

SISTEMAS EMPOTRADOS

PRACTICA 2.

Ignacio Ortega y Rocío Ruiz

Universidad Antonio de Nebrija

ÍNDICE

<i>Practica 2</i>	2
Enunciado.....	2
RxInterrupt de MIC1 y MIC2.....	3
TxInterrupt de MIC1 y MIC2.....	4
Programa principal MIC1.....	5
Programa principal MIC2.....	6
SISTEMA COMPLETO: PROTEUS.....	8

Mar, 27 oct. 2020







PRACTICA 2

En esta práctica hemos aprendido a realizar comunicaciones entre dos microcontroladores a través del puerto UART. Para ello, empleamos las interrupciones tanto para transmitir como para recibir datos correspondientes a la UART.

Enunciado

El primer microcontrolador (MIC1), es el máster. Se encarga de recibir los datos del usuario. Si corresponde a un comando preestablecido, manda al segundo microcontrolador la traducción de ese comando para que lo pueda interpretar.

El segundo microcontrolador (MIC2), es el esclavo (Slave). Se encarga de recibir los comandos ya traducidos por el MIC1 y realizar la acción correspondiente.

USERS (VT1) -> MIC1	MIC1-MIC2	ACCIÓN MIC2
set ledgreen 0	0x50, 0x00, 0xAA	
set ledgreen 1	0x50, 0x01, 0xAA	
set ledred 0	0x51, 0x00, 0xAA	
set ledred 1	0x51, 0x01, 0xAA	
stop MIC2_data	0x52, 0x00, 0xAA	
print MIC2_data	0x52, 0x01, 0xAA	

El comando "print MIC2_data" incluye un contador que va aumentando cada vez que se imprime. Si este contador llega a 99 antes de que el usuario haya pedido parar la impresión de datos, se reiniciará a cero. Se pide que se impriman los datos a 5 Hz, es decir, 5 veces por segundo.

Lo primero que se realizó fue gestionar la interrupción de recepción de datos de la UART de ambos microcontroladores.

RxInterrupt de MIC1 y MIC2

La base de esta interrupción nos vale para ambos microcontroladores ya que ambos reciben caracteres y comprueban si corresponden con alguno de los comandos predeterminados.

Cuando se reciben caracteres por el registro de recepción de la UART, salta esta interrupción. Lo que queremos hacer es guardar estos caracteres en un buffer para ver si en algún momento, lo almacenado, corresponde a un comando. El microcontrolador dos, solo necesita esta base ya que el microcontrolador 1 ya le envía el comando entero definido. Se implementaría de la siguiente manera:

```
void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void)
{
    dataCMD_ISR[data_count] = U1RXREG; // Obtener caracter recibido en el buffer
    data_count++;

    IFS0bits.U1RXIF = 0; // Reset Rx Interrupt
}
```

En cambio, en el microcontrolador 1 hay que definir algunas reglas más para detectar cuando un usuario envía un comando, y en ese caso, comprobar si es correcto.

REGLA: Si el carácter corresponde con la tecla ENTER (13 en ASCII), lo tomamos como que se ha detectado un comando. En este caso, en el programa principal, comprobaríamos si es correcto. Para ello, necesitamos una variable comando_detectado que se active cuando se pulse el ENTER. En este caso, también borraríamos el carácter del ENTER de el buffer ('\0').

```
void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void)
{
    if(comando_detectado == 0)
    {
        dataCMD_ISR[data_count] = U1RXREG; // Obtener caracter recibido en el buffer
        data_count++;

        if( (dataCMD_ISR[data_count - 1] == 13) ) // Retorno de carro-Intro detectado
        {
            dataCMD_ISR[data_count - 1] = '\0'; // Eliminar retorno de carro del stream recibido .
            comando_detectado = 1;
            data_count = 0;
        }
    }
    else
    {
        dummy = U1RXREG;
    }

    IFS0bits.U1RXIF = 0; // Reset Rx Interrupt
}
```

TxInterrupt de MIC1 y MIC2

Ambos microcontroladores tienen una interrupción de transmisión de la UART bastante similar. Lo primero que se hace es desactivar la interrupción (Registro UTXIE). Mientras que el buffer de transmisión no se encuentre completo, seguimos enviando datos (a través de registro de transmisión UTXREG). Si este buffer se llena, hay que desactivar el flag de la interrupción para no perder datos.

```
void __attribute__((__interrupt__, no_auto_psv)) _UTXInterrupt(void) // Actualmente configurado para U1STAbits.UTXI
{
    IECObits.UTXIE = 0; // Disable UART1 Tx Interrupt

    if(!U1STAbits.UTXBF) // Mientras el buffer de transmisión NO se encuentre completo, continuar carg
    {
        UTXREG = txbuffer_ISR[nextchar++]; // Cargar el buffer con un nuevo dato.
        asm ("nop");
        if(U1STAbits.UTXBF) // Si el buffer de transmisión se completó con el último dato incorporado al
        {
            IFSObits.UTXIF = 0; // Clear UART1 Tx Interrupt Flag
        }
    }
    else IFSObits.UTXIF = 0; // Clear UART1 Tx Interrupt Flag
}
```

El microcontrolador 1, mientras siga habiendo datos que transmitir, vuelve a activar la interrupción.

```
if(!(nextchar == strlen(txbuffer_ISR)) // Si se ha finalizado la transmisión de todos los caracteres --> Deshabilita
{
    IECObits.UTXIE = 1; // Enable UART1 Tx Interrupt
}
}
```

El microcontrolador 2 también, cuando ha completado de enviar todo activa la variable BufferLoadDone. Esta variable global es inicializada a 1 para que la primera transmisión funcione.

```
if(nextchar == strlen(txbuffer_ISR)) // Si se ha finalizado la transmisión de todos los caracteres --> Deshabilita
{
    BufferLoadDone = 1; // Informamos de que se ha terminado de cargar la cadena de texto de 'U1_TxBu
}
else IECObits.UTXIE = 1; // Enable UART1 Tx Interrupt
}
```

Programa principal MIC1

Si la interrupción de recepción de la UART recibió un ENTER la variable comando_detectado está activada. Si está activada, hay que comprobar si coincide con algún comando predeterminado.

```
const char cmd1[] = {"set ledgreen 1"}; // Comando para encender led verde
const char cmd2[] = {"set ledgreen 0"}; // Comando para apagar led verde
const char cmd3[] = {"set ledred 1"};   // Comando para encender led rojo
const char cmd4[] = {"set ledred 0"};   // Comando para encender led rojo
const char cmd5[] = {"print MIC2_data"}; // Comando para imprimir datos
const char cmd6[] = {"stop MIC2_data"};  // Comando para parar de imprimir
```

Si coincide, vaciamos el buffer de su anterior tarea y le llenamos con el comando que queremos pasarle al microcontrolador 2 para que lo reconozca y ejecute la acción pedida por el usuario. También hay que poner a cero la variable que indica por qué elemento empieza a leer la interrupción de transmisión (nextchar). Antes de llamar a la interrupción hay que asegurarse de que el buffer de transmisión no se encuentre lleno, y sino, habría que desactivar el flag de transmisión. Y a continuación, llamamos a la interrupción de transmisión que se ocupará de mandarle el nuevo comando al microcontrolador 2.

```
while(1)
{
    if(comando_detectado)
    {
        if (!strcmp(((const char*)dataCMD_ISR), cmd1) ) {
            memset(txbuffer_ISR, '\0', sizeof(txbuffer_ISR)); // Resetear buffer con NULL
            sprintf(txbuffer_ISR, "%c%c%c", 0x50, 0x01, 0xAA);
            nextchar = 0;
            if(U1STAbits.UTXBF) IFS0bits.U1TXIF = 0;          // Reseteo el flag de transmision ISR
            asm("nop");
            IEC0bits.U1TXIE = 1;                               // Iniciamos una nueva transmisión
        }
    }
}
```

Después de ejecutar la interrupción, el hilo de ejecución vuelve al programa principal donde se reinician las variables cambiadas anteriormente. Es decir, ya no tenemos ningún comando detectado, la cuenta de datos vuelve a cero y se reinicia el buffer donde almacenábamos lo que nos va transmitiendo el usuario.

```
    else{
        /*code here*/
    }

    memset(dataCMD_ISR, '\0', sizeof(dataCMD_ISR)); // Resetear buffer con NULL
    data_count = 0;
    comando_detectado = 0;
}
else
{
    /*code here*/
}
delay_ms(100);
}
```

Programa principal MIC2

El microcontrolador 2 comprueba si lo que le llega del microcontrolador 1 es uno de sus comandos predeterminados.

```
const char cmd1[50] = {0x50, 0x01, 0xAA}; // Comando para encender led verde
const char cmd2[50] = {0x50, 0x00, 0xAA}; // Comando para apagar led verde
const char cmd3[50] = {0x51, 0x01, 0xAA}; // Comando para encender led rojo
const char cmd4[50] = {0x51, 0x00, 0xAA}; // Comando para apagar led rojo
const char cmd5[50] = {0x52, 0x01, 0xAA}; // Comando para imprimir datos
const char cmd6[50] = {0x52, 0x00, 0xAA}; // Comando para parar de imprimir dato
```

Esto lo comprobamos con la función strcmp(). Si es así, ejecuta la acción que implica ese comando. Las acciones que se hacen directamente son las que implican encender o apagar un LED (cmd[1-4]). Estas son configuradas en el inicio del programa principal.

```
TRISAbits.TRISA0 = 0;
TRISBbits.TRISB3 = 0;
LATAbits.LATA0   = 0;
LATBbits.LATB3   = 0;
```

Los comandos que implican imprimir datos o dejar de imprimir lo que hacen es activar o desactivar la variable que les da ese permiso: Allowprint.

Una vez tomadas estas acciones se reinician los datos cambiados como hacíamos en el microcontrolador 1.

```
while(1)
{
    if ( !strcmp(((const char*)dataCMD_ISR), cmd1) ) LATAbits.LATA0 = 1;
    if ( !strcmp(((const char*)dataCMD_ISR), cmd2) ) LATAbits.LATA0 = 0;
    if ( !strcmp(((const char*)dataCMD_ISR), cmd3) ) LATBbits.LATB3 = 1;
    if ( !strcmp(((const char*)dataCMD_ISR), cmd4) ) LATBbits.LATB3 = 0;
    if ( !strcmp(((const char*)dataCMD_ISR), cmd5) ) Allowprint = 1;
    if ( !strcmp(((const char*)dataCMD_ISR), cmd6) ) Allowprint = 0;

    else
    {
        /*code here*/
    }

    memset(dataCMD_ISR, '\0', sizeof(dataCMD_ISR)); // Resetear buffer con NULL
    data_count = 0;
}
```

Para ejecutar las acciones que implican transmisión de la UART comprobamos que la variable Allowprint está activada, que todos los datos han sido cargados en el buffer y que la última transmisión ha sido finalizada.

Queremos transmitir estos datos 5 veces por segundo. Y el programa principal termina con un delay de 10 ms. Por lo tanto, queremos que cada 20 veces que haga el delay de 10 ms, imprima un dato. ($20\text{ms} \times 10\text{ms} = 200\text{ms} = (1000\text{ms}/5\text{veces})$)

Para ello, nos ayudamos con la variable auxiliar U2_PrintRate_ISR. Cada vez que esta llegue a 20, se imprimirá un dato.

```

if((Allowprint)&&(BufferLoadDone)&&(U1STAbits.TXMT)) // Si se cargaron todos los datos al buffer U1TXREG y finalizó la última transmisión...
{
    if(U1_PrintRate_ISR++ >= 20) // Determinar el ritmo de transmisión de datos de la Uart1 Tx. 20 veces el delay de 10 ms
    {
        if(contador == 100){
            contador = 0;
        }

        memset(txbuffer_ISR, '\0', sizeof(txbuffer_ISR)); // Clear Buffer and fill it with NULL
        sprintf(txbuffer_ISR, "IMPRIMIENDO DATOS: %d \r\n", contador++);

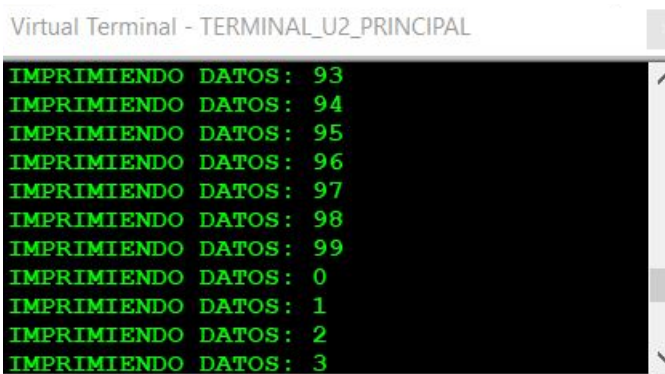
        nextchar = 0;
        BufferLoadDone = 0;
        U1_PrintRate_ISR = 0;

        if(U1STAbits.UTXBF && IFS0bits.UTXIF == 0; // Reseteo el flag de transmisión ISR
        asm("nop");
        IEC0bits.U1TXIE = 1; // Iniciamos una nueva transmisión
    }
}

delay_ms(10);
}

```

Además, si el contador alcanza el valor de 100 antes de que el usuario pare la transmisión, se reiniciará el contador a 0 antes de enviar el dato a la terminal virtual.



Si Allowprint se vuelve a poner a 0, no se entrará en esta condición y, por lo tanto, se dejarán de imprimir datos. Si se volviera a activar, el contador continuaría por donde se quedó la última vez.

SISTEMA COMPLETO: PROTEUS

El microcontrolador 1 conecta su pin de recepción de UART a el pin de transmisión de la terminal virtual donde el usuario va a escribir los comandos. Hay otra virtual terminal ECHO que se encarga de mostrar por pantalla lo que el usuario va metiendo, ya que sino, no se vería nada.

El pin de transmisión lo conecta al pin de recepción del microcontrolador 2 para pasarle la traducción de los comandos. El pin de transmisión del microcontrolador 2 va conectado al pin de recepción de terminal virtual, para mostrar los datos que va recibiendo. El microcontrolador 2 tiene además 2 LEDs en los pines que configuramos en el programa principal del microcontrolador 2.

Pusimos una terminal 2 ECHO conectada a la transmisión del MIC 1 para ver si enviaba los comandos. Nos pareció curioso dejarla ya que, aunque los comandos se envían correctamente, la terminal virtual muestra su equivalente en ASCII.

