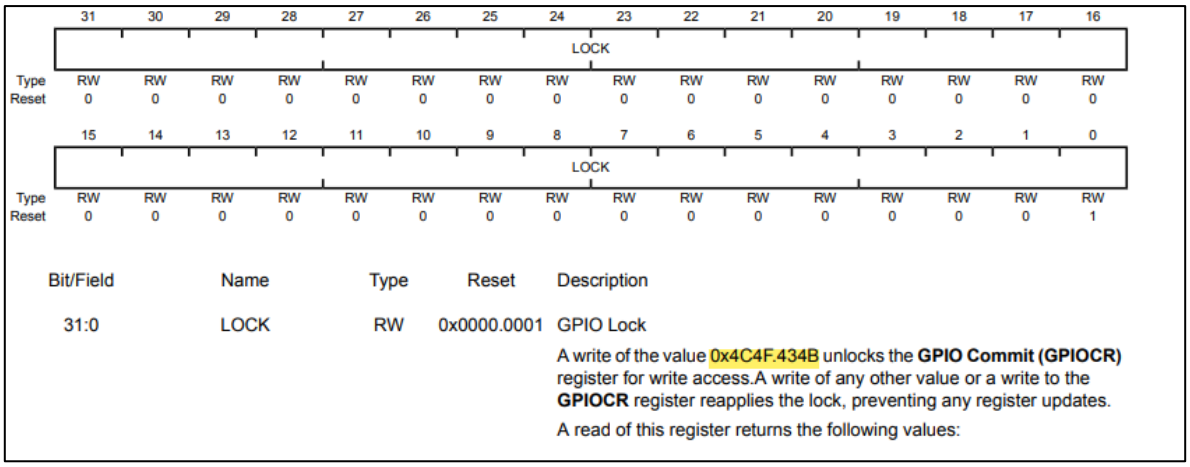
Figura 7. Asignación del registro RCGCGPIO en Visual Studio

En la página 650 nos encontramos con una tabla, la cual enlista los pines que cuentan con consideraciones especiales, entre ellos se encuentra el pin D7, por lo que va a ser necesario "desbloquearlo" para poder utilizarlo.

GPIO Pins	Default Reset State	GPIOAFSEL	GPIODEN	GPIOPDR	GPIOPUR	GPIOPCTL	GPIOCR
PA[1:0]	UART0	0	0	0	0	0x1	1
PA[5:2]	SSI0	0	0	0	0	0x2	1
PB[3:2]	$I^{21}C0$	0	0	0	0	0x3	1
PC[3:0]	JTAG/SWD	1	1	0	1	0x1	0
PD[7]	GPIO[a]	0	0	0	0	0x0	0
PF[0]	GPIO[a]	0	0	0	0	0x0	0

Figura 8. Pines con consideraciones especiales

Los registros que se encargan del desbloqueo de estos pines son GPIOLOCK y GPIOCR. El registro GPIOLOCK lo encontramos en la página 684.



Bit/Field	Name	Type	Reset	Description
31:0	LOCK	RW	0x0000.0001	GPIO Lock A write of the value 0x4C4F.434B unlocks the **GPIO Commit (GPIOCR)** register for write access.A write of any other value or a write to the **GPIOCR** register reapplies the lock, preventing any register updates. A read of this register returns the following values:

Figura 9. Bit 2 del registro GPIOLOCK.

El registro GPIOCR se encuentra en la página 685.

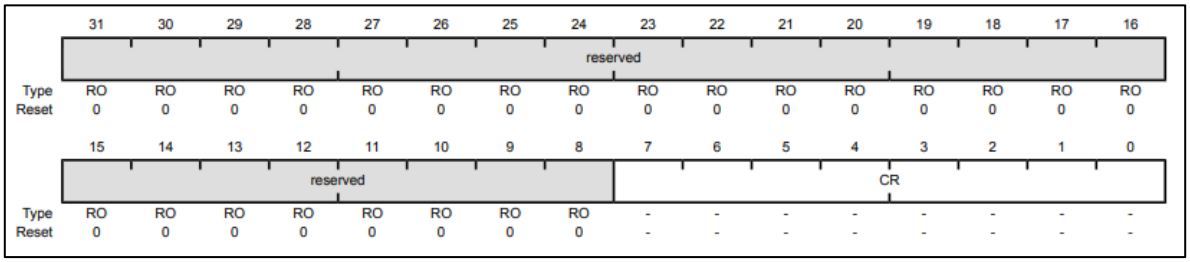


Figura 10. Bit 2 del registro GPIOCR

Como se muestra en la figura 9, el valor que se va asignar en el registro GPIOLOCK es 0x4C4F434B, para GPIOCR, del bit 0 al 7 hay que indicar los pines que se quieren desbloquear, en este caso 6 y 7, el valor hexadecimal correspondiente es 0xC0.

```
GPIOD -> LOCK = 0x4C4F434B;
GPIOD -> CR = 0xC0; // Bit 6 y 7
```

Figura 7. Asignación del registro GPIOLOCK y GPIOCR en Visual Studio

Ahora, se va configurar el registro AFSEL para indicarle al puerto que va funcionar con una de las funciones alternas.

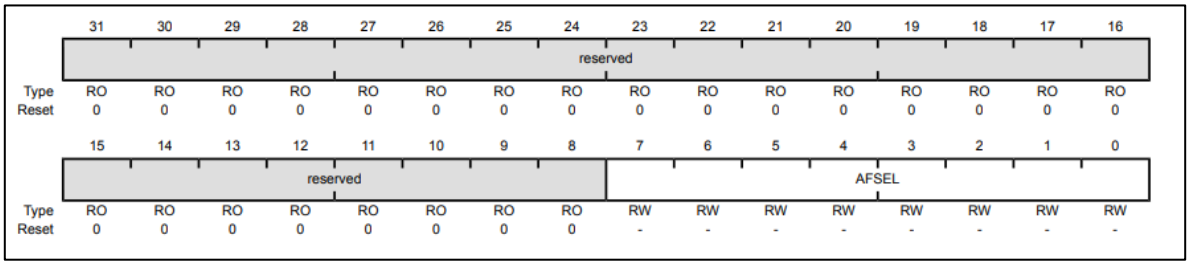


Figura 8. Registro AFSEL

7:0	AFSEL	RW	-	GPIO Alternate Function Select

Value	Description
0	The associated pin functions as a GPIO and is controlled by the GPIO registers.
1	The associated pin functions as a peripheral signal and is controlled by the alternate hardware function.

The reset value for this register is 0x0000.0000 for GPIO ports that are not listed in Table 10-1 on page 650.

Figura 9. BIT 0-7 del registro AFSEL

El pin al que le queremos asignar una función es el D6 y D7, por lo que se asigna un 1 en el bit 6 y 7, como se muestra en la figura 10.

```
GPIOD -> AFSEL |= (1 << 6) | (1 << 7); // pin D6 and D7
```

Figura 10. Asignación del registro AFSEL en Visual Studio.

Ya indicado que los pines D6 y D7 van a funcionar con una de las funciones alternas, se le asigna la función alterna a utilizar, en este caso analógica.

En la página 1351 se presenta una tabla (figura 11) la cual describe las funciones alternas de cada uno de los pines, la función analógica es la función alterna #1, valor que será asignado en el registro PCTL.

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PD6	53	-	U2Rx	-	-	MOFAULT0	-	PhA0	WT5CCP0	-	-	-	-
PD7	10	-	U2Tx	-	-	-	-	PhB0	WT5CCP1	NMI	-	-	-

Figura 11. Tabla de funciones alternas de los puertos D6 y D7.

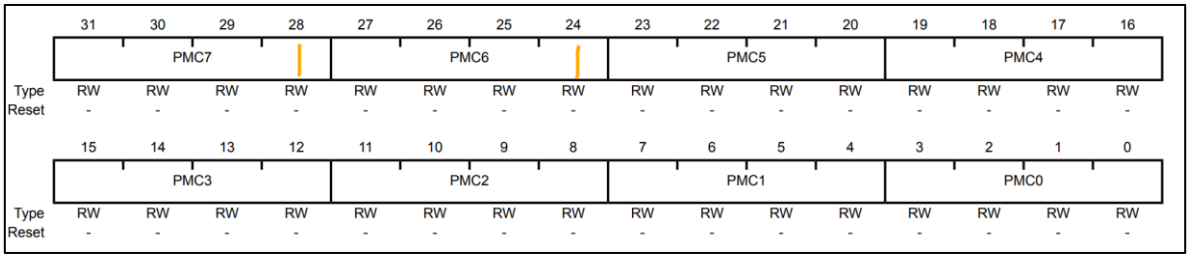


Figura 12. Registro PCTL.

Ya que la figura 15 nos muestra que la función alterna correspondiente a la señal analógica es la #1, asignamos un 1 (0001=1 hexadecimal) en el bit 24 y bit 28, el cual corresponde al pin D6 y D7. Adicionalmente se agrega una máscara, debido a que como se muestra en la figura 16, el registro PCTL, no cuenta con valores en el reset.

```
GPIOD -> PCTL |= (GPIOD -> PCTL & 0x00FFFFFF) | 0x11000000;
```

Figura 13. Asignación del registro PCTL en Visual Studio.

Ahora le indicamos al puerto D mediante el registro DEN, página 683, que el puerto D6 y D7 vaN ser un puerto digital.

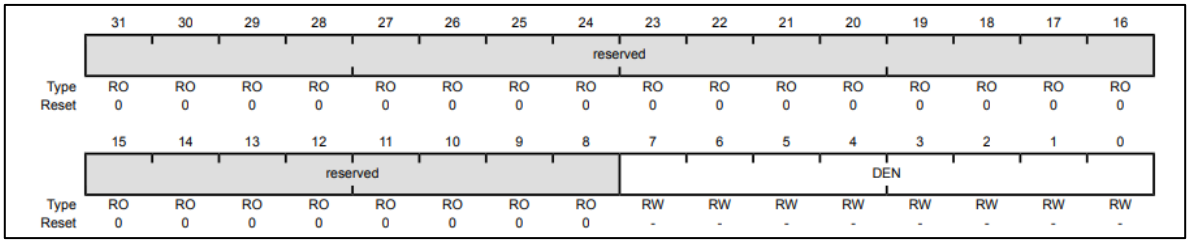


Figura 14. Registro DEN

7:0	DEN	RW	-	Digital Enable

Value	Description
0	The digital functions for the corresponding pin are disabled.
1	The digital functions for the corresponding pin are enabled.
	The reset value for this register is 0x0000.0000 for GPIO ports that are not listed in Table 10-1 on page 650.

Figura 15. Bit 0-7 del registro DEN.

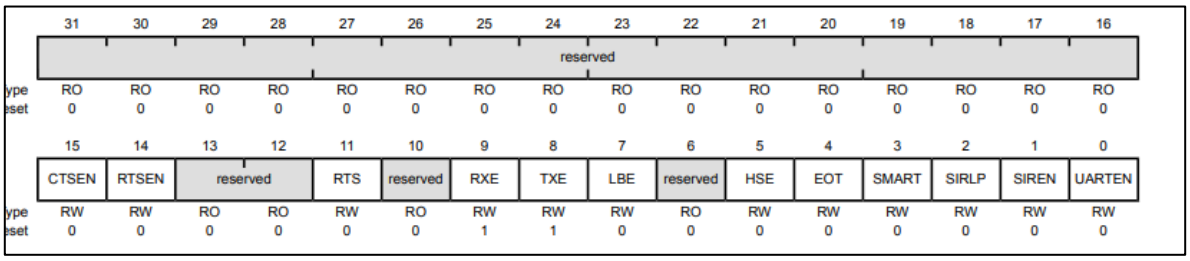
Como se muestra en la figura 15, para que funcione como puerto digital, es necesario asignar un 1, en este caso al pin D6 y D7 (bit 6 y 7), como se muestra a continuación:

```
GPIOB->DEN |= (1<<4);
```

Figura 20. Asignación del registro DEN en Visual Studio.

A continuación, configuramos los registros correspondientes al UART2.

Comenzamos por el registro UARTCTL, el cual es el registro de control, se encuentra en la página 918.



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
ype	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
eset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	CTSEN	RTSEN	reserved		RTS	reserved	RXE	TXE	LBE	reserved	HSE	EOT	SMART	SIRLP	SIREN	UARTEN
ype	RW	RW	RO	RO	RW	RO	RW	RW	RW	RO	RW	RW	RW	RW	RW	RW
eset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Figura 21. Registro UARTCL.

Deshabilitamos este registro previo a configurarlo asignando 0 en los bits 0, 8 y 9.

```
UART2 -> CTL = (0 << 0) | (0 << 8) | (0 << 9);
```

Figura 22. Asignación del registro UARTCTL en Visual Studio.

Cuentas correspondientes a baud rate:

$$\text{BAUD\_RATE} = 50,000,000/(16 * 57600) = 54.253472$$
$$\text{UARTFBRD}\,[\,\text{DIVFRAC}] = \text{integer}\,(0.253472 * 64 + 0.5) = 16.7 \text{ rounded} \rightarrow 17$$

Figura 23. Cuenas correspondientes a baud rate

Ahora, con el registro UARTIBRD, que se encuentra en la página 914, escribimos la parte entera de las cuentas correspondientes para baud rate

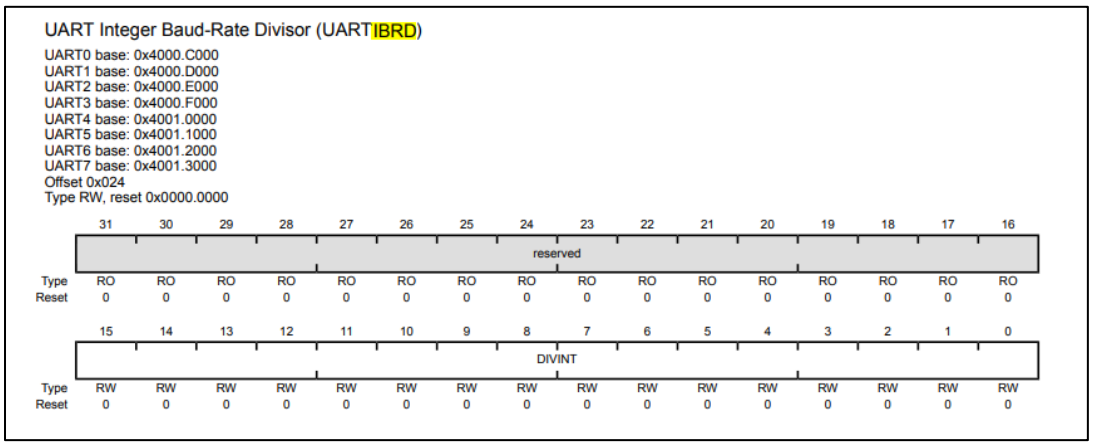


Figura 23. Registro UARTIBRD

Mientras que en el registro UARTFBRD escribimos la parte decimal:

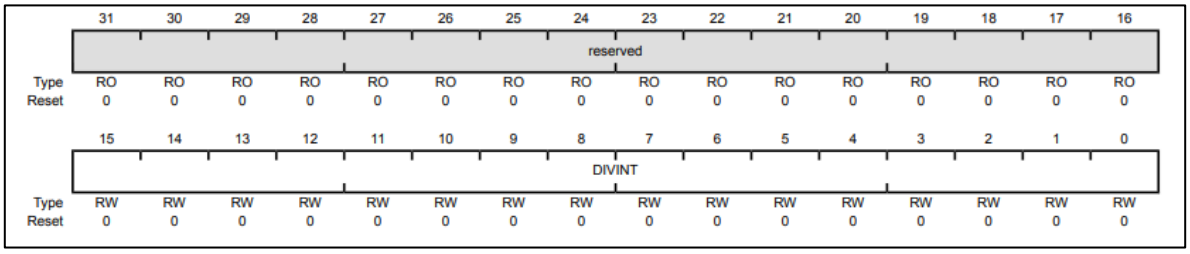


Figura 24. Registro UARTFBRD

```
UART2 -> IBRD = 54;
// Fractional  portion of brd
UART2 -> FBRD = 17;
```

Figura 24. Asignación del registro UARTIBRD y UARTFBRD en Visual Studio.

El registro UARTLCRH, es el registro de control de línea, donde establecemos 8 bits, que no vamos habilitar la paridad, habilitamos FIFO y se va tener 1 bit de paso

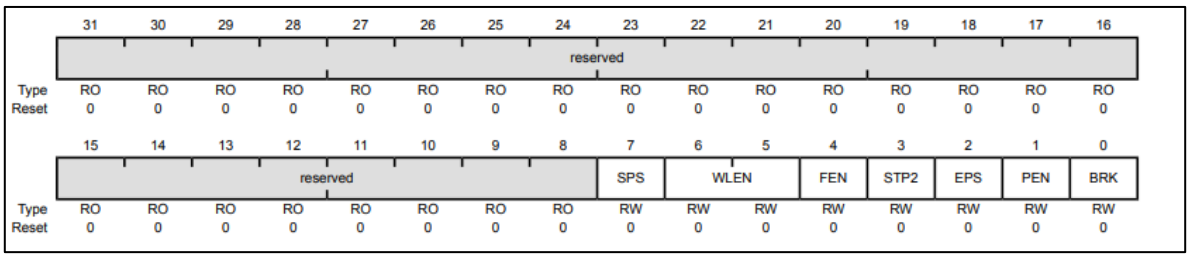


Figura 25. Registro UARTLCRH

```
UART2 -> LCRH = (0x3 << 5) | (1<<4);
```

Figura 26. Asignación del registro UARTLCRH.

El registro UARTCC controla la fuente de reloj en baudios para el módulo UART.

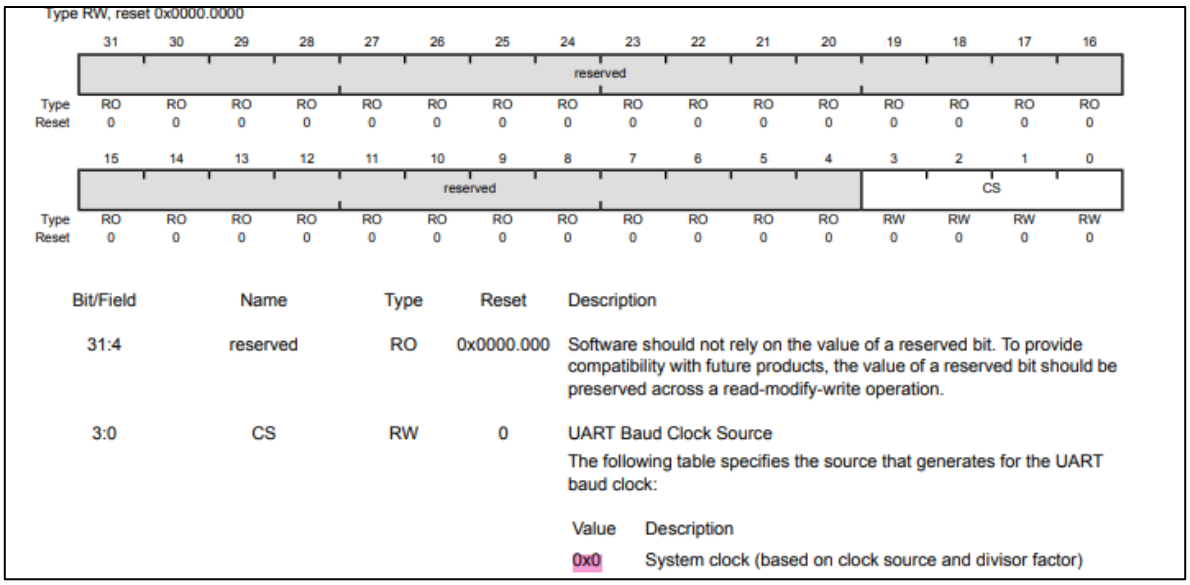


Figura 27. Registro UARTCC

En este caso asignamos un 0x0, porque se pretende utilizar el reloj del sistema.

```
UART2 -> CC = 0;
```

Figura 28. Asignación del registro UARTCC.

Finalmente habilitamos la recepción y transmisión de datos mediante el registro UARTCTL asignando un 1 en los bits 0, 8 y 9, como se muestra en la figura 29.

```
UART2 -> CTL = (1 << 0) | (1 << 8) | (1 << 9);
```

Figura 29. Asignación del registro UARTCTL.

Ya configurado el módulo UART2, se van a codificar funciones para lectura y escritura de caracteres, como se muestra en las imágenes 30 y 31.

```
extern char readChar(void)
{
    //UART FR flag pag 911
    //UART DR data 906
    int v;
    char c;
    while((UART2->FR & (1<<4)) != 0 );
    v = UART2->DR & 0xFF;
    c = v;
    return c;
}
```

Figura 30. Función de lectura de caracteres

```
extern void printChar(char c)
{
    while((UART2->FR & (1<<5)) != 0 );
    UART2->DR = c;
}
```

Figura 31. Función de escritura de caracteres

Para la función principal, dentro de un ciclo infinito, se codifico un switch case, donde dependiendo de la letra que recibía es el led que se encendía como se muestra en la imagen.

```
while(1)
{       c = readChar();
    switch(c)
    {
        case 'r':
            //GPIODATA port F 662
            printChar('a');
            GPIOF->DATA = (1<<1);
            break;
        case 'b':
            //GPIODATA port F 662
            printChar('b');
            GPIOF->DATA = (1<<2);
            break;
        case 'g':
            //GPIODATA port F 662
            printChar('c');
            GPIOF->DATA = (1<<3);
            break;
        case 'y':
            //GPIODATA port F 662
            printChar('d');
            GPIOF->DATA = (1<<3) | (1<<2);
            break;
        default:
            printChar((char)valor);
            GPIOF->DATA = (0<<1) | (0<<2) | (0<<3);
            break;
```

Figura 32. Switch case

Experimento 2.

Enviar andrea, y regresar al revés y con números intermedio entre cada letra, a1e2r3d4n5a6.

Para este experimento, la configuración del UART es idéntica.

Para el envío de caracteres se hizo uso de un script de Python como se muestra a continuación:

```
import serial as s
Ser=s.Serial('COM5',57600)
Ser.write(b'<')
Ser.write(b'a')
Ser.write(b'n')
Ser.write(b'd')
Ser.write(b'r')
Ser.write(b'e')
Ser.write(b'a')
Ser.write(b'>')


c=Ser.read(12)
print(c)
```

Figura 33. Script de python

En la función principal, se recibieron los datos uno a uno, hasta que se recibiera el símbolo '>'.

```
while (1)
    {

        c=readChar();
        if (c =='<')
        {
            while(condicion)
            {
                letra=readChar();
                if (letra!='>')
                    {
                        cuantos++;
                        *(letras+cuantos)=letra;

                    }
                else
                    {
                        condicion=false;
                    }
```

Figura 34. Recepción de datos

Ahora, mediante un ciclo for, se hace la construcción del nombre al revés con números entre cada letra, donde arreg es un arreglo que incluye valores del 0-F, porque FIFO es de 16 bits.

```
for (i=*puntero;i>0;i--)
{
    *(nombre+((cuantos-i)*2))=*(letras+i);
    //escribe letra sent inv
    printChar(*(letras+i));
    *(nombre+((cuantos-i)*2)+1)=*(arreg+(cuantos-i));
    //escribe simbolo sent ascendente
    printChar(*(arreg+(cuantos-i)));
}
```

Figura 35. Arreglo For.

RESULTADOS.

Experimento 1.

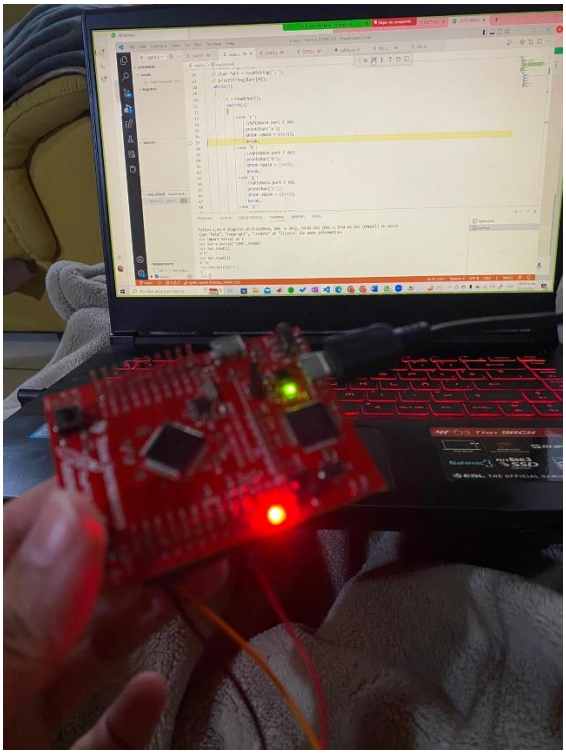


Figura 35. Encendido de led rojo, cuando se recibe una r

Experimeto2.

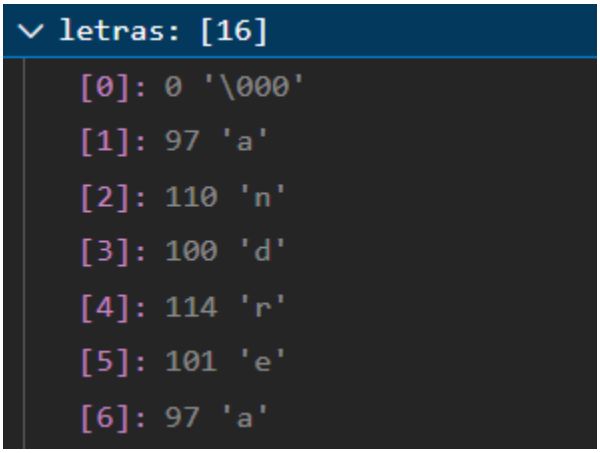


Figura 36. Recepción de andrea

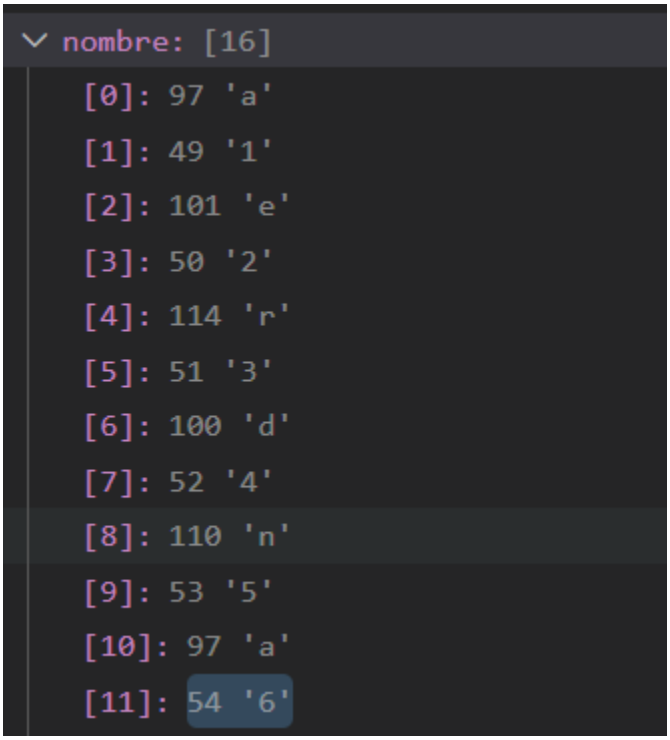


Figura 35. Construcción del nombre al revés

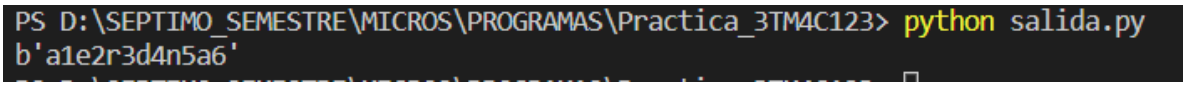


Figura 36. Impresión del nombre en python