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

Apellidos y Nombre: Coves Puelma, Paola

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?

No se puede utilizar la misma instanciación para poder realizar una simulación Gate-Level. Por esta razón simplemente hay que quitar el parámetro *fin cuenta*, porque una simulación Gate-Level no tiene parámetros, es una simulación física, ya que el circuito ya es un circuito fijo a nivel de puertas lógicas. Como ejemplo se ilustra la modificación a realizar en *tb_contador.v* en la Figura 1.1 y la correspondiente simulación Gate-Level en la Figura 1.2.

```
`timescale 1ns/1ps;

module tb_contador ();

localparam T = 20;
localparam modulo = 10;

reg CLK;
reg RESET_A;
reg ENABLE;
reg UP_DOWN;
wire [3:0] COUNT;
wire TC;

contador #(.fin_cuenta(modulo)) i1 (
                                    .iCLOCK(CLK),
                                    .iRESET_n(RESET_A),
                                    .iENABLE(ENABLE),
                                    .iUP_DOWN(UP_DOWN),
                                    .oCOUNT(COUNT),
                                    .oTC(TC));
```

```
`timescale 1ns/1ps;

module tb_contador ();

localparam T = 20;
localparam modulo = 10;

reg CLK;
reg RESET_A;
reg ENABLE;
reg UP_DOWN;
wire [3:0] COUNT;
wire TC;

contador i1 (
                                    .iCLOCK(CLK),
                                    .iRESET_n(RESET_A),
                                    .iENABLE(ENABLE),
                                    .iUP_DOWN(UP_DOWN),
                                    .oCOUNT(COUNT),
                                    .oTC(TC));
```

Figura 1.1: Modificación a realizar para una simulación Gate-Level.

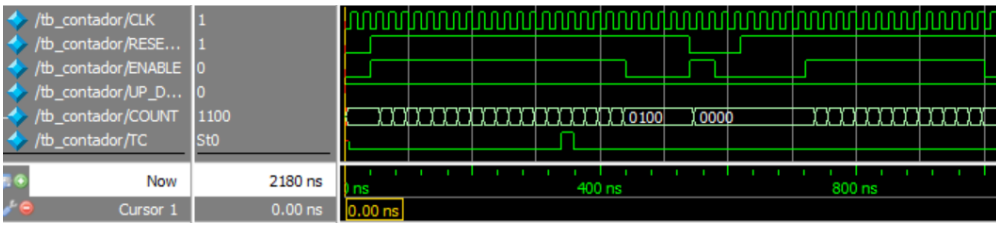


Figura 1.2: Simulación Gate-Level del contador. En esta ocasión ronda el orden de los ns (nano segundos), en contrapartida a una simulación RTL que para este caso es del orden de ps (pico segundos).

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

La simulación RTL (Register-Transfer-Level) se encarga de simular a nivel de transferencia de registro, por lo que no hay información de retardos temporales. Como solo se trata de código fuente la simulación es bastante rápida. En esta se suelen utilizar bloques “always” y declaraciones “assign”, las cuales son sintetizables (se pueden traducir a nivel de puerta).

Mientras que una simulación “Gate-Level”, es una simulación de la netlist compilada y tiene en cuenta los retardos temporales a nivel de puertas. Es decir, simula la lógica combinacional a través de las puertas lógicas (es lógica descrita únicamente por puertas y módulos). Debido a esto la simulación es bastante lenta.

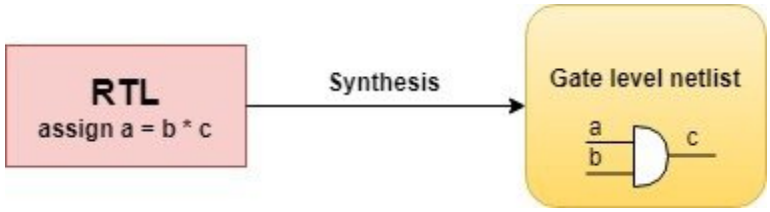


Figura 1.3: Diferencia entre simulación RTL y Gate-Level.

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

Todo diseño de sistema secuencial va controlado por una señal de reloj, que proviene de un oscilador. En la placa DE2 el oscilador del que disponemos tiene una frecuencia de 50 MHz, o lo que es equivalente a que los cambios de las señales se produzcan cada 20 ns. Sin embargo, para utilizar elementos de visualización los cambios inferiores a 50 ms no son apreciables. Debido a esto no podemos utilizar directamente el oscilador de 50 MHz.

Se trata de la descripción de un contador parametrizable, ascendente y descendente. Que el contador sea parametrizable significa que, por medio de un parámetro denominado *fin_cuenta*, se puede establecer el módulo del contador o el número de cuentas que debe realizar. Por lo tanto, modificando dicha variable del módulo *contador.v*, que se puede modificar directamente en la instancia*, contador:i1* en *mi_primer_contador.v*, podemos controlar la frecuencia con la que funciona el *contador:i2*.

Por ejemplo, para que el contador funcione cada medio segundo se deben generar 25 millones de ciclos de reloj ya que:

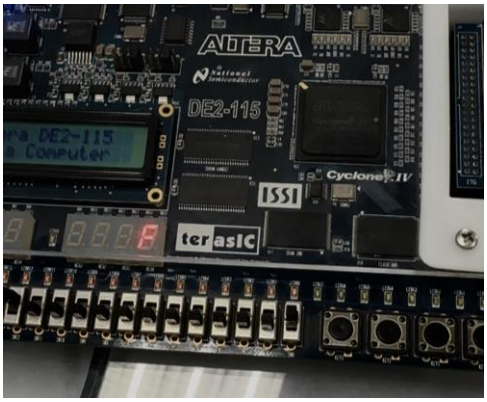
$$20\ ns * 25.000.000 = 0.5\ segundos$$

```
module mi_primer_contador (CLOCK, KEY, SW0, SW1, HEX);

input CLOCK, KEY, SW0, SW1;
output [6:0] HEX;
wire TC;
wire [3:0] COUNT;

contador #(.fin_cuenta(25000000)) i1 (
                                .iCLOCK(CLOCK),
                                .iRESET_n(KEY),
                                .iENABLE(SW0),
                                .iUP_DOWN(1'b1),
                                .oCOUNT(),
                                .oTC(TC));

contador #(.fin_cuenta(10)) i2 (
                            .iCLOCK(CLOCK),
                            .iRESET_n(KEY),
                            .iENABLE(TC),
                            .iUP_DOWN(SW1),
                            .oCOUNT(COUNT),
                            .oTC());

hexTo7seg i3 (.iHEX(COUNT), .o7SEG(HEX));
```

Figura 1.4: Para variar la velocidad de la cuenta se utiliza el parámetro fin_cuenta del contador:i1.

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

Por el momento se está realizando la cuenta de 0 a 9 en decimal (que equivale a lo mismo en hexadecimal), pero necesitamos realizar la cuenta de 0 a F (15) que sería lo que correspondería a un contador hexadecimal. El convertidor de 7 segmentos ya tiene implementada la sentencia *case* para un valor que llega hasta F por lo que no sería necesario tocarlo. En conclusión, habría que cambiar el módulo del *contador_i2* por 16, para que poder convertirlo en un contador hexadecimal.




```
module mi_primer_contador (CLOCK, KEY, SW0, SW1, HEX)

input CLOCK, KEY, SW0, SW1;
output [6:0] HEX;
wire TC;
wire [3:0] COUNT;

contador #(.fin_cuenta(25000000)) i1 (
                                .iCLOCK(CLOCK),
                                .iRESET_n(KEY),
                                .iENABLE(SW0),
                                .iUP_DOWN(1'b1),
                                .oCOUNT(),
                                .oTC(TC));

contador #(.fin_cuenta(16)) i2 (
                            .iCLOCK(CLOCK),
                            .iRESET_n(KEY),
                            .iENABLE(TC),
                            .iUP_DOWN(SW1),
                            .oCOUNT(COUNT),
                            .oTC());

hexTo7seg i3 (.iHEX(COUNT), .o7SEG(HEX));
```

Figura 1.5: Modificación para tener un contador hexadecimal y prueba en la placa DE2.

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

La señal en cuestión, *oTC*, se mantiene activa a nivel alto hasta que la señal *iENABLE* se reactive. Esto provocará que el *contador:i2* realice la cuenta, sin detenerse, a la frecuencia del oscilador de la placa, 50 MHz. Por lo que el cambio de cuenta estará controlado por la señal de reloj del *contador:i1, que es ahora el de la placa*. Para solucionar esto la señal *oTC* solo tendría que poder estar activa durante un ciclo de reloj.

Segunda parte: Tutorial de herramientas

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

Depende de la sentencia *while(delay<2000000)*, y más concretamente de la variable *delay*, en el código led_intermitente.c. Esta sentencia, la cual corresponde a un bucle se ejecutará hasta que la variable entera *delay* sea exactamente el número que le hayamos introducido. Esto causará que cuanto mayor sea el valor que introduzcamos, el retardo de la señal aumentará, por lo que la intermitencia del led se podrá apreciar. Por el contrario, cuando el valor de *delay* es relativamente pequeño no se apreciará intermitencia alguna.

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

Conocemos la relación que existe entre el tiempo y la frecuencia. A partir de esta, se puede obtener una intermitencia en el led que se ajuste a un parámetro *N,* de exactamente un número de segundos. Lo primero que se tendría que realizar es crear dicha variable temporal, tal que *int N = t;* segundos (también se podrían utilizar otros formatos como *double* dependiendo de lo que se desee).

De esta manera, y como conocemos la frecuencia del oscilador que posee la placa DE2, *f=50 MHz.* Por lo que el valor que habría que fijar en el delay correspondería a:

$$while(delay < 50000000 * N)$$

*led_intermitente.c

```c
/*
 * led_intermitente.c
 *
 *  Created on: 7 de oct. de 2022
 *      Author: paocopue
 */
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
int main()
{
printf("Hello from Nios II!\n");
int count = 0;
int N = 0.5;
int delay;
while(1)
{
IOWR_ALTERA_AVALON_PIO_DATA( LED_GREEN_BASE, count & 0x01);
delay = 0;
while(delay<50000000*N)
{
delay++;
}
count++;
}
return 0;
}
```

Figura 1.6: Modificación del código led_intermitente.c para que la intermitencia del led se ajuste a un parámetro temporal de N segundos.

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

Es posible modificar el programa para reflejar en los leds una cuenta binaria de un número determinado de bits, esto se realiza gracias a la operación lógica *AND* y la correspondiente máscara. Para realizar esta tarea se varia la máscara (que activa la cuenta), una mascará de *0x01 (ver Figura 1.6 línea 18)* activará únicamente un *LED* (*LEDG0*), y lo mismo sucederá con una máscara de *0x02* (*LEDG1*).

La Figura a la derecha muestra un ejemplo para una máscara de *0x07,* una cuenta binaria de 3 bits.

Para visualizar una cuenta de exactamente 4 bits, se necesitan cuatro *1's*, que equivale a una máscara de *0x0F*. Por ende, para activar 8 bits son necesarios ocho *1's*, es decir, una máscara de *0xFF*.

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

La placa se reconfigura, de acuerdo con el código actual descrito en Verilog. Esto ocurre porque *Nios II SBT* ha cedido el control del *JTAG* (Si está activo el *JTAG* por Eclipse la placa no podría reconfigurarse). Por lo que la FPGA se reconfigura a través del Programmer de Quartus Primer, sin embargo, el Nios no tiene ningún código para ejecutar y por lo tanto los leds no funcionan. Para que todo en la placa funcione como se desea, Nios II SBT descargará el programa que controla los *LEDS* en la FPGA. En conclusión, lo que le ha ocurrido a la placa es un cambio del hardware empleado.

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

El diseño final que se obtiene dentro de la FPGA corresponde con un contador decimal (Diseño RTL modelizado con Verilog HDL) funcionando en paralelo con el microprocesador Nios II. Dicho contador se encarga de la visualización en el display 7-segmentos, mientras que Nios ejecuta el programa de encendido de LEDS. Los LEDS se conectan al sistema Nios II a través de una interfase paralelo con los puertos de salida. El procesador Nios II ejecuta el programa que ha almacenado en la memoria interna de la FPGA.

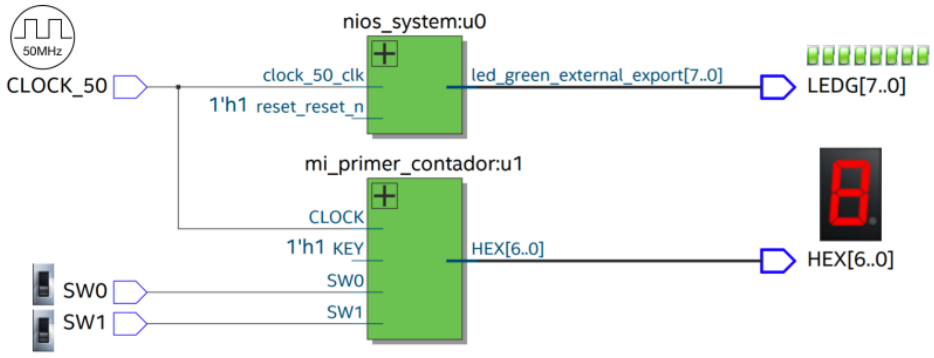


Figura 1.7: Diagrama de bloques.