Práctica: P4_UART_v1

27-02-2023

Alex Agustin y Amanda Sin

Dispositivos utilizados:

Leds: D3, D5, D7, D8, D9.
Pulsadores: S3, S4 y S6.
Módulos T5, T7 y T9.
Módulo oscilator.
Módulo LCD.
Módulo CN.
Módulo UART2.

Sincronización de los dispositivos (encuesta/interrupción):

Pulsador S3: se gestiona por encuesta inicialmente y posteriormente por interrupción.
Pulsador S4: se gestiona por encuesta.
Pulsador S6: se gestiona por interrupción.
Módulo T5: se gestiona por interrupción.
Módulo T7: se gestiona por interrupción.
Módulo T9: se gestiona por encuesta.
Módulo UART2: tanto el emisor como el receptor se gestionan por interrupción.

Funciones que llama el programa principal (nombre y funcionamiento)

inic_oscilator (en oscilator.c)

- Selecciona e inicializa el reloj a 80MHz, de modo que las instrucciones se ejecuten a 40 MHz.

Init_LCD (en LCD.c)

- Inicializa el módulo *LCD*. Se envía una secuencia de comandos para realizar la inicialización, además de las esperas correspondientes para el procesamiento de cada comando. Esta función se nos ha dado hecha y únicamente hemos añadido las esperas de 40 us.

copiar_FLASH_RAM (en memoria.c)

- Copia una línea de texto de *FLASH* a *RAM*, en una línea determinada de la variable *Ventana_LCD* (la cual contiene el texto a mostrar en pantalla).

line_1 (en LCD.c)

- Posiciona el cursor en la primera línea de la pantalla.

line_2 (en LCD.c)

- Posiciona el cursor en la segunda línea de la pantalla.

puts_lcd(en LCD.c)

- Proyecta en la pantalla un número de caracteres, ambos dados como entrada (número de caracteres y dirección de comienzo de la variable en la que se almacenan los caracteres).

inic_pulsadores (en GPIO.c)

- Define los pines analógicos AN16-AN31 como digitales. El pin *RA7* (pulsador S5/led D10) es el AN23: hay que definirlo como digital si se usa alguno de esos dispositivos (no es el caso en este proyecto).
- Inicializa los pines de los pulsadores **S3** (*RD6*), **S4** (*RD13*) y **S6** (*RD7*) como entrada (*TRIS*=1).

inic_crono (en timers.c)
- Inicializa a 0 las variables del cronómetro correspondientes a milisegundos (mili), décimas de segundo (deci), segundos (seg) y minutos (min).

inic_CN(en CN.c)
- Inicializa el módulo *CN* para que los pulsadores **S3** (pin *CN15*) y **S6** (pin *CN16*) se gestionen por interrupción. Para ello, se habilita tanto la interrupción general de *CN* como la interrupción de los pines correspondientes a los pulsadores.

inic_leds (en GPIO.c)
- Inicializa los pines de todos los leds como salida (*TRIS*=0).

inic_Timer7(en timers.c)
- Inicializa el módulo T7 con un periodo de 10 milisegundos y prescaler 01 (escala 1:8), puesto que la medida de ciclos a esperar es mayor que 65535 (2^16, el tamaño del registro del temporizador).

inic_UART2 (en UART2_RS232.c)
- Inicializamos el módulo **UART2** con el **BRG** correspondiente: High1, usado para velocidades altas. Este se calcula así: ((Fcy/BAUDRATE2)/4)-1. Cabe destacar que para que el módulo **UART** tenga el comportamiento esperado, ha de inicializarse igual en los dos extremos (no estamos usando paridad, 1 bit de stop, misma velocidad,...). Además, hemos habilitado las interrupciones tanto del emisor como del receptor del módulo **UART2**.

inic_Timer5 (en timers.c)
- Inicializa el módulo **T5**, con un periodo de 2.5 milisegundos y prescaler 01 (escala 1:8), puesto que la medida de ciclos a esperar es mayor que 65535 (2^16, el tamaño del registro del temporizador).

cronometro(en timers.c)
- Cuando el flag que indica que se ha de inicializar el cronómetro está activado (*inicializar_crono*), inicializa el cronómetro mediante la función *inic_crono*. Además, guarda en la variable *Ventana_LCD* los nuevos valores (0) de las variables utilizadas para contabilizar el tiempo para su proyección en la pantalla y apaga los LEDs. Finalmente, pone a 0 el flag *inicializar_crono*.
- Actualiza, cuando corresponde, las variables que mantienen constancia del tiempo transcurrido desde que se ha inicializado el cronómetro. De este modo, modifica (incrementar en uno o restar la cantidad de unidades correspondiente) según corresponda los milisegundos, décimas de segundo (cada 100 milisegundos), segundos (cada 10 décimas de segundo) y minutos (cada 60 segundos). Además, guarda en la variable *Ventana_LCD* dichos valores para su proyección en la pantalla. Adicionalmente, se conmutan los siguientes LEDs cuando pasa el tiempo correspondiente:
  - LED D3 -> se conmuta cada décima de segundo.
  - LED D5 -> se conmuta cada segundo.
  - LED D9 -> se conmuta cada minuto.

Otras funciones importantes (no se les llama desde el main)

Delay_ms (en timers.c)

- Calcula el número de ciclos correspondiente a los milisegundos especificados por parámetro.
- Si el número de ciclos es válido, inicializa el módulo **T9** con dicho valor. Tras esto, espera (por encuesta) a que se active el flag *IF* del módulo **T9**, indicando que ha transcurrido el tiempo especificado. Finalmente, marca la interrupción como atendida y se apaga el temporizador.
- Si el número de ciclos es superior al valor máximo que puede aceptar el temporizador con el mayor prescaler, se conmuta el **LED D7** y la función permanece en un *while* infinito.

Delay_us (en timers.c)

- Calcula el número de ciclos correspondiente a los microsegundos especificados por parámetro.
- Si el número de ciclos es válido, inicializa el módulo **T9** con dicho valor. Tras esto, espera (por encuesta) a que se active el flag *IF* del módulo **T9**, indicando que ha transcurrido el tiempo especificado. Finalmente, marca la interrupción como atendida y se apaga el temporizador.
- Si el número de ciclos es superior al valor máximo que puede aceptar el temporizador con el mayor prescaler, se conmuta el **LED D8** y la función permanece en un *while* infinito. Cabe destacar que dado que el valor máximo que puede almacenar el registro de entrada de la función (es un **unsigned int**) es menor al que podría permitirnos el prescaler, esta situación sería inalcanzable.

inic_Timer9 (en timers.c)

- Inicializa el módulo **T9**, con un periodo y prescaler acordes al número de ciclos especificado por parámetro.

conversion_tiempo (en utilidades.c)

- Obtiene los caracteres de un valor de dos dígitos (<100) y los guarda a partir de una dirección proporcionada.

lcd_cmd (en LCD.c)

- Es una función brindada por el profesorado que se encarga de enviar un comando a la pantalla. El único cambio realizado es la adición de 10 Nop()'s para asegurar que los datos estén estables.

lcd_data (en LCD.c)

- Es una función brindada por el profesorado que se encarga de enviar un dato a la pantalla. El único cambio realizado es la adición de 10 Nop()'s para asegurar que los datos estén estables.

Rutinas de atención

Rutina módulo CN (en CN.c): _CNInterrupt()

- Comprueba los pulsadores S3 y S6 (los habilitados para interrumpir) y si están pulsados se realiza la acción correspondiente:
  - Pulsador S3 (RD6) -> puesta en marcha y detención del cronómetro.
  - Pulsador S6 (RD7) -> inicialización del cronómetro (puesta a 0) mediante la activación del flag *inicializar_crono*.

Rutina módulo T7 (en timers.c): _T7Interrupt()

- Suma 10 milésimas de segundo a la variable correspondiente a los milisegundos transcurridos (*mili*).

Rutina módulo T5 (en timers.c): _T5Interrupt()

- Realiza las acciones pertinentes según el estado de un autómata en lo referente a la proyección de texto en la pantalla LCD:
  - Estado LCD_LINE1 (0) -> posicionamiento del cursor en la primera línea.
  - Estado LCD_DATA1 (1) -> envío de los datos de la primera línea.
  - Estado LCD_LINE2 (2) -> posicionamiento del cursor en la segunda línea.
  - Estado LCD_DATA2 (3) -> envío de los datos de la segunda línea.

Rutina módulo U2RX (en UART2_RS232.c): *_U2RXInterrupt*()

- Realiza las acciones correspondientes al carácter recibido en el registro **U2RXREG**, además de guardarlo en la variable *Ventana_LCD* para su proyección (posición 15 de la segunda línea, esquina inferior derecha).
  - 'P' / 'p' -> parar el cronómetro.
  - 'C'/'c' -> poner en marcha el cronómetro.
  - 'I' /'i' -> inicialización del cronómetro (puesta a 0) mediante la activación del flag *inicializar_crono*.
  - Cualquier otro carácter -> ninguna acción asociada.

Rutina módulo U2TX (en UART2_RS232.c): *_U2TXInterrupt*()

- Refresca el contenido de *Ventana_LCD* también en la pantalla del PC a través de un autómata:
  - Estado UART_HOME (0) -> posicionamiento al principio de la pantalla.
  - Estado UART_DATA1 (1) -> envío de los datos de la primera línea.
  - Estado UART_NEXTLINE (2) -> salto de línea y posicionamiento al principio de la línea, en dos tiempos.
  - Estado UART_DATA2 (3) -> envío de los datos de la segunda línea.

Funcionamiento general (verificado en las pruebas)

El programa principal realiza las inicializaciones del oscilador, LCD y pulsadores mediante las funciones descritas anteriormente (en ese orden). Tras esto, copia a memoria RAM las primeras líneas a mostrar en la pantalla y las proyecta en la misma tras posicionarse en la línea correspondiente. A continuación, espera por encuesta a que se pulse **S3** (RD6). Una vez pulsado, copia a memoria RAM el segundo par de líneas a mostrar en la pantalla y las proyecta en la misma tras posicionarse en la línea correspondiente.

Tras esto, espera por encuesta a que se pulse **S4** (RD13). Una vez pulsado, copia a memoria RAM el tercer par de líneas a mostrar en pantalla y las proyecta en la misma tras posicionarse en la línea correspondiente. Es en este punto cuando realiza las inicializaciones del cronómetro, del módulo **CN**, de los **LED**s, del temporizador **T7,** del módulo **UART2** y del temporizador **T5** mediante las funciones descritas anteriormente (en ese orden). Tras la ejecución de la función *inic_CN* (y no antes), el tratamiento de los pulsadores **S3** y **S6** se realizará por interrupción. Además, en este momento se ha dado el valor 'Z' a ***U2TXREG*** para desencadenar las interrupciones del emisor del módulo ***UART2***. Se ha de enviar algo inicialmente para provocar la primera interrupción.

Cabe destacar que a partir de este momento (tras *inic_Timer5*) la actualización de la pantalla se realizará mediante refresco distribuido. A continuación, se dará un bucle infinito durante el cual se ejecutará la función *cronometro* para contabilizar y visualizar (mediante LEDs, el módulo LCD y la pantalla del PC a través del **emisor** del módulo **UART2**) el tiempo transcurrido desde que se ha activado el cronómetro, además de que se tratarán las interrupciones que puedan darse mediante los pulsadores mencionados o teclas concretas. De este modo, en cualquier momento en que se pulse uno de dichos pulsadores se realizarán las acciones correspondientes (ver *_CNInterrupt*). Adicionalmente, se realizarán las acciones pertinentes en caso de pulsar teclas específicas en el PC debido al **receptor** del módulo **UART2** (ver *_U2RXInterrupt*). Además, se refrescará el contenido de *Ventana_LCD* también en la pantalla del PC gracias al **emisor** del módulo **UART2** (ver *_U2TXInterrupt*).

Conclusiones obtenidas en relación con el código escrito y las pruebas relacionadas

En las pruebas de ejecución, se ha comprobado la existencia de los rebotes en el pulsador **S3** al tratar de poner en marcha / detener el cronómetro. Es decir, a veces la activación de dicho pulsador se detecta más de una vez, y por tanto se detiene/pone en marcha el cronómetro más de una vez, con lo que es posible que parezca que no ha sucedido nada (cantidad par de cambios en la señal). Consecuentemente, hemos mantenido las variables globales explicadas en las prácticas previas para las comprobaciones relacionadas con dichos rebotes. Es decir, las variables asociadas a cada pulsador tratado por interrupción, las cuales se inicializan a 0 y cuyo valor se incrementa cada vez que se detecta un cambio en la señal correspondiente.

En lo referente al emisor del módulo ***UART2***, hemos visto que siendo que tenemos 37 caracteres en total a enviar (32 datos + 5 comandos) y el refresco ha de darse cada décima de segundo, 3700b/s sería la velocidad mínima de transmisión para asegurar que se realiza el refresco a velocidad suficiente. De este modo, la velocidad óptima de entre las permitidas por *Tera Term* sería 4800 baudios. Con velocidades superiores estaríamos enviando datos en exceso, mientras que con velocidades inferiores el refresco no se daría a la velocidad suficiente y por ende se apreciarían saltos en las décimas. Hemos realizado pruebas en lo referente al envío y recepción del módulo **UART2** a diferentes velocidades en baudios: 4800, 9600 y 19200. A través de dichas comprobaciones, estableciendo cada velocidad en ambos extremos, hemos visualizado que los datos se transmitían de manera correcta en ambos sentidos.

Cabe destacar que con velocidades inferiores a 4800 baudios, como sería el caso de 2400 baudios, hemos visualizado que el refresco no se daba de manera satisfactoria. Es decir, los caracteres no se actualizaban a la velocidad suficiente y se podían apreciar saltos en las décimas. Por otro lado, al asignar diferentes velocidades en los dos extremos hemos visualizado que se transmitía basura en vez de datos válidos.