

# PRÁCTICA 2

Por Luis Díaz del Río Delgado

## Introducción

En esta práctica se nos pide configurar un módulo UART.

Las tareas pedidas son:

1. Imprimir por pantalla (Virtual Terminal) el nombre del Alumno/a (cualquier integrante del grupo es suficiente). En la misma línea de datos, se deberá imprimir una variable llamada 'contador', que deberá aumentar de valor constantemente (Permite visualizar que la comunicación está funcionando durante todo el proceso). Esta variable debe aumentar de uno en uno, cada 0.5 segundos.
2. Al detectar la tecla 'E' del teclado, el microcontrolador deberá encender un LED de color rojo. Al detectar la tecla 'A', el microcontrolador deberá apagar dicho LED. Por defecto, el LED debe permanecer apagado.
3. Al pulsar la tecla 'H', se deberá ejecutar una rutina que haga parpadear un LED verde cada 250 milisegundos. Volver a pulsar la tecla 'H', deberá deshabilitar dicha función y por tanto el LED verde deberá permanecer apagado
4. Al detectar la barra espaciadora, el contador del apartado 1, deberá resetearse y comenzar de nuevo la cuenta partiendo del valor 0.

Todo ello mediante MPLAB X IDE en conjunción con Proteus.

### Librerías, definiciones y variables globales:

Además de la librería "xc.h" la cual viene por defecto, he utilizado la librería "stdio.h" para hacer uso de la función "PRINTF()".

A su vez tenemos los siguientes "#define":

**LED\_GREEN:** En el tenemos almacenado el pin al que está conectado para que sea más sencillo trabajar con él.

**LED\_RED:** Igual que en LED\_GREEN. Solo cambia el pin.

**LED\_ON\_state y LED\_OFF\_state:** Guardamos un 1 y un 0 respectivamente. Tiene como objetivo dar claridad al código.

**Baud\_9600:** Tras aplicar la formula dada en los apuntes (Para BRGH = 1) sacamos que la velocidad de transmisión para la frecuencia de reloj elegida (4Mhz) es de 103. He decidido hacerlo así por la claridad de código.

Las variables globales son:

**TxBuffer:** Un array de char también llamado string que tiene como objetivo almacenar la información que será transmitida a la UART.

**Counter:** Es el contador pedido en el apartado 1 de tareas.

**DelayCounter:** No servirá para manejar el delay de nuestro programa.

```
#include "xc.h"
#include <stdio.h>

#define LED_GREEN LATAbits.LATA0
#define LED_RED   LATAbits.LATA1
#define LED_ON_state 1
#define LED_OFF_state 0

#define baud_9600 103

char TxBuffer[200];
int counter = 0;
int delayCounter = 0;
```

## Funciones

Hay varias funciones:

**Uart\_config:** Con valor de retorno void esta función se encarga de la configuración de los pines de la UART. Desde los pines de entrada y salida de la UART1 hasta el Baud Rate.

**Delay\_msg:** Como en la práctica anterior, esta función ejecuta la función en ensamblador "NOP" con el objetivo de perder tantos ciclos de espera como le digamos multiplicado por un valor elegido por prueba y error con la idea de reducir el uso de la CPU.

**sendChar:** Encargar de mandar toda aquella información de tipo char que entre como parámetro a la función. Mediante un bucle while esperamos a que la FIFO deje de estar completa. Una vez sale del bucle, le asignamos al puerto de entrada de la UART1 el valor del char que ha entrado por parámetro para que sea mandado.

**sendString:** Mediante punteros recorreremos un array de char, por cada iteración mandamos cada char con la función sendChar, explicada con anterioridad, hasta encontrarnos con un símbolo NULL ('\0') que hace de separador. Esto nos permite distinguir varios strings y chars dentro de un mismo buffer e imprimir cada uno de ellos manteniendo el significado o valor deseado.

**arrayInit:** La función se encarga de inicializar cada elemento del array TxBuffer a NULL ('\0'). Cuando escribamos en TxBuffer el mensaje que deseamos mandar, reescribiremos sobre elementos de tipo NULL, sin preocuparnos de añadir los separadores de otras formas.

(Imágenes del código en la siguiente hoja)

```

void uart_config(unsigned short baud){
    //UART PORT/PIN COMMUNICATION ASSIGNMENT
    TRISCbits.TRISC0 = 1; //Pin RC0 input (digital).
    RPINR18bits.U1RXR = 16; //Pin RC0 connected to reception port of UART1.
    RPOR8bits.RP17R = 3; //Pin RC1 connected to transmission pin of UART1.

    //U1MODE REGISTER CONFIG
    U1MODEbits.UARTEN = 0; //Unable UART before config.
    U1MODEbits.USIDL = 0; //Continue operation in idle mode.
    U1MODEbits.IREN = 0; //Infrared.
    U1MODEbits.RTSMD = 1; //Flow control. No using RTS/CTS. RX/TX mode.
    U1MODEbits.UEN = 0; //Only using TX and RX pin.
    U1MODEbits.WAKE = 0; //Awake when idle mode and recive data.
    U1MODEbits.LPBACK = 0; //Loopback desable.
    U1MODEbits.ABAUD = 0; //Auto baud rate desable.

    U1MODEbits.URXINV = 0; //Idle state = '1'.
    U1MODEbits.BRGH = 1; //High-Speed Mode (1 bit = 4 clock cycles).
    U1MODEbits.PDSEL = 0; //8 bits data lenght and null parity.
    U1MODEbits.STSEL = 0; //1-bit Stop at the end of data frame (8N1).

    //U1STA REGISTER CONFIG
    U1STAbits.URXISELO = 0;
    U1STAbits.URXISEL1 = 0;
    U1STAbits.ADDEN = 0;
    U1STAbits.UTXBRK = 0; //Sync frame desable.
    U1STAbits.OERR = 0; //Buffer is empty (No Overflow problems)
    U1STAbits.FERR = 0;
    U1STAbits.FERR = 0;
    U1STAbits.UTXEN = 1; //Enable transmisor

    U1BRG = baud;
    U1MODEbits.UARTEN = 1; //Enable UART after config.
}

```

```

void delay_ms(unsigned long time_ms){
    unsigned long i;
    for(i=0; i < time_ms*100; i++){
        asm("NOP");
    }
}

void sendChar(char c){
    while(U1STAbits.UTXBF);
    U1TXREG = c;
}

void sendString(char *s){
    while(*s != '\0'){
        sendChar(*(s++));
    }
}

void arrayInit(){
    int i;
    for(i = 0; i < 200; i++){
        TxBuffer[i] = '\0';
    }
}

```

## Main

Lo primero que hacemos en el main es ajustar la frecuencia del oscilador a la deseada. En mi caso he elegido una frecuencia de 4Mhz. He elegido esta frecuencia por varios motivos, ya he trabajado en un proyecto con esta misma frecuencia (práctica 1) por lo que me es más cómodo. Por otro lado es una frecuencia no muy alta por lo que no consumirá tanta CPU.

Para pasar de los 8Mhz de serie a los 4Mhz hemos utilizado las fórmulas dadas en clase y siguiendo el mismo proceso que en la práctica anterior.

Tras elegir si vamos a trabajar como analógico o digital, especificamos la direccionalidad de los pines A0 y A1 (Para el LED verde y el LED rojo respectivamente). Los dos van a ser pines de salida y van a estar apagados por defecto.

Mediante "uart\_config(baud\_9600)" configuramos la UART y le especificamos a que baudios va a trabajar. Además, inicializamos cada elemento del array a NULL.

La variable "pressH" es utilizada para saber cuando el LED verde debe estar apagado o parpadeando.

Finalmente entramos en el bucle infinito donde se definirá el comportamiento de nuestro programa.

Según el enunciado de la práctica, debemos mandar el nombre de un componente del grupo (En este caso voy solo) junto con un contador cada 0.5s, que equivaldría a 500ms. A su vez nos pide que al pulsar la tecla correspondiente a la letra 'h' el LED verde parpadee cada 250ms, por lo que el delay general va a ser de 250ms (al ser la más baja) y vamos a esperar dos veces ese delay mediante el contador delayCounter para conseguir sacar el nombre cada 500ms.

Con el primer "if" compruebo lo dicho con anterioridad, que realmente ya han pasado 500ms desde la última vez. Una vez ya dentro utilizo la función "sprintf" (incluida en la librería "stdio.h") para guardar dentro de TxBuffer un string con mi nombre, junto con el contador pedido en el enunciado de la práctica, un retorno de carro y un salto de línea.

Llamamos a la función sendString para mandar nuestro mensaje a la UART, aumentamos el contador para que en la siguiente iteración el mensaje se mande con un nuevo número y reseteamos delayCounter.

Dentro del segundo "if" procesaremos la información transmitida por la UART al micro. Para empezar se hace una comprobación que cumple medianamente bien su propósito. La idea inicial que tenía era que, mediante un while, esperara a que la FIFO no estuviera vacía e intentara leer algo que no había. El problema era que, al estar esperando la UART un input que más tarde sería mandado al micro, el programa no avanzaba hasta que se le añadía ese input. Por otro lado, si no se hacía ninguna comprobación la UART mandaba constantemente el último valor que había tenido la FIFO hasta que volviera a tener un valor dentro de ella.

La solución fue este "if". No funciona a la perfección, pero se asemeja bastante a los resultados que me hubiera gustado conseguir.

Dentro del "if" comprobamos que valor Ascii nos llega desde la UART. Si el valor es 101 (e), encendemos el LED rojo, si es 97 (a) lo apagamos. En el caso de que el valor sea 104 (h) aumentamos el contador pressH (con esto cambiaremos el estado del LED verde más

adelante). Si el valor es 32 (barra espaciadora) reseteamos "counter" que es el contador del apartado 1.

Finalmente hacemos la comprobación del contador pressH. En el caso de que sea par, apagamos el LED verde. En el caso de que sea impar lo dejamos parpadeando.

```
int main(void) {
    //Fosc = Fpri*M/(N1*N2) => 8MHz*4/(2*2) = 32/4 = 8;
    //Fcpu = Fosc/2 = 8MHz/2 = 4MHz;
    PLLFBD = 2;
    CLKDIVbits.PLLPRE = 0;
    CLKDIVbits.PLLPOST = 0;
    while(OSCCONbits.LOCK != 1);

    //Control de pin para trabajar como analógico/digital
    AD1PCFGL = 0xFFFF;

    //Control de direccionalidad de los pines de mi proyecto
    TRISAbits.TRISA0 = 0;    //Pin A0 -> D1 output (Green LED)
    TRISAbits.TRISA1 = 0;    //Pin A1 -> D2 output (Red LED)

    uart_config(baud_9600);
    arrayInit();
    int pressH = 0;

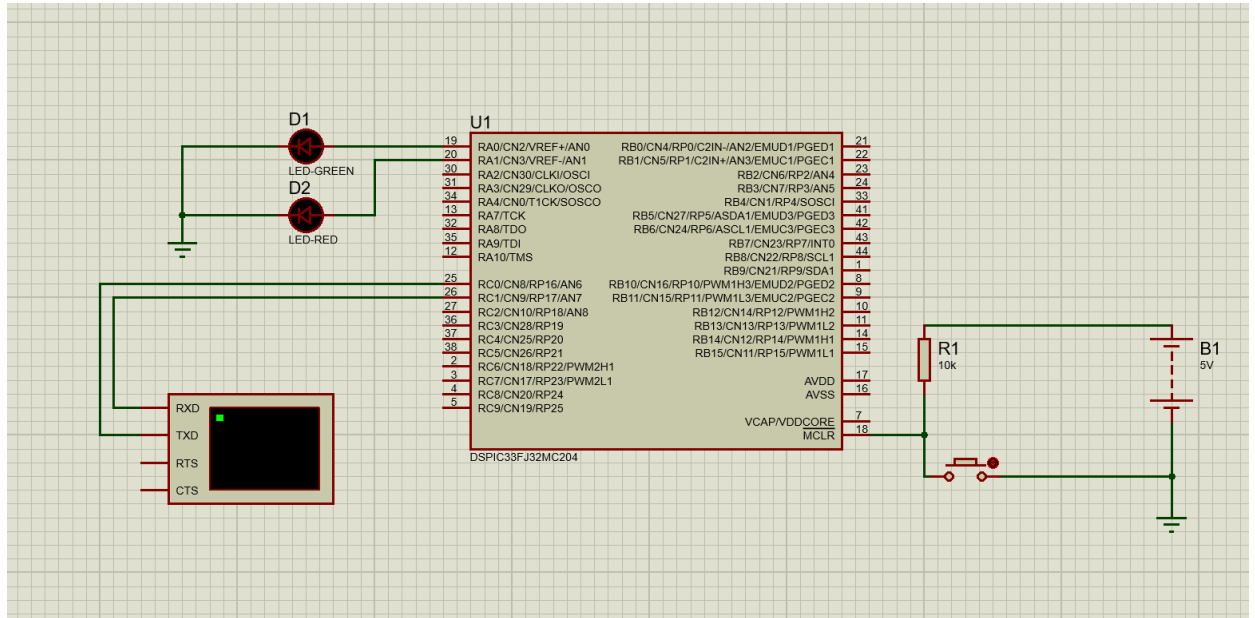
    while(1){
        if(delayCounter++ == 1){
            sprintf(TxBuffer, "Luis %d \r \n", counter);
            sendString(TxBuffer);
            counter++;
            delayCounter = 0;
        }
        if(U1STAbits.URXDA){
            if(U1RXREG == 101){ // e
                LED_RED = LED_ON_state;
            }
            if(U1RXREG == 97){ // a
                LED_RED = LED_OFF_state;
            }
            if(U1RXREG == 104){ // h
                pressH++;
            }
            if(U1RXREG == 32){ //Spacebar
                counter = 0;
            }
        }
        if((pressH % 2) == 0){
            LED_GREEN = LED_OFF_state;
        }else{
            LED_GREEN = !PORTAbits.RA0;
        }

        delay_ms(250);
    }
    return 0;
}
```

## Resultados.

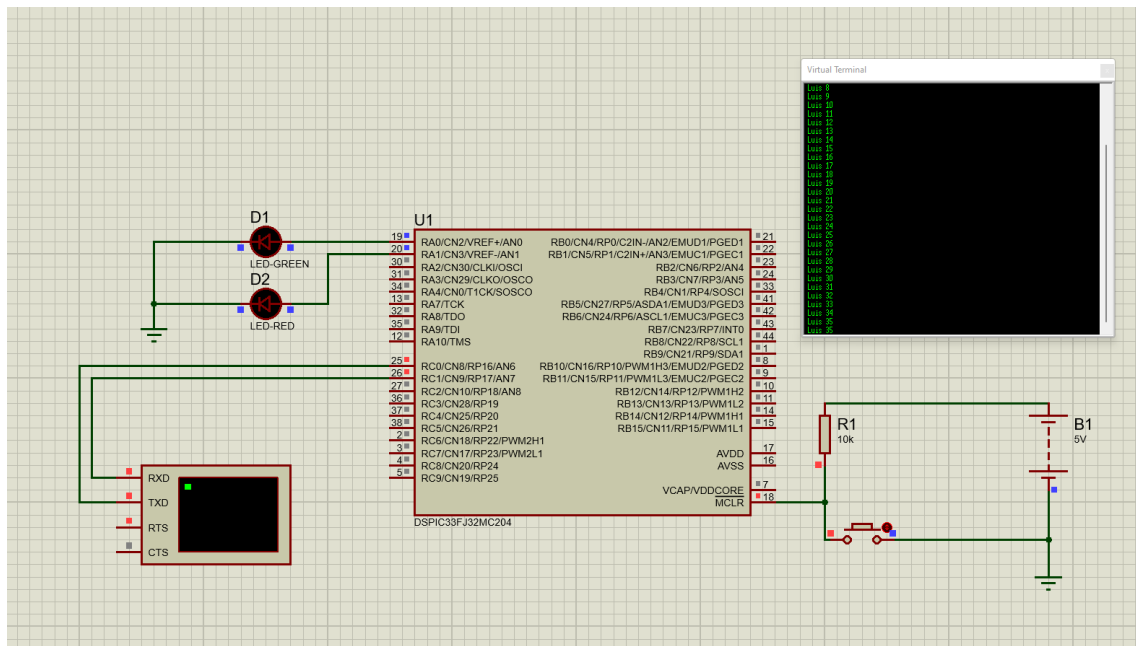
Debo comentar antes de nada que ha sido imposible conseguir una captura de pantalla en la que se viera el LED verde encendido al parpadear tan rápido.

Esquema de Proteus:



### Apartado 1:

Lo primero de todo es mostrar que efectivamente el apartado 1 funciona correctamente, en la siguiente imagen se muestra la simulación de Proteus junto al terminar virtual sacando mi nombre junto al número de la iteración.



Si nos acercamos a ver la Terminal Virtual veríamos esto:



## Virtual Terminal

```

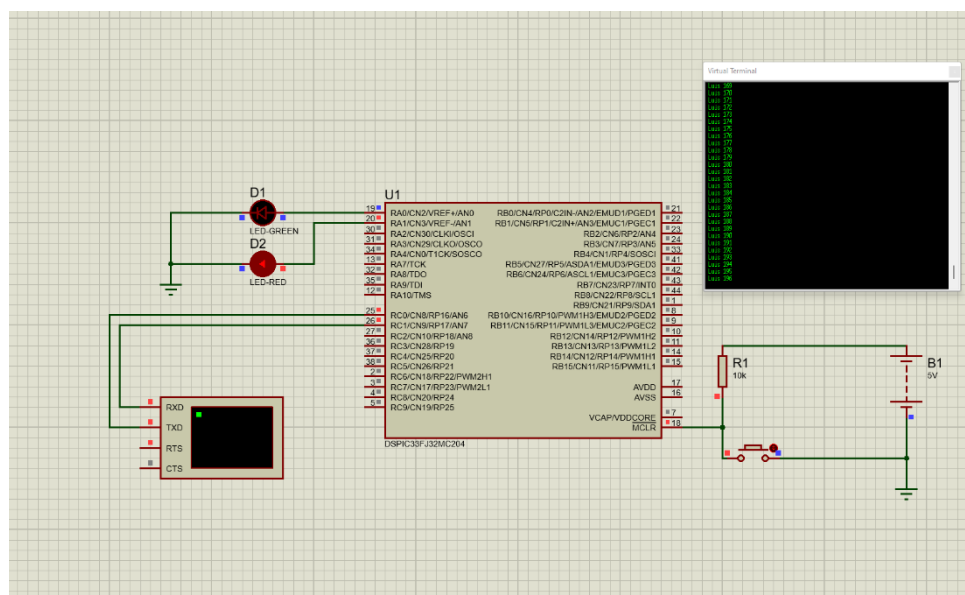
Luis 8
Luis 9
Luis 10
Luis 11
Luis 12
Luis 13
Luis 14
Luis 15
Luis 16
Luis 17
Luis 18
Luis 19
Luis 20
Luis 21
Luis 22
Luis 23
Luis 24
Luis 25
Luis 26
Luis 27
Luis 28
Luis 29
Luis 30
Luis 31
Luis 32
Luis 33
Luis 34
Luis 35
Luis 35

```

Si se amplía la imagen anterior (donde se muestra el Terminal Virtual junto al esquema en Proteus) se verían los mismos resultados, pero de esta forma podemos ver más de cerca que efectivamente funciona.

## Apartado 2:

Los resultados que arrojaron las pruebas son satisfactorios. En primer lugar he de comentar que esta prueba la realice de las últimas porque no me acordaba exactamente del orden de cada apartado. Digo esto porque los números en la Terminal Virtual son considerablemente más altos que los mostrados en el apartado anterior.



Como podemos observar el LED rojo se enciende correctamente.

Tras esa comprobación debemos ver si realmente se apaga.

Quiero hacer otra anotación aquí. Existe cierto “delay” entre el momento que la tecla es pulsada y se hace la captura. En este caso y según las capturas, el LED rojo debería haberse encendido entorno a la iteración 196, pero esto ocurrió unas iteraciones atrás. Con la captura en la que se muestra que el LED rojo se apaga ocurre exactamente lo mismo.

En la siguiente captura se ve que el LED rojo se apaga entorno a la iteración 242, algo que como he dicho antes, se aleja de la realidad. El primer valor que podemos ver dentro de la Terminal Virtual es el 215, algo más cercano al momento en el que es pulsada la tecla ‘a’.

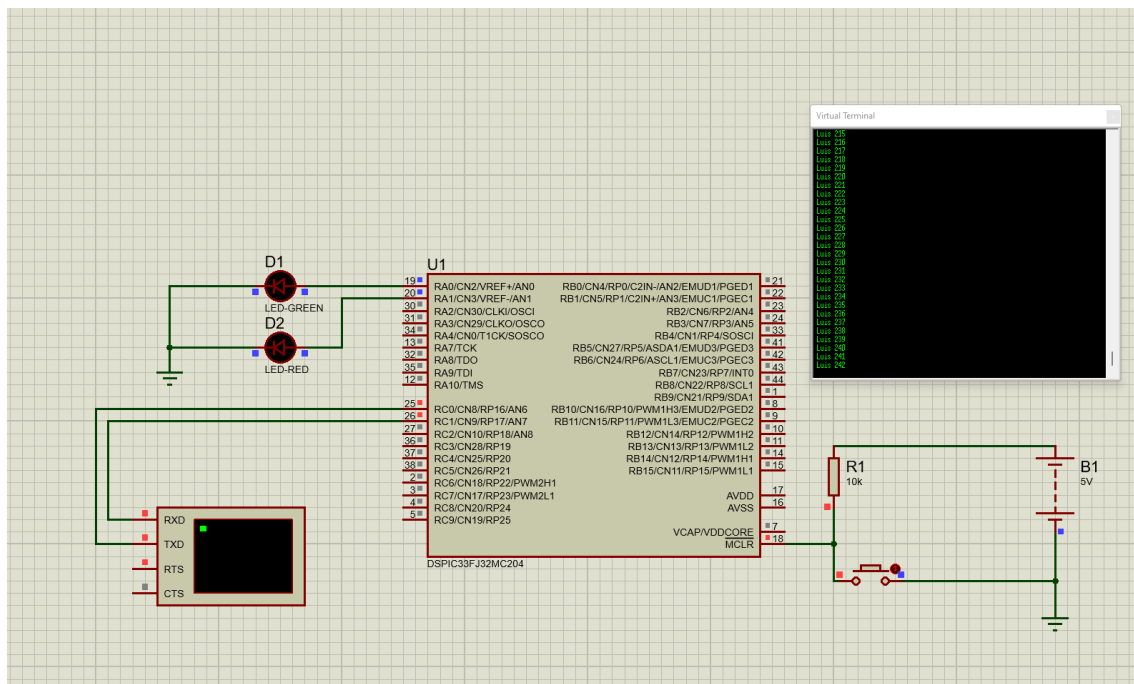


Imagen ampliada del terminal virtual del LED rojo encendido:

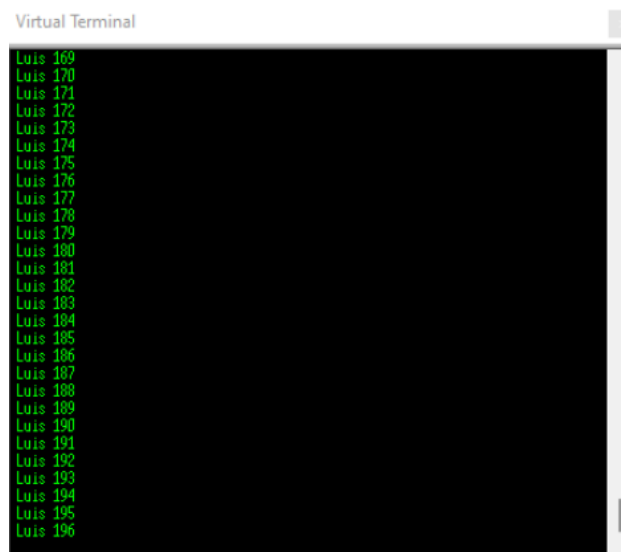
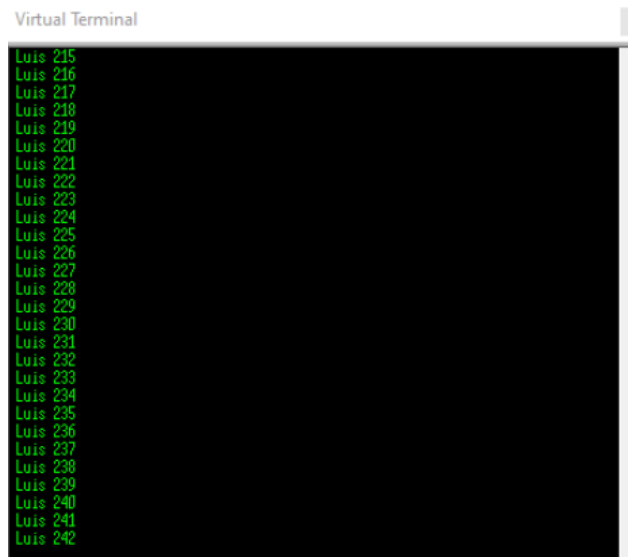


Imagen ampliada del terminal virtual del LED rojo apagado:



Apartado 3:

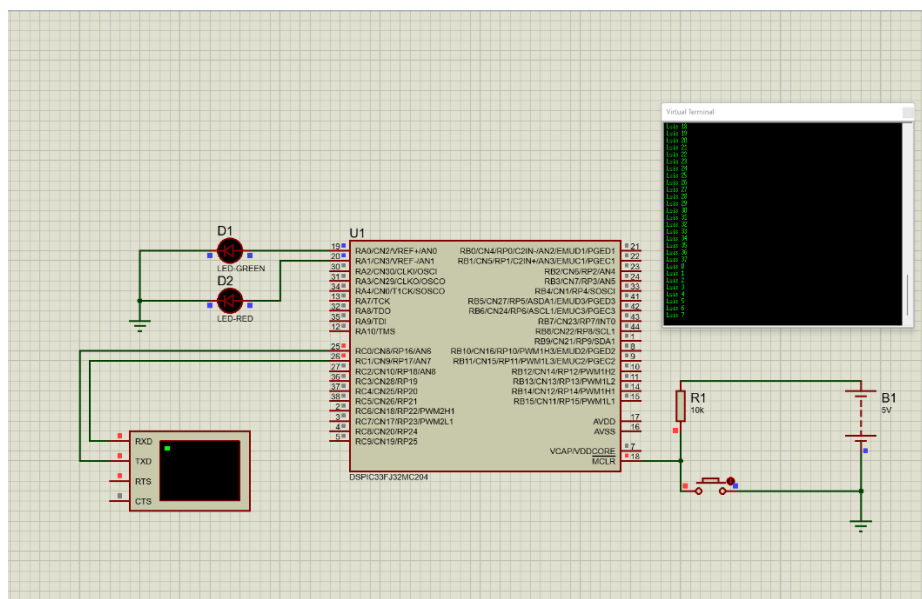
Ya he comentado antes que se me ha hecho imposible realizar la captura del LED verde encendido. No se si es por la herramienta utilizada para las capturas (Gyazo) o por la dificultad de encontrar el momento justo para hacer la captura.

Aunque a simple vista se puede apreciar en el código que funciona, no tener una demostración visual afecta a la veracidad del funcionamiento de este apartado. Si ejecuta el código junto a la simulación en Proteus podrá verificar por si mismo que realmente funciona.

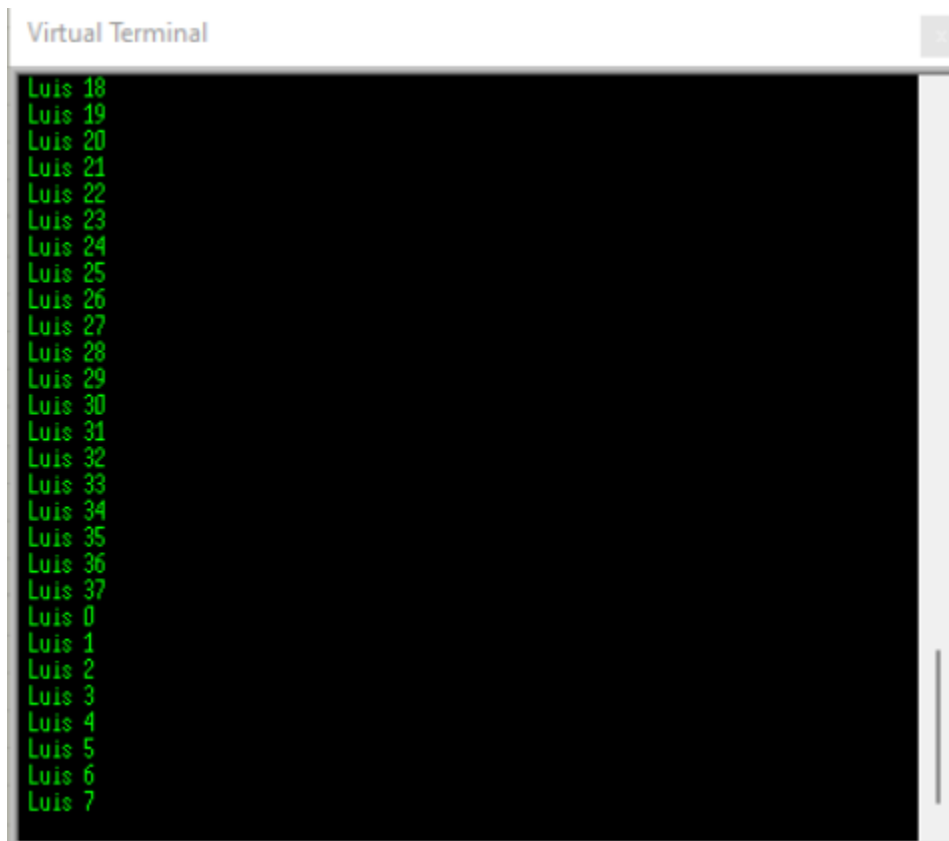
De nuevo siento el inconveniente.

Apartado 4:

Los resultados arrojados tras las pruebas son satisfactorios, el contador se resetea al ser pulsada la barra espaciadora sin problemas y no afecta al funcionamiento total del programa.



Al igual que en los anteriores apartados le dejo una imagen ampliada del Terminal Virtual:



The image shows a screenshot of a 'Virtual Terminal' window. The window has a title bar with the text 'Virtual Terminal' and a close button. The main area is a black terminal with green text. The text consists of a list of names followed by numbers, starting with 'Luis 18' and ending with 'Luis 7'. The list is as follows:

```
Luis 18
Luis 19
Luis 20
Luis 21
Luis 22
Luis 23
Luis 24
Luis 25
Luis 26
Luis 27
Luis 28
Luis 29
Luis 30
Luis 31
Luis 32
Luis 33
Luis 34
Luis 35
Luis 36
Luis 37
Luis 0
Luis 1
Luis 2
Luis 3
Luis 4
Luis 5
Luis 6
Luis 7
```

### Conclusiones:

Estoy medianamente satisfecho con el trabajo realizado.

Los resultados son correctos y funcionan según se pedía en el enunciado de la práctica.

Respecto al código creo que podría ser optimizado enormemente, en especial la forma en la que se comprueba el estado de la FIFO.

Aún con todo, el resultado general creo que es lo suficientemente bueno como para considerarse un programa de mediana calidad.

### Notas

Frecuencia de reloj elegida: 4Mhz

Baud Rate elegido: 9600