UNIVERSIDAD NEBRIJA

MEMORIA PRACTICA 3
SISTEMAS EMPOTRADOS

José María Fernández Gómez
Pablo Rayón Zapater
Fernando Pérez Ballesteros

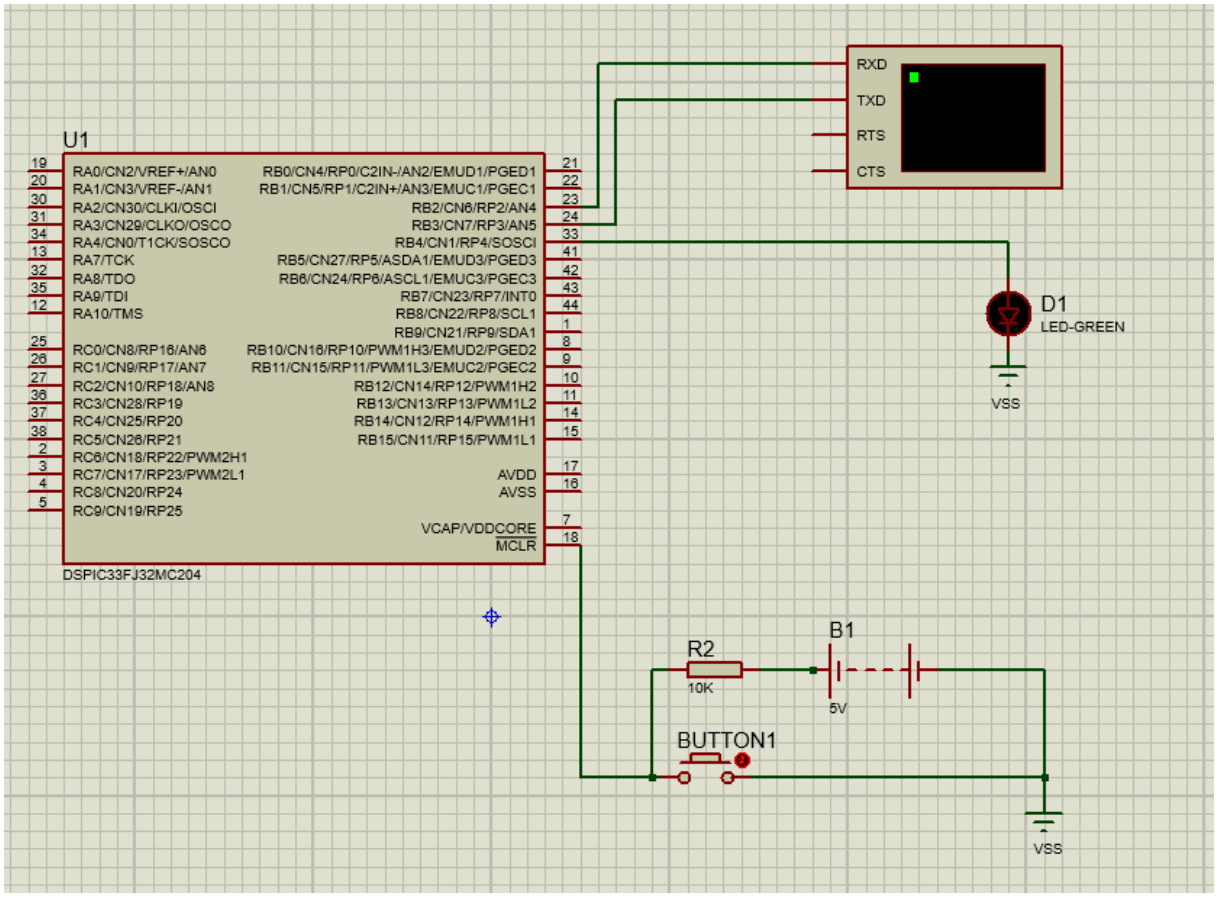
OBJETIVOS

En esta práctica se va a desarrollar un sistema que trate de ejecutar dos funciones por medio del uso de las interrupciones propias del microcontrolador DSPIC33FJ32MC204.

Se va a tratar las interrupciones del modulo UART para la recepción y envío de datos de manera no persistente, y por otro lado se tratará las interrupciones de un Timer, empleado para ejecutar una tarea a intervalos de tiempo regulados, esta pudiendo ser activada o desactivada por medio del input del usuario.

CONFIGURACION HARDWARE

A continuación, se muestra la configuración hardware empleada por el sistema.



Se han configurado los pines RB3 Y RB2 como pines de recepción y transmisión del módulo UART respectivamente.

El pin RB5 está conectado a un diodo LED, el cual parpadeará cuando el usuario lo indique (Requerido por el punto 2).

En el pin 18 se ha configurado un circuito de reinicio, cuando el botón es presionado la corriente deja de fluir hacia el microcontrolador, haciendo así que se restee.

CONFIGURACION SOFTWARE

```
#define baud_9600  1041

void delay_ms(unsigned long time_ms) {
    unsigned long u;
    for (u = 0; u < time_ms * 200; u++) {
        asm("NOP");
    }
}

void Timer1_Config(void) {
    T1CONbits.TON = 0;
    T1CONbits.TSIDL = 0;
    T1CONbits.TCKPS = 1; //Prescaler 256
    T1CONbits.TCS = 0; //Frecuencia de CPU como base de tiempos del Timer
    T1CONbits.TSYNC = 0;

    //Interrupciones
    IPC0bits.T1IP = 7;
    IFS0bits.T1IF = 0;
    IEC0bits.T1IE = 0;


    PR1 = 25000;
    TMR1 = 0;

    T1CONbits.TON = 1;
}
```

Aquí se muestra la primera parte de la configuración por medio de código, primero se declara una macro global que contendrá el valor del UxBRG, necesario para calcular la velocidad de transmisión del modulo UART, este calculo también depende de la propia frecuencia del microcontrolador. Después se realiza la función de configuración del contador que será empleado en el punto número 2, se declara el prescaler como 1 para que no haya desfase entre los ciclos de reloj acontecidos y la función que detona este contador. Por otro lado se declaran los registros de interrupción de este contador, asignándole una prioridad 7, ya que se requiere que sea del grado más alto, y se inicializan los valores del “enable” y “flag” a 0, para que así esta interrupción comience desactivada. Por último se ha configurado el registro PR1 como 25000, ya que esto indicará el semiperiodo en nanosegundos que deseamos que este timer controle, en este caso 50 ms o 50000 nanosegundos.

CONFIGURACION SOFTWARE

```
void uart_config(unsigned int baud
    //Interface uart (tx/rx)
    TRISBbits.TRISB3 = 1;
    TRISBbits.TRISB2 = 0;
    RPOR1bits.RP2R = 3; //U1TX con
    RPINR18bits.U1RXR = 3; //U1RX

    //mode
    U1MODEbits.UARTEN = 0;
    U1MODEbits.USIDL = 0;
    U1MODEbits.IREN = 0;
    U1MODEbits.RTSMD = 1;
    U1MODEbits.UEN = 0;
    U1MODEbits.WAKE = 0;
    U1MODEbits.LPBACK = 0;
    U1MODEbits.ABAUD = 0;
    U1MODEbits.URXINV = 0;
    U1MODEbits.BRGH = 1;
    U1MODEbits.PDSEL = 0;
    U1MODEbits.STSEL = 0;
    //status
    U1STAbits.UTXISEL0 = 0;
    U1STAbits.UTXISEL1 = 0;
    U1STAbits.URXISEL = 0;
    U1STAbits.UTXINV = 0;
    U1STAbits.UTXBRK = 0;
    U1STAbits.UTXEN = 1;
    U1STAbits.ADDEN = 0;
    U1STAbits.OERR = 0;

    //baudios
    U1BRG = baud;

    IPC2bits.U1RXIP = 5;
    IFS0bits.U1RXIF = 0;
    IEC0bits.U1RXIE = 1;

    // Tx
    IPC3bits.U1TXIP = 5;
    IFS0bits.U1TXIF = 0;
    IEC0bits.U1TXIE = 0;

    U1MODEbits.UARTEN = 1;
}
```

A la izquierda se muestra la configuración del módulo UART, en primer lugar, se configuran los pines de recepción y de transmisión de dicho módulo, esos siendo el RB3 para la recepción y el RB2 para la transmisión.

Mas abajo se le asigna al U1BRG la macro declarada al comienzo, este registro se encargará de calcular la velocidad de transmisión/recepción deseada.

Por último se configuran los registros de interrupciones de este módulo, a ambos se le asigna la prioridad numero 5, para que se encuentren por debajo del hilo de alta prioridad previamente declarado, pero presentan una diferencia a la hora de comenzar activados/desactivados, el registro de enable de interrupción de recepción se configura originalmente activado ya que es necesario “permanecer a ala escucha” de posibles transmisiones entrantes, por lo que la interrupción siempre estará activada, que no en uso, ya que solo saltará cuando el buffer de recepción de la UART no esté vacío.

El registro de enable de la interrupción de transmisión se encuentra originalmente desactivado para poder ser activado únicamente en el momento que el microcontrolador tenga una transmisión saliente preparada.

VECTORES DE INTERRUPCIÓN

```
void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void) {
    LATBbits.LATB4 = !PORTBbits.RB4;
    IFS0bits.T1IF = 0;
}

void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void) {
    txbuffer[0] = U1RXREG;
    IFS0bits.U1RXIF = 0;
}

void __attribute__((__interrupt__, no_auto_psv)) _U1TXInterrupt(void)
{
    IEC0bits.U1TXIE = 0;

    if (!U1STAbits.UTXBF)
    {
        U1TXREG = countbuffer[nextchar++];
        asm("nop");
        if (U1STAbits.UTXBF)
        {
            IFS0bits.U1TXIF = 0;
        }
    } else IFS0bits.U1TXIF = 0;

    if (nextchar == strlen(countbuffer))
    {
        BufferLoadDone = 1;
    } else IEC0bits.U1TXIE = 1; // Enable UART1 Tx Interrupt

}
```

En la imagen superior se muestra la funcionalidad de los vectores de interrupción, los cuales saltarán cuando su registro de activación esté activado y su flag también se encuentre activada.

En el primer caso el vector de interrupción del Timer no se encuentra activado originalmente, pero su flag se activa y se desactiva periódicamente cada 50 ms, para que esta interrupción salte es necesario activar el registro de enable de esta, cosa que se realiza en la función main tras comprobar un input de parte del usuario en concreto, como se explica en el siguiente punto, la función que realiza este vector es cambiar el estado del pin RB4, conectado al LED, generando así un efecto de parpadeo cada 50 ms cuando la interrupción es activada, acto seguido procede a desactivar su propia flag para que el efecto del contador pueda volver a comenzar y se salga de esta interrupción.

El segundo vector corresponde al registro de recepción de la UART, el cual es cargado en un buffer de recepción para poder extraer los ratos que se están recibiendo, y acto seguido apaga la flag que hace que esta misma interrupción salte, la flag es activada siempre que haya algún dato en el buffer de recepción del módulo UART.

Por último, el vector de interrupción de transmisión del módulo UART, se encarga de cargar en el registro de transmisión de dicho módulo el contenido de un array que hemos definido previamente, siempre que el buffer de transmisión no esté lleno, en caso de que lo esté , se apagará la flag de este vector para evitar que se carguen más datos. Por último se declara un flag BufferLoadDone el cual indicará si se han transmitidio todos los caractere del array que estamos intentando enviar, si no es así el enable de este vector se mantendrá activado para así terminar de enviar todos los caracteres de dicho array.

FUNCION MAIN

```
while (1) {
    if (txbuffer[0] == 'P'){
        txbuffer[0] = '\0';
        IEC0bits.T1IE = !IEC0bits.T1IE;
    }
    if (txbuffer[0] == ' ') {
        if (count == 0)count = 1;
        else count = 0;
        txbuffer[0] = '\0';
    }
    if (count == 1 && BufferLoadDone == 1 && (U1STAbits.TRMT)) {

            memset(countbuffer, '\0', sizeof (countbuffer));
            sprintf(countbuffer, "Contador: %d \r\n", contador++);
            nextchar = 0;
            BufferLoadDone = 0;
            if (U1STAbits.UTXBF) IFS0bits.U1TXIF = 0;
            asm("nop");
            IEC0bits.U1TXIE = 1;

        delay_ms(10);
    }
    delay_ms(100);
```

El flujo principal del programa comienza comprobando los caracteres que el buffer de recepción contiene, en caso de que sea una letra “P” primero eliminará esta letra para que no entre en buble el programa, y después alternará el estado de la flag de activación del contador, haciendo así que cada vez que se reciba dicha letra se active o se desactive el vector del Timer que dará lugar al parpadeo del LED. En caso de que el carácter recibido sea un espacio, modificaremos el valor de una flag declarada por nosotros llamada count, la cual se unirá a una serie de comprobaciones que se realizarán mas abajo, también es necesario extraer ese carácter del buffer una vez comprobado su valor para que nos se produzca un bucle.

Por último tenemos una comprobación de tres valores diferentes, los cuales se encargan de comprobar si hay que enviar algo (count) si la transmisión pasada ha cargado todos lo caracteres al buffer de transmisión (BufferLoadDone) y si existe una conexión abierta, en caso de que no exista (1) quiere decir que la comunicación previa se ha completado (TRMT). Después de realizar estas comprobaciones se carga el buffer con los datos a enviar (Contador : n_contador), se resetea el numero de caracteres que se han enviado, para que la comunicación comience desde la posición 0 del array y se indica que no se han cargado todos lo caracteres en el buffer de transmisión, para que luego este valor vuelva a ser actualizado dentro del vector de interrupciones de tranmisión, también se comprueba si el buffer de transmisión esta lleno por evitar errores.