



UNIVERSIDAD
AUTÓNOMA
DE QUERÉTARO



FACULTAD
DE INGENIERÍA

PRÁCTICA 3: UART MICROCONTROLADORES

RAMÍREZ ÁLVAREZ CARLO IVÁN - 280847
ONTIVEROS MARTÍNEZ BEATRIZ - 244784
TREJO DOMÍNGUEZ NELLY BIBIANA - 242494

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
INGENIERÍA BIOMÉDICA

MICROCONTROLADORES
ING. JOSÉ DE JESÚS SANTANA RAMÍREZ
ENERO, 2023

I. OBJETIVOS

Intercambiar datos en serie entre dos dispositivos, usando dos cables entre el transmisor y el receptor para transmitir y recibir en ambas direcciones.

- Recibir un carácter (r,g,b,t) para que se enciendan los leds correspondientes en la Tiva TM4C123G.
- Recibir una cadena de caracteres, invertirla y enviarla de vuelta intercalando números.

II. MARCO TEÓRICO

a. Comunicaciones asíncronas (UART's)

El receptor/transmisor asíncrono universal (Universal Asynchronous Receiver/Transmitter, UART) es el dispositivo clave de un sistema de comunicaciones serie. Su función principal es convertir los datos serie a paralelos cuando se trata de datos recibidos (de entrada) y de convertir datos paralelos a serie para transmisión (de salida). En la figura 1. se muestra el esquema general con los bloques básicos de un UART. Se distinguen los registros de datos, tanto de recepción como de transmisión y sus correspondientes registros de desplazamiento (RxD, TxD), los registros de control de transmisión y recepción y señales de sincronización para comienzo de la transmisión/recepción (RTS, CTS). Aunque la transmisión es realmente síncrona, tradicionalmente se les ha llamado asíncrona para hacer referencia al hecho de que existe flexibilidad para la transmisión de la siguiente palabra de datos.

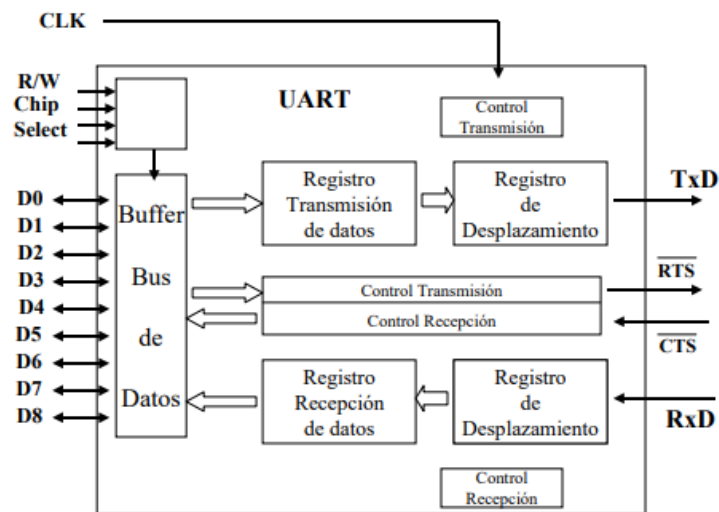


Figura 1. Diagrama de bloques de un Receptor/Transmisor Asíncrono Universal.

Los UART's también suministran otras funciones como las señales handshaking necesaria para interfaces RS-232-C. Entre las aplicaciones básicas de la UART se puede citar a las comunicaciones con el PC o con PDA's, transmisiones de audio, vídeo, juegos por infrarrojos o bluetooth (teléfonos móviles), y a matrices de diodos para paneles de atención al cliente, vallas publicitarias, etc. Ejemplos de UART's son el MC6850 de Motorola, llamado ACIA (Asynchronous Communication Interface Adapter) y el i8256 de Intel, por citar los más conocidos.

Los parámetros que se han de definir para el correcto diseño de un interfaz de comunicaciones usando un UART son los siguientes:

Sincronismo entre el receptor y el transmisor.

- Codificación de los datos.
- Prioridad en el envío de los bits.
- Tasa de envío de datos.
- Señales handshaking.
- Señales eléctricas de los valores lógicos.

La sincronización en la transmisión de los datos se lleva a cabo colocando en primer lugar un bit de comienzo (start bit), después se envían los datos (data bits) usualmente entre 5 y 9 bits empezando siempre por el bit menos significativo, LSB, y por último, se envía un bit de parada (stop bit). Los UART's que trabajan con 8 bits añaden un bit de detección de error o bit de paridad. Esto se realiza secuencialmente hasta completar la transmisión.

b. Temporizador y tono de los protocolos 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. Las velocidades de baudios del UART más comunes que se utilizan en estos días son 4800, 9600, 19.2K, 57.6K y 115.2K.

c. 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. Se puede configurar un segundo bit de parada (opcional), generalmente para darle tiempo al receptor de preparación para la siguiente trama, pero esto no es común en la práctica.

d. Bits de datos.

Los bits de datos son los datos de usuario o bits «útiles» y vienen inmediatamente después del bit de inicio. Puede haber de 5 a 9 bits de datos de usuario, aunque de 7 u 8 bits es lo más común. Estos bits de datos normalmente se transmiten primero con el poco menos importante.

Por ejemplo: si queremos mandar la «S» mayúscula en código ASCII de 7 bits, la secuencia de bits es 1 0 1 0 0 1 1. Primero y antes de enviarlos, invertimos el orden de los bits para ponerlos en el orden de bit menos importante, que es 1100101. Después de que se envía el último bit de datos, se utiliza el bit de parada para finalizar la trama y la línea regresa al estado de reposo.

- Código ASCII de 7 bits para «S» (0x52) = 1 0 1 0 0 1 1
- Orden de bit menos significativo = 1 1 0 0 1 0 1



Figura 2. Bits de inicio o parada.



Figura 3. Bits de datos.

III. MATERIAL

- Módulo Convertidor Serial USB a TTL – FT232RL
- 3 cables dupont macho hembra.
- Tiva LaunchPad TM4C123G.

IV. DESARROLLO DE LA PRÁCTICA

Habilitamos el reloj del UART y el del GPIO con registros. En este caso habilitamos el puerto C, pines 6 y 7.

```
7 #include "../include.h"
8
9 extern void Configurar_UART3(uint32_t baudrate,uint32_t reloj){
10 //Habilitar el reloj del uart y el del gpio con registros
11 SYSCTL->RCGCUART |= (1<<3); //M3
12 SYSCTL->RCGCGPIO |= (1<<2); //PUERTO C
13 //Configurar el gpio
14 GPIOC->AFSEL |= (1<<6) | (1<<7); //pines 6 & 7
15 //
16 //      Rx      Tx
17 GPIOC->DIR |= (0<<6) | (1<<7); //Entrada | salida
18 GPIOC->PUR |= (0<<6) | (0<<7); //Sin resistencia pull up
19 GPIOC->PDR |= (0<<6) | (0<<7); //sin resistencia pull down
20 GPIOC->DEN |= (1<<6) | (1<<7); //Pines digitales
21 GPIOC->AMSEL = 0x00; //desactivamos la funcion analogica
22 GPIOC->PCTL = (GPIOC->PCTL & 0x00FFFFFF) | 0x11000000; //que pines del puerto se configuren pag 689
23 }
```

Limpiamos los registros desactivando uart, tx y rx para configurarlos. Con una Velocidad 19200 baudrate, frecuencia de reloj 50,000,000.

```
23 UART3->CTL |= (0<<8) | (0<<9) | (0<<0) | (0<<4); //Limpiar los registros (desactivar uart, tx y rx) para configurarlos
24 //Velocidad 19200 baudrate , frecuencia de reloj 50,000,000
25 //400,000,000/(reloj + 1) = velocidad de reloj en Hz
26 uint32_t velocidad_de_reloj = (400000000/(reloj + 1));
27 uint32_t BRD_I = velocidad_de_reloj/(16*baudrate);
28 UART3->IBRD = BRD_I;
29 uint32_t BRD_F = (float)velocidad_de_reloj/(float)(16 * baudrate) - BRD_I;
30 BRD_F = lround(BRD_F * 64 +0.5);
31 UART3->FBRD = (int)BRD_F;
32 UART3->LCRH = (0x3<<5) | (1<<4); // Activa el fifo y el bit de paridad
33 //habilitar_IntUART();
34 UART3->CC = 0x00; //relo pag939 0x00 System clock
35 UART3->CTL |= (1<<8) | (1<<9) | (1<<0) | (1<<4); //Limpiar los registros (activar uart, tx y rx)
36 }
```

Leemos "n" carácter.

```
51 extern char Rx_char(void){ //leemos n caracter
52     int32_t v;
53     char c;
54     while((UART3->FR & (1<<4))!=0); //FR pag 911 flag de bandera
55     v = UART3->DR & 0xFF; // Dr 906
56     c = v;
57     return c;
58 }
```

Leemos una cadena.

```
74 extern char * Rx_string2(char string[], uint8_t len){ //leemos una cadena
75     uint32_t i=0;
76     do
77     {
78         string[i] = Rx_char();
79         if (string[i] == '%')
80         {
81             if (string[i-1] == '\r' && i > 1)
82             {
83                 string[i-1] = '\0';
84             }
85             else
86             {
87                 string[i] = '\0';
88             }
89             break;
90         }
91         i ++;
92     }
```

Enviamos un carácter y una cadena.

```
75 extern void Tx_char(char c){ //enviamos un caracter
76     while((UART3->FR & (1<<5))!=0);
77     UART3->DR = c;
78 }
79
80 extern void Tx_string(char *cadena){ //enviamos una cadena
81     while(*cadena){
82         Tx_char(*(cadena++));
83     }
84     Tx_char('\n');
85 }
```

Invertimos la cadena para el experimento 2.

```
91 extern char * invertir_string(char *str){ //invertimos una cadena
92     uint32_t n = strlen(str);
93     uint32_t i = 0;
94     char *inv = (char*)calloc(10,sizeof(char));
95     while(i<n){
96         *(inv + (n-i-1)) = str[i];
97         i++;
98         if(i%10==0){
99             inv = realloc(inv,(i+10)*sizeof(char));
100         }
101     }
102     return inv;
103 }
```

El código principal para intercambiar datos en serie entre nuestra Tiva y el módulo se presenta a continuación.

```

/**
 * main.c
 */
#include "include.h"

////////// EXP 1 al habilitar y configurar UART3 y EXP 2 al
recibir y enviar str
int main(void)
{
    SetSystemClock_80MHz(_25MHZ);
    Configurar_UART3(19200,_25MHZ);
    Configurar_GPIO();

    char data_str[16] = "";  /// arreglo donde se guardarán los str
    // uint8_t duty=0;

    while(1){
        ////////// EXP1 -->leds de la tiva
        /*
            char c;
            c= Rx_char();

            switch(c)
            {
            case 'r': {
                Tx_char('R');
                GPIOF->DATA = 0x02; //R

                break;
            }

            case 'g': {
                Tx_char('G');
                GPIOF->DATA = 0x08; //g

                break;
            }

            case 'b': {
                Tx_char('B') ;
                GPIOF->DATA = 0x04; //b
                break;
            }

            case 'T': {
                Tx_char('t') ;
                GPIOF->DATA = 0x02;
                SysTick_ms(.00001);
                GPIOF->DATA = 0x04;
                SysTick_ms(.00001);
                GPIOF->DATA = 0x08;
                SysTick_ms(.00001);
            }
        }
    }
}

```

```

        break;
    }

default:
    Tx_char('N') ;
    GPIOF->DATA = 0x00;
    break;
}

*/

////////// invertir y recibir string EXP

Rx_string2(data_str, sizeof(data_str) / sizeof(data_str[0]));
/////////recibir str
uint32_t n = strlen(data_str); ///longitud de lo recibido
int j= 0;
for(j=0; j<n; j++)
{
    char strinv= data_str[strlen(data_str)-j-1];    ///longitud(5) -0-1
= 4 --> data_str[4]
    Tx_char(strinv);
    // Tx_char(enviar[strlen(enviar)-i-1]);
    char num=j+49; //////////+49 para enviarlo de int a ascii
    Tx_char(num);
    // Tx_char(i+1);

}
/*
////////// recibir pwm -->práctica del pwm (practica)
char c;
c= Rx_char();

switch(c)
{
case 'r': {
    data_str[0] = Rx_char();
    data_str[1] = Rx_char();
    data_str[2] = Rx_char();
    data_str[3] = '\0';
    duty= atoi(data_str);
    break;

}

}*/
}
}

```

Para las conexiones de la Tiva TM4C123G con el Módulo Convertidor Serial USB a TTL - FT232RL conectamos el módulo 3, puerto C. De tal manera que PC6 corresponde a Rx y

PC7 corresponde a Tx, como se muestra en la figura 4.

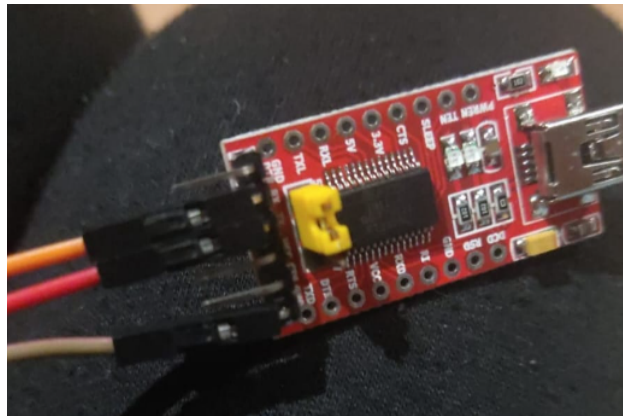


Figura 4. Conexiones Módulo Convertidor Serial USB a TTL – FT232RL.

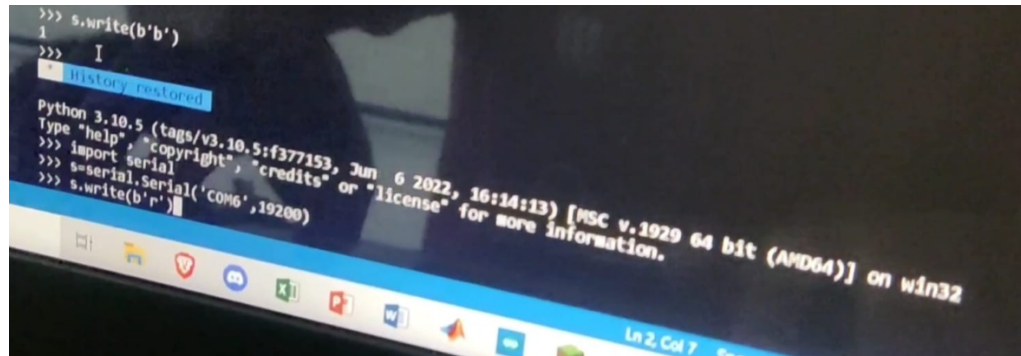


Figura 5. Conexión de pines Tiva TM4C123G.

V. RESULTADOS

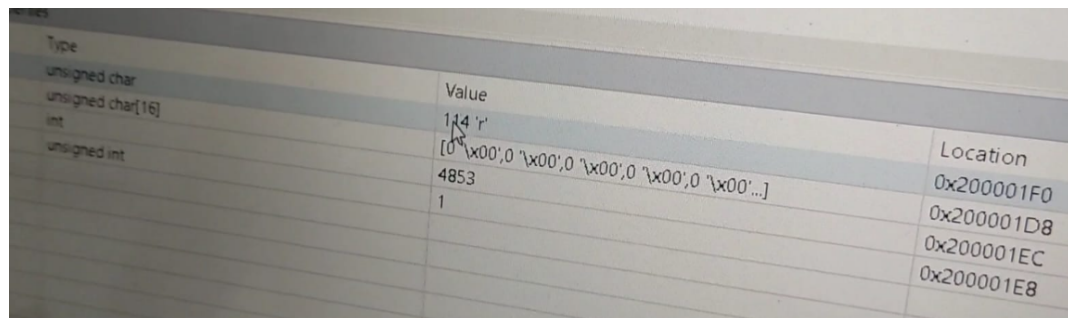
Experimento 1: Recibir un carácter (r,g,b,t) para que se enciendan los leds correspondientes en la Tiva TM4C123G.

- Abrimos Python en el Visual Studio Code.
- Importamos serial.
- Establecemos la velocidad y el COM donde se encuentra.
- Utilizamos la instrucción `s.write(b'r')` para enviar el carácter "r" que corresponde al led rojo de la tiva.



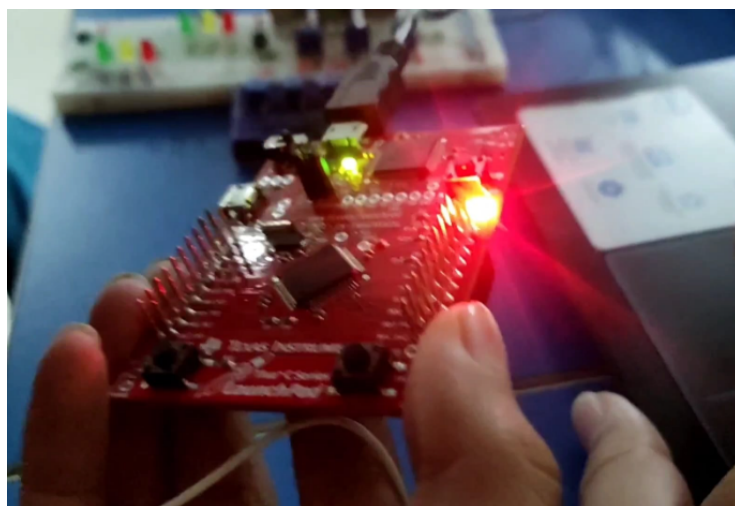
```
>>> s.write(b'b')
1
>>> I
History restored
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import serial
>>> s=serial.Serial('COM6',19200)
>>> s.write(b'r')
```

- La tiva recibe el carácter "r" en código ASCII, que corresponde al número 114.

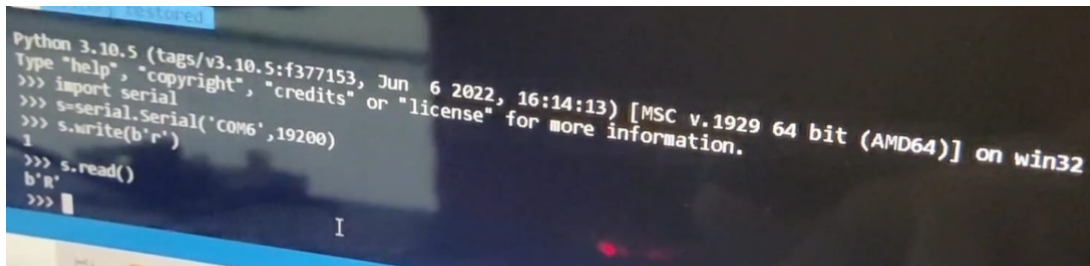


Type	Value	Location
unsigned char	114 'r'	0x200001F0
unsigned char[16]	[0 '\x00', 0 '\x00', 0 '\x00', 0 '\x00', 0 '\x00', 0 '\x00', ...]	0x200001D8
int	4853	0x200001EC
unsigned int	1	0x200001E8

- Enciende el led rojo en la Tiva TM4C123G.



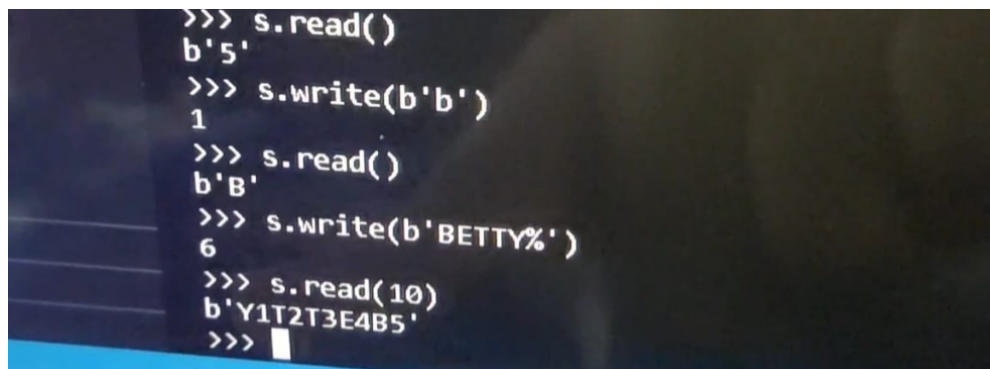
- Utilizamos la instrucción `s.read()` para leer lo que envía la tiva. En este caso regresa una "R", correspondiente al led rojo.



```
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import serial
>>> s=serial.Serial('COM6',19200)
>>> s.write(b'r')
1
>>> s.read()
b'r'
>>>
```

Experimento 2: Recibir una cadena de caracteres, invertirla y enviarla de vuelta intercalando números.

- Utilizamos las instrucciones en Python.
- Escribimos la cadena con el carácter de fin de trama que definimos en el código de recibir un str.
- Se observa la cadena invertida.



```
>>> s.read()
b'5'
>>> s.write(b'b')
1
>>> s.read()
b'B'
>>> s.write(b'BETTY%')
6
>>> s.read(10)
b'Y1T2T3E4B5'
>>>
```

BIBLIOGRAFÍA

Rohde & Schwarz. (s.f.). *Understanding UART*. Obtenido de Digital oscilloscope and probe fundamentals:
https://www.rohde-schwarz.com/lat/productos/prueba-y-medicion/oscilloscopios/educational-content/entendiendo-el-uart_254524.html