Práctica 4

Síntesis de circuitos en Verilog

Objetivos

Los objetivos que se persiguen con el desarrollo de esta práctica son:

- Conocer las características de las FPGAs: ventajas e inconvenientes, arquitectura interna, metodologías de diseño.
- Desarrollar las habilidades de diseño de circuitos secuenciales y puramente combinacionales en Verilog.
- Familiarizarse con APIO, un entorno libre para el desarrollo de sistemas hardware basados en FPGA abiertas que dispone de sintetizador Verilog (simulación y síntesis hardware) y todas las herramientas para la configuración de varias tarjetas FPGA.
- Familiarizarse con las FPGA iCE40 de Lattice.

NOTA IMPORANTE:

En los apartados donde hay que desarrollar código (módulos, testbenches, ficheros de configuración), es necesario aportar este código y explicarlo.

En los apartados donde es necesario realizar simulaciones, es necesario aportar capturas de pantalla y explicar lo que se observa.

Tarea 1: Conoce las FPGAs. [1 punto]

1. ¿Sabes lo que es un FPGA?

1.1. Busca información sobre los principales fabricantes de FPGAs (Xilinx/AMD, Altera/Intel, Microchip y Lattice) y compara las arquitecturas de cada uno de ellos destacando las diferencias en sus arquitecturas (celdas básicas, FPGAs SRAM vs FPGAs flash), ventajas e inconvenientes de cada una.

1.2. Las FPGAs son dispositivos flexibles en los que se puede implementar lógica digital compleja como microcontroladores, puertos Ethernet, etc. Explica qué es una “soft macro” o “soft IP core” y qué es un “hard macro” o “hard IP core”.

1.3. Actualmente hay una interesante comunidad de desarrollo entorno a RISC-V. Busca información sobre este proyecto y explica cómo se están implementando en FPGAs. ¿Se puede implementar con soft y con hard IP cores?

1.4. Los SoCs incorporan módulos diversos como, por ejemplo, microcontroladores (ARM, RISC-V, PowerPC). Busca información sobre las arquitecturas más modernas tipo SoC (System on Chip) identificando su arquitectura, tipo de módulos y periféricos que incorporan.

TEORÍA: El profesor introduccirá al lenguaje de descripción de hardware Verilog.

El profesor realizará una introducción teórica a las FPGAs:

- Circuitos digitales
- Ventajas e inconvenientes
- Arquitectura
- Ciclo de vida de desarrollo

Tarea 2: Nuestro primer diseño: ¡Hola Mundo! [2 puntos]

Instalación de APIO en cualquier distribución de Linux (suponemos que Python2 y pip está instalado):

a. Vamos a instalar usando el gestor de paquetes pip de Python, el entorno APIO (https://apiodoc.readthedocs.io/en/stable) en un **subdirectorio local** de nuestro usuario.

```
$ pip install -U apio
```

b. Comprobamos que todo ha ido bien (en algunas distribuciones basta con escribir **apio**):

```
$ apio
```

c. Vamos a instalar todo el *tool-chain* de APIO

```
$ apio install -a
```

d. Vamos a probar con a simular un ejemplo:

```
$ apio examples -d leds
$ cd leds
$ apio init --board icezum
$ apio sim
```

e. Carga el fichero de formas de onda VCD generado por la simulación anterior. Interpreta la simulación.

2. Ahora vamos a crear nuestro primer diseño RTL. Crea el archivo de texto **LED_blink.v** con el siguiente contenido:

```
`timescale 1ns/100ps
module LED_blink (input wire CLK,
                  output wire LED);
  // Declaración de señales
  reg [12:0] data = 0;
  // Asignación: bit 24 de data se asigna a la salida LED
  assign LED = data[12];
  // Contador: incrementa con cada flanco de subida del reloj
  always@(posedge CLK)
    begin
      data <= data + 1;
    end
endmodule
```

2.1. Explica el funcionamiento de LED_Blink.v, detallando qué se hace en cada línea de código. ¿Por qué exactamente vemos parpadear el LED con una frecuencia de más o menos 1 segundo?

2.2. Ahora vamos a sintetizar este código y a cargarlo en nuestra placa.

Crea un archivo de texto de nombre, por ejemplo, **LED_blink.pcf** con el siguiente contenido:

```
set_io LED 104
set_io CLK 21
```

Síntesis y configuración (conecta antes su FPGA al puerto USB de su máquina).

```
$ apio build --board icezum
$ apio upload --board icezum
```

Nota: Podemos ahorrarnos el "--board icezum" si añadimos un archivo **apio.ini** con el siguiente contenido para especificar la placa FPGA que estamos usando (en nuestro caso la "iCEZUM Alhambra"):

```
[env]
board = icezum
```

¿Qué se observas? Explica cómo funciona el diseño en la FPGA y captura un vídeo.

Saca la salida a otro LED (mira el esquema de la tarjeta al final de este guión de prácticas o visite https://github.com/FPGAwars/icezum/wiki)

2.3. Sabiendo cómo funciona el código del ejercicio anterior, modifica el código para realizar un contador usando todos los LEDs de su tarjeta FPGA. Se debe poder apreciar el apagado y encendido de los LEDs.

Tarea 3: Aprendiendo a simular y testear [2 puntos]

3. Ahora vamos a simular el diseño para comprobar que funciona como pretendemos.

3.1. Crea un archivo **LED_blink_tb.v** con el siguiente contenido:

```
`timescale 1ns/100ps
module testbench ();

  reg clk;
  wire led;

  // Generamos reloj
  // Periodo = 5 * timescale = 5 * 1ns  = 5ns
  localparam CLOCK_PERIOD = 5;
  // Inicialmente está a 0
  initial clk = 1'b0;
  // Cambio valor de clk cada 5ns
  always #CLOCK_PERIOD clk = ~clk;

  initial
    begin
      // Vuelca formas de onda en el fichero dump.vcd
      $dumpfile("LED_blink_tb.vcd");
      $dumpvars(0, LED_blink_tb);
      #50000;   // Espera hasta tiempo 50000ns
      $display(" End of simulation time is %d", $time);
      $finish;    // Acaba la simulación
    end

  // Instanciamos el modulo que vamos a simular
  LED_blink DUT (.CLK(clk),
                 .LED(led));

endmodule
```

3.2. Crea un proyecto de simulación que incluya los ficheros creados y simula tu módulo. Captura las formas de onda y explica qué sucede en la simulación.

3.3. Realiza un testbench en Verilog para simular el diseño del apartado **2.3** y comprobar su corrección antes de sintentizar. Posteriormente prueba que el diseño funciona correctamente en la placa. Aporta formas de onda y vídeo de funcionamiento explicando qué sucede.

Tarea 4: Entradas, salidas y varios procesos [2 puntos]

4. Ahora vamos a complicar un poco más nuestro diseño incorporando entradas y utilizando varios procesos para gestionar entradas y salidas.

4.1. Utiliza el código del ejercicio **2.3** y modifica lo necesario para que al pulsar el botón 2 (SW2) de la placa, el contador se reinicie.

4.2. Crea el testbench y sintetiza tu diseño cuando la simulación sea correcta (aporta formas de onda y explicación). El test tiene que simular las entradas y comprobar automáticamente que las salidas se comportan como se espera. Como resultado de la comprobación, el test tiene que imprimir en pantalla un mensaje indicando si el test ha fallado o no.

4.3. Prueba que el diseño funciona en la placa (aporta vídeo).

4.4. El efecto de los rebotes en los interruptores es posible que esté afectando negativamente al sistema. Diseña un módulo basado en un contador que sólo considere el botón cuando se ha pulsado durante al menos 10 ms. Diseña un test para comprobar su funcionamiento y pruébalo en la placa (aporta vídeo).

4.5. Modifica el código para que al pulsar el botón 1 (SW1) el comportamiento del contador cambie. Inicialmente el contador sumará y al pulsar el botón 1, el contador restará, al volver a pulsarlo el comportamiento cambiará de nuevo.

4.6. Testea tu nuevo diseño (aporta formas de onda y explicación). El test tiene que simular las entradas y comprobar automáticamente que las salidas se comportan como se espera. Como resultado de la comprobación, el test tiene que imprimir en pantalla un mensaje indicando si el test ha fallado o no.

4.7. Prueba que el diseño funciona correctamente también en la placa (aporta vídeo).

[Reto JEDI] [1 punto]

4.8. Con todo lo aprendido en los anteriores ejercicios, simula las luces del coche fantástico con los LEDs de la placa. Realiza un test y sintetiza el diseño.

(https://www.youtube.com/watch?v=TeUZYH0is2o).

Tarea 5: Máquinas de estado [3 puntos]

5. Vamos a introducir las máquinas de estados finitos en nuestro diseño.

5.1. Diseña una máquina de estados que se comporte de la siguiente manera:

a. Después de reset, el sistema queda en estado de espera. Se encienden todos los LEDs.
b. Al pulsar los dos interruptores a la vez se entra en modo programación. Se indica encendiendo los dos primeros LEDs 0, 1, 6 y 7.
c. En el modo programación se introduce un comando pulsando sucesivamente el botón de 1 a 4 veces. Al pulsar el botón 2 se almacena el comando. Si el comando es correcto (hemos 1, 2, 3 o 4 veces el botón 1), se encienden los LEDs 2, 3, 4 y 5, se espera aproximadamente 1 segundo y pasamos al modo ejecución del comando seleccionado. Si es incorrecto, se va al estado de espera.
d. Comando 1: parpadea el LED 0 como en el ejercicio **2.1**.
e. Comando 2: parpadean todos los LEDs a la vez de forma.
f. Comando 3: parpadean todos los LEDs como en ejercicio **2.3**.
g. Comando 4: parpadean los LEDs como en el ejercicio **4**.
h. Al pulsar los dos botones en cualquier momento se entra en el estado de programación.

5.2. Implementa el código Verilog que implemente la máquina de estados.

5.3. Desarrolla un testbench que compruebe su correcto funcionamiento.

5.4. Prueba el diseño en la placa.

[Reto JEDI del borde exterior (+5 puntos)]. Juega a Simon

Vamos a implementar una versión sencilla del juego Simon que funciona así:

1. Almacena una secuencia aleatoria de 128 bits en un registro.
2. Al pulsar los dos botones, Simon parpadea todos los LEDs 4 veces de forma visible y comienza el juego.
3. Simon enciende y apaga los LEDs 0 y 4 cuatro veces utilizando los 4 primeros valores de la secuencia:
    - Cuando la secuencia tiene un 0, se enciende el LED 0 y se apaga el LED 4.
    - Cuando la secuencia tiene un 1, se enciende el LED 4 y se apaga el LED 0.
    - El LED permanece encendido aproximadamente un segundo y luego se apagan los dos LEDs.
    - Al acabar la secuencia de 4, Simon enciende todos los LEDs.
4. El jugador introduce los 4 valores de la secuencia. Con el botón 1 indica que se activó el LED 0 y con el botón 2 indica LED 4.
5. Si es correcto, Simon parpadea todos los LEDs 4 veces de forma visible. Los LEDs acaban apagados.
6. Simon vuelve al paso 3 pero esta vez utilizará los bits siguientes de la secuencia.
7. Pulsando los dos botones el juego para en cualquier momento y Simon parpadea todos los LEDs 4 veces de forma visible (los LEDs acaban apagados) y vuelve al estado 2.

Diseña la máquina de estados, escribe el código Verilog junto con su testbench y pruébalo en la placa.

Más información sobre la placa y las herramientas involucradas en:

- APIO: https://github.com/FPGAwars/apio
- YOSYS: http://www.clifford.at/yosys/
- Tarjeta Icezum Alhambra 1.1:
  https://github.com/FPGAwars/icezum/wiki
  https://alhambrabits.com/alhambra/
- EDA Playground: https://www.edaplayground.com/