Ejercicio 1. Tutorial de Herramientas. Cuestiones Prácticas Co-diseño

Rocabado Rocha, Jose Luis:

Responder las preguntas utilizando el espacio necesario. Formatear al gusto.

Ejercicio 1: Tutorial de herramientas

Primera parte: Tutorial de refresco de uso de las herramientas

1. ¿Qué modificaciones hay que realizar en el código para poder realizar la simulación "Gate-Level"? ¿Por qué?

La simulación "*Gate-Level*" se realiza tras la compilación del proyecto, una vez se ha rutado y emplazado con los parámetros por defecto. Por ello, no es posible cambiar los valores de los parámetros en el código del *TestBench* puesto que el *Hardware* ya se ha "creado".
Para solucionarlo, se debe de seleccionar como parámetros por defecto, los valores que se quieren poner a prueba además de realizar la instanciación de DUT en el *TestBench* sin la parametrización del módulo.

2. ¿Qué diferencias hay entre la simulación RTL y la "Gate-Level"?

La principal diferencia entre ambas simulaciones es que una simulación RTL no tiene en cuenta los retardos de línea. De esta diferencia se puede derivar el problema mencionado en el ejercicio anterior.

3. ¿Cómo se podría modificar la velocidad con que varía la cuenta?

Es posible seleccionar la velocidad de variación de la cuenta cambiando el módulo del contador:i1. Si se tiene en cuenta la frecuencia del sistema de 50MHz, es posible cambiar la frecuencia a la que el oTC se activa.

4. ¿Qué modificaciones del diseño se tendrían que hacer para tener un contador de hexadecimal, en lugar de decimal?

Simplemente se debe cambiar el módulo del contador:i2 por un valor de 16 en lugar de 10.

5. ¿Qué ocurre si se detiene el `contador:i1` en el instante que su salida `oTC` está activa? ¿Cómo solucionar este pequeño fallo?

Puesto que el pin oTC del contador:i1 está conectado al pin iENABLE del contador:i2, el segundo contador estaría contando a la frecuencia del reloj del sistema continuamente.

Para solucionar este fallo de la forma más sencilla, se podría añadir una puerta AND cuyas entradas sean los pines iENABLE y oTC del contador:i1 y la salida se conecte al iENABLE del conador:i2.

De esta forma, en el momento en que el primer contador este desactivado, se desactivará el segundo contador también.

Segunda parte: Tutorial de herramientas

6. ¿De que depende la velocidad de intermitencia del led? ¿Cómo se puede modificar?

```
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"

int main()
{
      printf("Hello from Nios II!\n");
      int count = 0;
      int delay;
      while(1)
      {
            IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, count & 0x01);
            delay = 0;
            while(delay<2000000)
            {
                  delay++;
            }
            count++;
      }
      return 0;
}
```

Fig. 1. Código C que controla la intermitencia de los LED's de la placa de desarrollo DE2-115. Figura 25 del guion de práctica proporcionado en la asignatura.

Si observamos el código de la figura 1 podemos observar que la velocidad de intermitencia depende del bucle *while* que depende de la variable "*delay*".

En cierta forma, este bucle actúa como el contador de la primera parte, contador:i1.

Para cambiar la velocidad simplemente se debe de cambiar el número con el que se compara la variable "*delay*" (módulo del contador).

7. ¿De que manera se puede conseguir que la intermitencia del led se ajuste a un parámetro temporal de N segundos exactos e invariables?

Tal y como se ha comentado en el apartado anterior, tanto el código como el contador:i1 nos permiten variar el tiempo de cambio.
La principal diferencia es que el retraso implementado mediante código C tendrá una frecuencia base que depende de la frecuencia de ejecución de instrucciones en ensamblador por lo que es difícil obtener un retraso de N segundos exactos e invariables.
La única forma sería añadir otro contador HW, con la configuración del contador:i1, donde sabemos que la frecuencia de trabajo es exactamente la del oscilador externo de 50MHz cuyo módulo se podría obtener como:

$$módulo_{contador} = \frac{f_{clk}}{delay(s)}$$

8. ¿Cómo modificar el programa para reflejar en los leds una cuenta binaria de 4 bits? ¿y si se quieren 8 bits de cuenta?

Si volvemos a revisar el código de la figura 1, podemos observar que la función que escribe datos en el puerto paralelo de los LEDs tiene como segundo parámetro de entrada el valor de la variable "contador" junto con una máscara hexadecimal.
Por lo tanto, para reflejar una cuenta binaria de 4 bits, se deben activar los 4 LEDs necesarios cambiando el valor de la máscara por 0x0F.
Del mismo modo, para reflejar una cuenta de 8 bits, se cambia el valor de la máscara por 0xFF.

9. ¿Qué ocurre en la placa después de la realización del punto **58**? ¿Por qué?

Tras el punto 58, se observa que, al contrario que los LEDs, el contador decimal funciona. Esto se debe a que se ha realizado un nuevo emplazamiento del sistema, cuyo *Hardware* a cambiado. Por ello se debe volver a subir el código C adaptando el proyecto de Ecipse al nuevo archivo "nios_system.sopcinfo".

10. Describe que diseño se tiene dentro de la FPGA después del punto **64**.

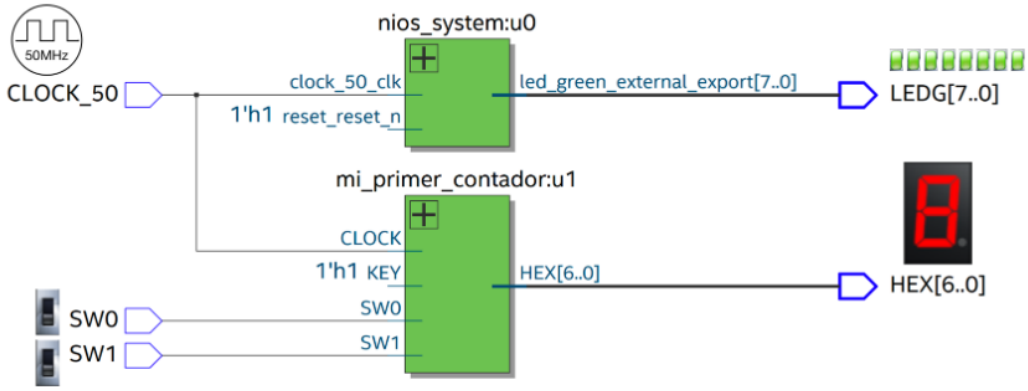

Fig. 2. Diagrama de bloques del diseño final emplazado en la placa de desarrollo DE2-115. Figura 29 del guion de prácticas proporcionada en la asignatura.

Finalmente, el diseño obtenido se corresponde al que se observa en el diagrama de bloques de la figura 2.
Este diseño consiste en el procesador NIOSII, configurado con una memoria *on chip*, un interfaz jtag uart, un timer, puertos paralelos y un *system ID*, que nos permitirá ejecutar el código C que controla la intermitencia de los LEDs verdes que contiene la placa de desarrollo.

De forma paralela también se tiene el contador hexadecimal diseñado en la primera parte de este ejercicio que nos permite controlar el *display* 7-segmentos para contar en base 10.