Máster Universitario en Ingeniería de Sistemas Electrónicos

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Sistemas Integrados Digitales

Ejercicio 3

Jose Luis, Rocabado Rocha

ÍNDICE

ÍNDICE DE FIGURAS

Introducción del ejercicio

Para el desarrollo del este ejercicio práctico se plantea la resolución de dos subtareas que se llevaran a cabo en el modelo verificado del sistema con el microprocesador Nios II en la placa de desarrollo DE-115. Para trabajar los conceptos teóricos desarrollados en las sesiones de teoría se pretende realizar la programación de:

- **Reloj digital**. El reloj digital nos permitirá visualizar las horas, minutos y segundos tanto en los *displays* de 7 segmentos como en la pantalla LCD. Para ello, y mediante interrupciones, se utilizará el *timer* que incorpora el sistema para obtener con precisión los cambios de tiempo.
  También mediante interrupciones, se utilizarán los pulsadores KEY3 y KEY2 para incrementar y ajustar las horas y los minutos respectivamente.
  Por último, dado que el *display* LCD posee dos filas, se utilizará la fila restante para mostrar texto adicional.
- **Visualización en la Pantalla MTL2**. Se implementará un teleprónter en el que mediante un desplazamiento vertical de abajo a arriba se visualizarán las noticias almacenadas en los documentos de texto de la memoria *flash* de la placa. El período del desplazamiento será de 2.4 segundos y además se debe de incluir la posibilidad de pausar o continuar con la secuencia de visualización mediante los interruptores de tipo *slider*.

A continuación, se describirá la programación que implementa las subtareas del ejercicio además de un pequeño manual de usuario del reloj.

Reloj digital

1. Programación

Para llevar a cabo la implementación mediante interrupciones se utilizan los ficheros “pushbutton_ISR.c” y “Interval_timer_ISR.c” que definirán las funciones de *callback* que gestionarán la interrupción. Además, para evitar problemas con la introducción de parámetros a estas funciones, se utilizarán variables globales (“msec_counter” y “key_pressed”) que se podrán acceder tanto durante la ejecución del código principal como en la gestión de la interrupción.

Para mantener lo mas simple posible y reducir el tiempo de ejecución dedicado a la gestión de la interrupción, la función “Interval_timer_ISR()” incrementará en uno la variable *msec_counter* con cada interrupción (figura 1). Cuando llegue a 10, se reiniciará y así podrá contar nuevamente otros 1000ms.

```c
#include "key_codes.h"  // define los valores de KEY1, KEY2, KEY3
#include "system.h"
#include "sys/alt_irq.h"

extern volatile int msec_counter;
extern volatile int frame_refresh;

void interval_timer_isr( )
{
    volatile int * interval_timer_ptr = (int *) TIMER_BASE;

    *(interval_timer_ptr) = 0;              // Borra la interrupci□n
    // Count up to 10 to count seconds (reset done in main "Real_Time_Clock.c")
    msec_counter++;
    // Count up to 10 to count 2.4s to refresh vga screen (reset done in main "Real_Time_Clock.c")
    frame_refresh++;
    return;
}
```

Fig. 1. Snippet del código del documento "interval_timer_isr.c" que implementa la gestión de las interrupciones del timer del sistema.

Por otro lado, la función "pushbutton_ISR()" actualizará la variable "*key_pressed"* con los valores definidos en el documento "key_codes.h" indicándonos así que botón ha sido presionado. En la siguiente figura, se observa que se comprueban los cuatro botones a pesar de que en esta tarea solo utilizamos 2. No hay ningún problema con ellos puesto que solo los botones que utilicemos serán configurados para generar interrupciones.

Además, se crean las subrutinas "int two_hex_to_seven (int two_hex)"," void print_7_seg_time (int h, int m, int s)" y "void print_LCD_text (int h, int m, int s)" para gestionar la visualización del reloj en los *displays* y la conversión de datos para el 7-segmentos en conjunto con las subrutinas facilitadas por los docentes en el ejercicio 2.

La función "two_hex_to_seven()" contiene una variable auxiliar que codifica la conversión del número de 4 bits para controlar el 7-segmentos. Tras obtener los dos dígitos se realiza su codificación por separado y finalmente se concatenan para obtener el resultado final. Esta función seria innecesaria si se realiza la instanciación de la IP desarrollada en la práctica del bloque II y únicamente sería necesario enviar los valores a codificar.

La función "print_7_seg_time()" realiza la conversión de las horas, minutos y segundos y envía los datos a los 7-segmentos correspondientes.

Por último, "print_LCD_text()" muestra el reloj en la segunda línea del *display* y en la primera línea, alternará la visualización de dos textos (nº ejercicio y nombre del alumno) cada 5 segundos.

```c
#include "key_codes.h"  // define los valores de KEY0, KEY1, KEY2, KEY3
#include "system.h"
#include "sys/alt_irq.h"

extern volatile int key_pressed;

void pushbutton_ISR( )
{
    volatile int * KEY_ptr = (int *) PUSHBUTTONS_BASE;
    int press;

    press = *(KEY_ptr + 3);                 // lee el registro de los pulsadores
    *(KEY_ptr + 3) = 0;                     // borra la interrupcion

    if (press & 0x2)                        // KEY1
        key_pressed = KEY1;
    else if (press & 0x4)                   // KEY2
        key_pressed = KEY2;
    else if (press & 0x1)                   // KEY0
        key_pressed = KEY0;
    else                                    // press & 0x8, means KEY3
        key_pressed = KEY3;
    return;
}
```

Fig. 2. Snippet del código que implementa la gestión de la interrupción de los botones de la placa de desarrollo en. "pushbutton_ISR.c".

Finalmente, y tras inicializar el reloj a las 12:00:00, podremos encontrar la gestión del reloj donde cada segundo se incrementa el contador de segundos y tras llegar a 59s reiniciará dicho contador e incrementará el de los minutos. Análogamente, ocurrirá lo mismo con el cambio de minutos a horas. Las horas se reiniciarán una vez se llegue a 23. También se tiene en cuenta los cambios producidos por los botones como se puede observar en la siguiente figura.

```
135     while(1)
136     {
137         if(msec_counter == 10)
138         {
139             msec_counter = 0;
140             sec ++;
141         }
142         if (key_pressed == KEY2)
143         {
144             key_pressed = -1;
145             min ++;
146         }
147         if (key_pressed == KEY3)
148         {
149             key_pressed = -1;
150             hour ++;
151         }
152         if (sec == 60)
153         {
154             sec = 0;
155             min ++;
156         }
157         if (min == 60)
158         {
159             min = 0;
160             hour ++;
161         }
162         if (hour == 24)
163         {
164             hour = 0;
165         }
```

Fig. 3. Código que implementa la gestión del reloj dentro del código principal.

2. Manual de usuario

Una vez empieza a ejecutarse el programa, el reloj se inicia con las 12:00:00. Cada segundo se irá actualizando tanto en el LCD como en el 7-segmentos. Para ajustar la hora al tiempo real se debe apretar el botón KEY3 para incrementar las horas y el KEY2 para cambiar los minutos. No es posible reducir estos valores, por lo que se debe ir por todo el ciclo para volver a reiniciar las horas y los minutos respectivamente. No es posible ajustar los segundos. En la siguiente figura, podemos encontrar la información de visualización y control en la placa de desarrollo DE2-115.

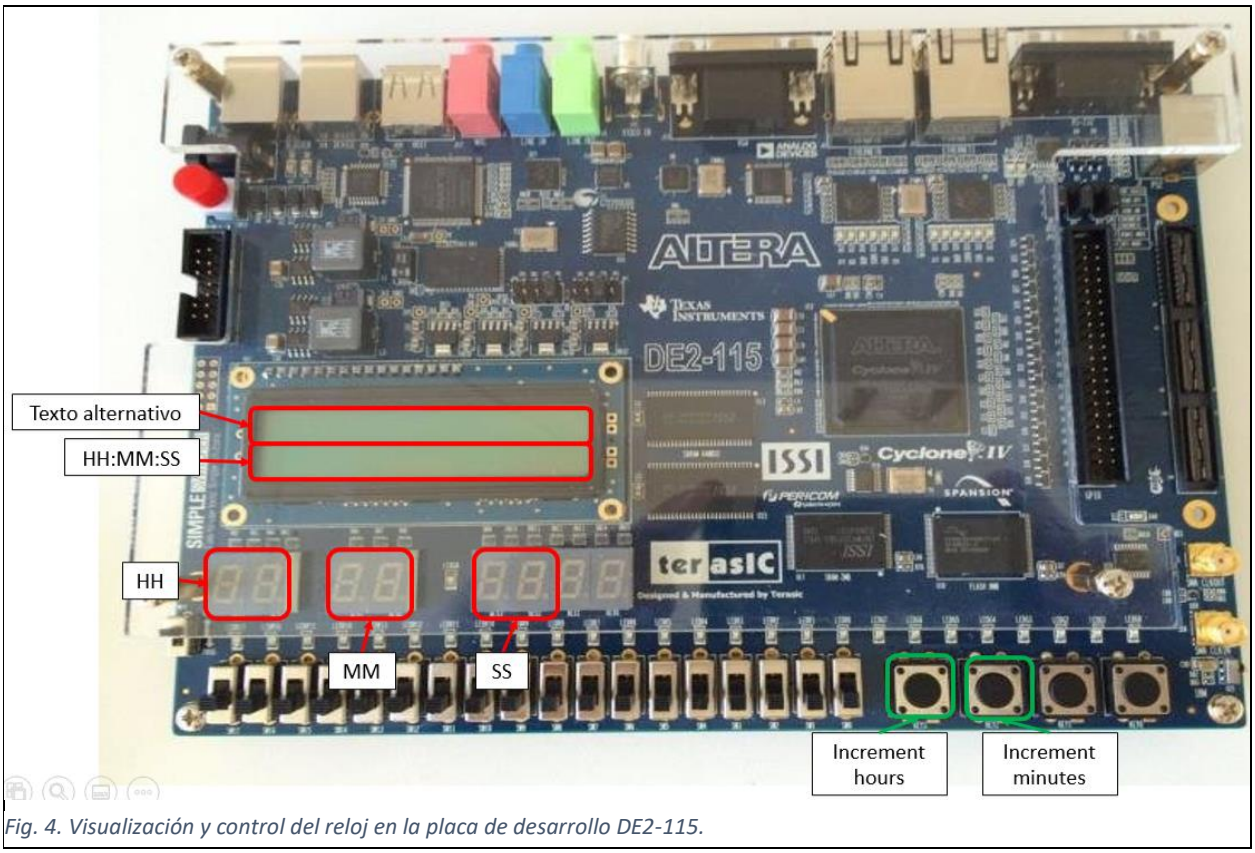

Fig. 4. Visualización y control del reloj en la placa de desarrollo DE2-115.

Visualización en la Pantalla MTL2

Para la implementación del teleprónter se utilizarán dos subrutinas (“int read_VGA_line (FILE *filePointer, char* line)” y “void data_update(alt_up_char_buffer_dev* char_buffer_dev_MTL, char data[TEXT_Y_RES][TEXT_X_RES], char* newdata)”) además de las librerías HAL para realizar la visualización en la pantalla. Además, se configurará el interruptor SW0 para que mediante interrupciones actúe como un *enable* activo a nivel alto del teleprónter. Para ello, se utiliza el archivo “switch_ISR.c” que contiene la función que controla la gestión de la interrupción del interruptor, cambiando una variable global (“VGA_enable”) que se utilizará en el código principal como el *enable*. También se incorpora otra variable global (“frame_refresh”) que nos permitirá contar 2.5 segundos tal y como se ve en la figura 1.

La subrutina “read_VGA_line()” nos permitirá leer una nueva línea mediante la función “getc()” teniendo en cuenta la resolución de la pantalla con un cierto margen definidos de forma global con un “#define”. Esta función devuelve un 0 si todo ha ido correctamente, un 1 si el EOF se encuentra al principio de una línea y 2 si el EOF está en la misma línea que contiene caracteres. De esta forma se podrá añadir una línea de separación cada vez que se cambie de documento de texto.

Esta función comprobará, además, si la última palabra leída encaja dentro de la pantalla y en caso contrario, retrocederá el puntero que controla la lectura del documento y terminará la línea en el último espacio encontrado.

Por otro lado, “data_update()” actualizará la matriz de datos que contiene todas las líneas escritas en la pantalla para poder realizar el *scroll* vertical haciendo un desplazamiento y añadiendo la nueva línea al final. Al mismo tiempo, se irá escribiendo los datos en la pantalla tras limpiar la pantalla.

Cabe comentar que, como una posible mejora y optimización, se podría modificar el código para generar una clase de datos de tipo “struct” que contenga la información necesaria y nos permita trabajar con los datos de la pantalla de forma más ordenada.

Finalmente, la ejecución del teleprónter en el “*main()*” quedará como en la siguiente figura.

```
177         if (frame_refresh == 25)
178         {
179             frame_refresh = 0;
180             if (VGA_enable)
181             {
182                 if (file_status == 0) //proceed normally
183                 {
184                     file_status = read_VGA_line (fh, newline);
185                     if(file_status == 1)
186                     {
187                         data_update(char_buffer_dev_MTL, text_data, separator);
188                         fclose(fh);
189                         file_status = 0;
190                         txt_id ++;
191                         fh = fopen(txt_files[txt_id & 0x3],"r");
192                         if (fh == NULL)
193                             {
194                                 printf("Error loading file");
195                                 return 1;
196                             }
197                     }
198                     else // status 0 or 2 (test the status 2 uncomment lower else)
199                     {
200                         data_update(char_buffer_dev_MTL, text_data, newline);
201                     }
202                 }
203                 /*
204                 else //print a separator line when EOF in case EOF is at the end of the line
205                 {
206 
207                 }*/
208             }
```

Fig. 5. Snippet del código principal que controlará la ejecución del teleprónter dentro de la función main().