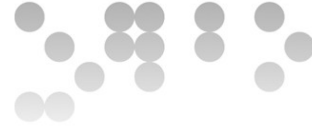


Práctica 2

Presentación

Esta práctica se focaliza en la utilización del lenguaje VHDL para describir un diseño sencillo y poder verificar su correcto funcionamiento mediante bancos de pruebas y simulaciones. La implementación sobre una FPGA concreta tiene que permitir también obtener información adicional de consumo, frecuencia de funcionamiento, etc. En resumen, se trata de seguir un proceso de diseño lo más parecido al real posible, pero en un entorno controlado y con un diseño sencillo. Se trata de poner en práctica los conocimientos adquiridos gracias al estudio del material base facilitado en la asignatura, en concreto los contenidos de los Módulos 4 y 5.

Competencias

- Saber las características generales y las herramientas involucradas en el proceso de diseño de un circuito integrado de aplicación específica (ASIC), específicamente una FPGA.
- Saber implementar funciones lógicas en dispositivos lógicos programables mediante lenguajes de descripción de *hardware*, en concreto, VHDL.
- Saber diseñar sobre FPGAs complejas, aplicando técnicas específicas, y verificar el funcionamiento correcto de la implementación resultante.

Objetivos

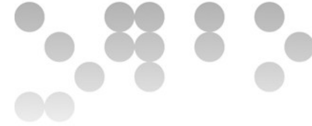
- Saber desarrollar un banco de pruebas específico para un módulo obtenido de internet y comprobar su correcto funcionamiento.
- Implementar un diseño simple aprovechando módulos VHDL externos, modificándolos según nuestras necesidades específicas.
- Poder verificar el funcionamiento de un diseño implementado sobre una FPGA.
- Entender como funcionan las herramientas de diseño avanzado que los fabricantes ponen a nuestro alcance y que van más allá de la simulación funcional clásica.
- Compartir los conocimientos adquiridos con el resto de compañeros del aula mediante el foro del aula.

Recursos

Los recursos que se recomienda usar por esta práctica son los siguientes:

Básicos: Los módulos 4 y 5 de los materiales.

Complementarios: Comentados al texto.



Criterios de valoración

- **Razonad la respuesta en todos los ejercicios. Las respuestas sin justificación no recibirán puntuación.**
- **La valoración se indica en cada uno de los subapartados.**
- **ATENCIÓN: La práctica evalúa el conocimiento del VHDL y de las herramientas de simulación basadas en bancos de pruebas. Si estos dos aspectos no quedan acreditados, la puntuación máxima de la práctica será de D (2 puntos).**

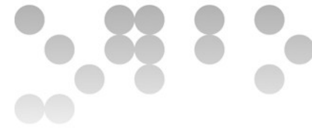
Formato y fecha de entrega

- Hay que entregar la solución **en un fichero ZIP**, que contenga la solución **de la práctica en formato PDF** usando una de las plantillas entregadas conjuntamente con este enunciado, así como el **archivo completo de los diseños** de los ejercicios que requieren codificación VHDL (preferiblemente utilizando la herramienta del Quartus, *Project -> Archive Project*).
- **La memoria en PDF tiene que incluir todo el código en VHDL, tanto del diseño como de los bancos de pruebas** utilizados en las simulaciones, así como **todas las gráficas obtenidas**, junto con los comentarios adecuados. Pensad que si hay algún problema para reproducir los resultados de vuestro diseño, esto será lo único que quedará para defender vuestro trabajo.
- Se tiene que entregar a través de la aplicación de **Entrega y registro de EC** del apartado Evaluación de vuestra aula.
- Para dudas y aclaraciones sobre el enunciado, dirigíos al consultor responsable de vuestra aula, **preferiblemente mediante los foros** (excepto en el supuesto de incluir código o detalles específicos sobre la resolución).
- La fecha tope de entrega es el **21 de diciembre** (a las 24 horas).

Descripción de la Práctica

Esta práctica está compuesta por tres partes que pueden funcionar de manera más o menos independiente:

- Parte 1 (30%): Se trata de modificar el funcionamiento de un módulo VHDL que ya nos viene dado para adaptarlo a nuestras necesidades. A continuación lo tendremos que sintetizar y comprobar mediante un banco de pruebas.
- Parte 2 (40%): Se trata de implementar en una FPGA, y verificar posteriormente su funcionamiento mediante un banco de pruebas, un pequeño diseño que utiliza el bloque modificado en el apartado anterior.
- Parte 3 (30%): En esta última parte se quiere trabajar con herramientas avanzadas disponibles en los entornos de diseño que los fabricantes de FPGAs ponen a nuestro alcance: visualizadores del resultado del proceso de síntesis, herramientas de análisis temporal y de consumo. Cómo seguramente son herramientas nuevas para muchos/-as, se valorará también la información que se comparta en el foro por parte de cada uno. Se trata de **participar activamente para mejorar el conocimiento colectivo.**



Enunciado de la Práctica

Parte 1: Comprovación y modificación de un módulo (30%)

Un LFSR (*Linear Feedback Shift Register*) es un registro de desplazamiento con retroalimentación lineal. Se trata de un registro de desplazamiento en el cual la entrada se obtiene al aplicar una función de transformación lineal a un estado anterior (es decir, a la combinación de los bits del registro). Típicamente, la función de realimentación está basada en los coeficientes de un polinomio módulo 2 (con coeficientes 0 o 1).

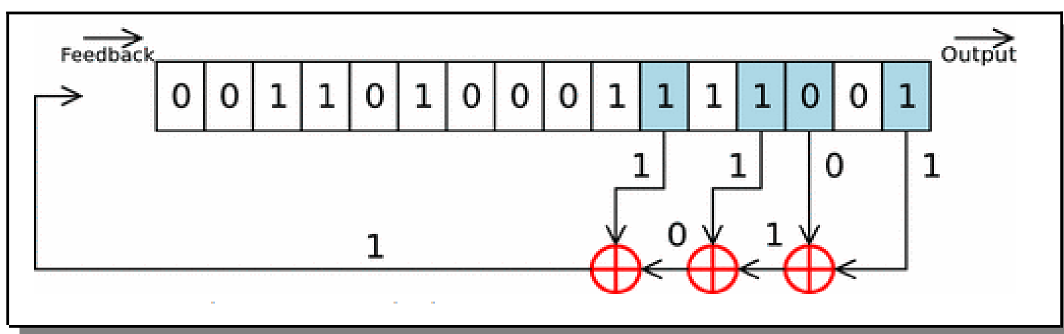
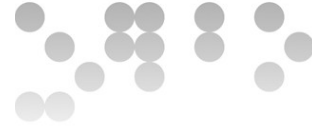


Imagen cortesía de Wikipedia: http://ca.wikipedia.org/wiki/fitxer:Lfsr.gif

El contenido del registro va evolucionando a lo largo del tiempo, generando diferentes valores, hasta que la secuencia vuelve al valor inicial y se repite. Si se escoge bien la transformación, este periodo de repetición llega a ser máximo, 2^n-1 para *n* biestables, y entonces el LFSR tiene un interés especial, puesto que la distribución de los valores de cada uno de sus bits, escogidos por separado, parece ser aleatoria.

Es por este motivo que este tipo de registros se usan a menudo para generar números aleatorios. De hecho, se denominan pseudo-aleatorios, puesto que aunque su comportamiento sea similar a una secuencia aleatoria, en realidad no lo es, puesto que se acaba repitiendo exactamente el mismo comportamiento cuando la secuencia se repite de nuevo.

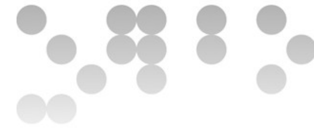
Sin entrar en más detalles teóricos, que dejamos para el lector interesado, nosotros queremos utilizar un LFSR para generar números pseudo-aleatorios. Buscando por internet hemos encontrado el código VHDL (https://surf-vhdl.com/how-to-implement-an-lfsr-in-vhdl/) que se muestra en el listado de la página siguiente, correspondiendo a un LFSR parametrizable de 7 bits de tipo Galois.



```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.
4.  entity lfsr is
5.  generic(
6.    G_M              : integer          := 7          ;
7.    G_POLY           : std_logic_vector  := "1100000") ;  -- x^7+x^6+1
8.  port (
9.    i_clk            : in  std_logic;
10.   i_rstb           : in  std_logic;
11.   i_sync_reset     : in  std_logic;
12.     i_seed           : in  std_logic_vector (G_M-1 downto 0);
13.     i_en             : in  std_logic;
14.     o_lsfr           : out std_logic_vector (G_M-1 downto 0));
15. end lfsr;
16.
17. architecture rtl of lfsr is
18. signal r_lfsr           : std_logic_vector (G_M downto 1);
19. signal w_mask           : std_logic_vector (G_M downto 1);
20. signal w_poly           : std_logic_vector (G_M downto 1);
21.
22. begin
23. o_lsfr  <= r_lfsr(7 downto 1);
24.
25. w_poly  <= G_POLY;
26. g_mask : for k in G_M downto 1 generate
27.   w_mask(k)  <= w_poly(k) and r_lfsr(1);
28. end generate g_mask;
29.
30. p_lfsr : process (i_clk,i_rstb) begin
31.   if (i_rstb = '0') then
32.     r_lfsr   <= (others=>'1');
33.   elsif rising_edge(i_clk) then
34.     if(i_sync_reset='1') then
35.       r_lfsr   <= i_seed;
36.     elsif (i_en = '1') then
37.       r_lfsr   <= '0'&r_lfsr(G_M downto 2) xor w_mask;
38.     end if;
39.   end if;
40. end process p_lfsr;
41.
42. end architecture rtl;
```

Código cortesía de Surf-VHDL: https://surf-vhdl.com/how-to-implement-an-lfsr-in-vhdl/

1. A partir de este código se pide:
   a) Si cogemos cuatro bits cualquiera del LFSR y consideramos que forman un número en representación binaria, podemos decir que el LFSR genera números pseudo-aleatorios de 4 bits. Sabiendo que el código proporcionado genera una secuencia de longitud máxima, ¿cuántos números "aleatorios" podemos generar hasta que se vuelva a repetir la misma secuencia?
   b) Buscad por internet un polinomio generador de grado 64 o superior y modificad el diseño para que trabaje con él por defecto.
   c) Hay que verificar que, efectivamente, el código (con el nuevo polinomio de grado 64) puede generar números pseudo-aleatorios de 4 bits. Por ello hay que diseñar un banco de pruebas que genere secuencias largas (miles de



números) y contar cuántas veces aparece cada uno de ellos, para ver que no hay ninguno que aparezca muchas más veces que los demás o a la inversa, que aparezca mucho menos (pensad, sin embargo, que no será nunca completamente igual, dadas las desviaciones estadísticas). Si repetís este proceso varias veces, una después de otra, ¿obtenéis siempre el mismo resultado? ¿Y si cada vez hacéis un reset, antes de volver a empezar el experimento?

2. Cómo querremos utilizar el módulo LFSR en un diseño más grande, necesitamos hacer algunas modificaciones. En concreto debemos:

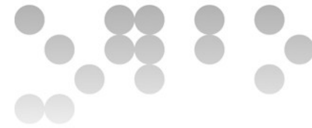
   a) Implementar la arquitectura de la entidad *randomGenerator* que se muestra a continuación, de tal manera que:

   - Utilice el módulo *lfsr_64*, con el polinomio de grado 64, **sin modificarlo**.
   - Añada una máquina de estados que trabaje conjuntamente con el módulo *lfsr_64* para obtener la nueva funcionalidad deseada.
   - Las entradas/salidas de la entidad *randomGenerator* se tienen que conectar a las entradas/salidas del módulo *lfsr_64* o a la máquina de estados.

```
1  entity randomGenerator is
2      port (
3          nReset      : in std_logic;                          -- reset, active low
4          Clk         : in std_logic;                          -- system clock
5
6          seed        : in std_logic_vector(63 downto 0);      -- initial seed value for the LFSR
7
8          GetNumber   : in std_logic;                          -- When 1, the BCD number is consumed and another should be generated
9
10         BCD         : out std_logic_vector(3 downto 0);      -- Valid 0-9 random number
11         Ready       : out std_logic                          -- 1 when BCD contains a valid number
12     );
13 end randomGenerator;
```

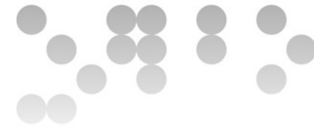
   b) En lugar de esperar una señal para iniciar el proceso de generación de un número aleatorio, el módulo procurará tener uno siempre disponible, de tal manera que se pueda consumir cuanto antes. Añadid funcionalidad siguiente:

   - Queremos que genere números aleatorios equi-probables entre 0 y 9. Si se genera internamente un número fuera de este rango, se tiene que generar otro.
   - No disponga de la señal *i_en* para iniciar la generación del siguiente número aleatorio. En lugar de eso, el código empezará a generar un número aleatorio en el rango [0-9] enseguida que pueda. Cuando lo tenga disponible, pondrá la señal externa *Ready* a 1.
   - Utilizará una entrada *GetNumber*, que se pondrá a 1 cuando se haya leído externamente el número aleatorio disponible. En este momento, *Ready* volverá a 0 y empezará el cálculo de un nuevo número aleatorio para tenerlo lista cuando cuanto antes.
   - Se pide conservar la posibilidad de cargar una semilla inicial (*seed*) para generar la secuencia pseudo-aleatoria, cuando *load* está a alta.



En concreto, se pide:

- Diseñar en VHDL el módulo descrito anteriormente, cumpliendo con las especificaciones indicadas.
- Entrar el diseño y sintetizar-lo sobre una FPGA de Altera de la familia Cyclone IV GX. Se tienen que mostrar los resultados de ocupación (elementos lógicos, número de pines, etc).
- Diseñar un banco de pruebas completo en VHDL para comprobar su funcionamiento y mostrar el resultado de la simulación con Modelsim-Intel.



Parte 2: Implementación de un diseño usando el módulo (40%)

Una pequeña empresa se quiere iniciar en el mundo de las máquinas recreativas y quiere implementar un pequeño sistema de control en una FPGA que pueda servir más adelante como controlador de una máquina de bar o casino comercial. En una primera prueba de concepto, se quiere implementar la clásica máquina tragaperras (https://es.wikipedia.org/wiki/M%C3%A1quinas_tragamonedas) con 3 ruedas giratorias con dibujos de frutas u otros motivos, que da un cierto número de premios de diferente cuantía según la combinación resultante.

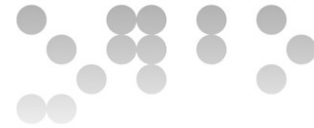
Por normativa, estas máquinas tienen que devolver en premios un porcentaje mínimo de sus ingresos, que para las máquinas de tipos B no puede ser inferior al 80%. En casinos y locales especializados, este porcentaje acostumbra a ser mucho más alto, típicamente por encima del 90%. También es conveniente que se den premios cada pocas jugadas. Aunque sean premios pequeños, como por ejemplo el reintegro de la apuesta, con ellos se consigue mantener el interés del jugador. Otro buen incentivo es el premio principal, que tiene que ser basta alto para interesar incluso a un posible jugador ocasional.

En concreto, el sistema tiene que presentar la siguiente interfaz:

```
entity slotMachine is
port (
   nReset      : in std_logic;                     -- reset, active low
   clock       : in std_logic;                     -- system clock

   newCredit   : in std_logic;                     -- Adds 1 credit per positive pule
   play        : in std_logic;                     -- Plays, if credit is available

   credit      : out std_logic_vector(9 downto 0); -- Accumulated credit/prize

   pay         : in std_logic;                     -- Return the credit or pay the prize (when high)

   slot_L      : out std_logic_vector(3 downto 0); -- Left slot
   slot_M      : out std_logic_vector(3 downto 0); -- Mid slot
   slot_R      : out std_logic_vector(3 downto 0); -- Right slot

   endPlay     : out std_logic;                    -- 1 when the slots stop moving and prize (if any) is ready

   prize       : out std_logic(9 downto 0);        -- Current prize earned (from the last play)

   showPrizes  : in std_logic;                     -- 1 to enter in show prizes mode

   ejectCredit : out std_logic;                    -- When paying a prize, send 1 pulse of 3 clock cicles (at high) per credit
                                                   -- with a separation of 10 clock cicles (at low) between them

   endPay      : out std_logic;                    -- 1 when the prize/credit is fully payed

   configLevel : out std_logic_vector(15 downto 0) -- show the currently implemented options in the module
end slotMachine;
```

Dónde:

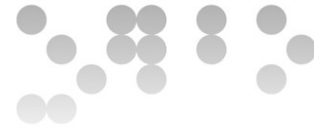
- *nReset* es una señal de inicialización activo a baja . Se usa sólo la primera vez.
- *clock* es el reloj global del sistema, para sincronizar todo el proceso.
- *newCredit* es una señal que recibe un ciclo a alta cuando se inserta una moneda.
- *play* es una señal que se mantiene a 1 mientras el botón de jugar se mantiene pulsado.



- *credit* indica el número de monedas acumulado, disponible por jugar. Incluye las que se han introducido con *newCredit* y los premios no cobrados.
- *pay* es una señal que cuando pasa a alta provoca el pago de un premio (si lo hay) o el reembolso del *credit* disponible.
- *slot_L, slot_M, slot_R* son los símbolos que se muestran en cada carrete de la máquina, en cada momento.
- *endPlay* es una señal que se pone a alta cuando la jugada ha acabado. En este momento, si hay una combinación ganadora en los carretes, la cuantía del premio estará indicada en *prize* y se podrá cobrar pulsando el botón *pay*.
- *prize* indica la cuantía del premio obtenido en la última jugada. Si se cobra (*pay*) o se empieza una nueva jugada (*play*), se pone a cero.
- *showPrizes* es una señal que, al pasar a 1, hace que la máquina entre en modo «mostrar premios». En este modo, en cada ciclo de reloj la máquina muestra una combinación ganadora en las ruedas de números (*slots*) junto con su correspondiente premio en *prize* . Los premios se muestran siguiendo el mismo orden que en la tabla de premios que se puede ver a continuación. En las combinaciones múltiples (p.e. x77) las ruedas con una 'x' se muestran a F (16). Cuando se muestra el último premio, la señal *endPay* se pone a 1.
- *ejectCredit* es una señal que, al pulsar *pay*, emite un pulso por cada moneda a pagar o devolver. El pulso se debe mantener 3 ciclos a alta y 10 a baja.
- *endPay* es una señal que se pondrá a 1 cuando se haya acabado de pagar un premio o de devolver un crédito, y también para salir del modo «mostrar premios».
- *configLevel* es una salida que indica qué funciones están realmente implementadas en el módulo. El significado de cada bit se explicará a continuación.

La tabla de posibles premios que puede dar la máquina es la siguiente:

Combinación	Tipo	Comentarios
777	MAX	Tiene que devolver como mínimo 200 veces la apuesta
111	HI	Tiene que devolver como mínimo 25 veces la apuesta
222	HI	Tiene que devolver como mínimo 25 veces la apuesta
333	HI	Tiene que devolver como mínimo 25 veces la apuesta
555	HI	Tiene que devolver como mínimo 25 veces la apuesta
x77	NEAR	Tiene que devolver como mínimo 10 veces la apuesta
77x	NEAR	Tiene que devolver como mínimo 10 veces la apuesta
xx1	LO	Tiene que devolver como mínimo la apuesta
xx3	LO	Tiene que devolver como mínimo la apuesta
xx5	LO	Tiene que devolver como mínimo la apuesta
xx7	NEAR-LO	Tiene que devolver como mínimo la apuesta



0xx	BANK	Banca. Anula una posible combinación con premio.
x0x	BANK	Banca. Anula una posible combinación con premio.
xx0	BANK	Banca. Anula una posible combinación con premio.

La máquina puede tener varios niveles de implementación, según qué funciones incluya. Estos niveles se describen a continuación.

Nivel 0 [obligatorio]

Este es el nivel más simple de implementación. Las características a incluir en este nivel son:

- La salida *credit* no se utiliza. No se controla la acumulación de crédito y/o premios. Por lo tanto, una jugada consistirá en la llegada, por orden, de las señales *newCredit*, seguida de *play* .
- Al hacer una jugada (*play*), las ruedas de la máquina mostrarán inmediatamente el resultado final, sin pasar por los diferentes valores intermedios.
- La salida *ejectCredit* no se utiliza. Al obtener un premio, no se podrá continuar jugando hasta pulsar la tecla *pay*. La excepción son los premios de reintegro de la apuesta, donde directamente se podrá hacer una nueva jugada con *play*, sin tener que esperar a *newCredit* .
- La entrada *showPrizes* no se utiliza.

Nivel 1 [Recomendado]

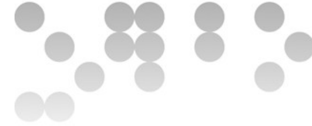
Implementa las funciona básicas del Nivel 0 y añade:

- Implementa la funcionalidad de «mostrar premios» ligada a la entrada *showPrizes*.
- Implementa el control del crédito disponible ligada a la salida *credit*. Los premios se pueden acumular como crédito, en lugar de tener que cobrar-los.

Nivel 2 [Opcional]

Implementa las funcionalidades de los Niveles 0 y 1 y añade:

- Al hacer una jugada (*play*) los carretes de la máquina hacen varias vueltas hasta pararse por orden *slot_R*, *slot_M* y finalmente *slot_L*. Como mínimo, cada carrete tiene que hacer una vuelta adicional al anterior, antes de pararse en el valor final. Además, típicamente se usan velocidades distintas en cada carrete.
- Implementa el pago de premios ligado a la salida *ejectCredit.*



Configuración [obligatorio]

La configuración realmente implementada en la máquina se debe indicar obligatoriamente en la señal de salida *configLevel*. En concreto, cada bit tiene el siguiente significado (1 para implementado, 0 para no implementado):

- 0 (LSB): Nivel 0 implementado.
- 1: Nivel 1 implementado.
- 2: Nivel 2 implementado.
- 3..9: Reservados.
- 10: Premios BANK implementados.
- 11: Premio NEAR-LO implementado.
- 12: Premios LO implementados.
- 13: Premios NEAR implementados.
- 14: Premios HI implementados.
- 15 (MSB): Premio MAX implementado.

Análisis de lsistema

En primer lugar, hay que estudiar la distribución de los premios y su probabilidad de salir, para asegurarse que se cumple con la normativa (mínimo 80% de retorno en premios). También es conveniente dar un premio MAX lo más alto posible y premios menores a menudo para incitar al jugador a continuar apostando (recomendable un mínimo de un premio cada 10 jugadas, en promedio).

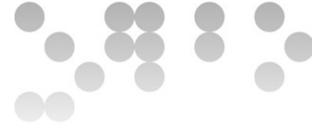
Hay que considerar la utilización de diferentes bloques (aritméticos, máquinas de estados finitos, etc...) para separar la entrada/salida de datos, la gestión de las ruedas de números, el control de los premios, el control del crédito y el control y gestión de todo el sistema.

Las diferentes opciones y niveles de implementación tienen una dificultad bastante variada, que hay que tener en cuenta. Hay que analizar los requerimientos de diseño, cálculo y validación (vía simulación con banco de pruebas) de cada una de ellas.

Realización del diseño

Para la resolución de la práctica, en concreto, se pide:

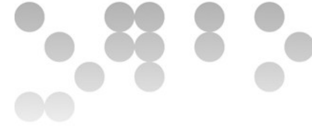
- Hacer **un análisis completo de todo el sistema**, premios y probabilidades, y también de la funcionalidad asociada a los diferentes niveles de implementación, independientemente de si después serán realmente implementados o no en la solución presentada.
- **Como mínimo se tienen que dar los premios MAX y HI.**
- **Como mínimo se debe implementar el Nivel 0**, pero es bastante recomendable implementar también el Nivel 1.



- Se debe estudiar la rentabilidad de la solución finalmente implementada, en base a los premios y probabilidades calculados anteriormente.
- **Diseñar en VHDL la arquitectura correspondiente** a la entidad anterior, haciendo uso del módulo obtenido en la primera parte de la práctica (*randomGenerator*).
- Entrar el diseño y sintetitzarlo sobre una FPGA de Altera de la familia Cyclone IV E. Se tienen que mostrar los resultados de ocupación (elementos lógicos, número de pines, etc).
- **Diseñar un banco de pruebas completo en VHDL para comprobar su funcionamiento y mostrar el resultado de la simulación** con *Modelsim-Intel Starter Edition*. Los valores resultantes se tienen que mostrar en las formas de onda y también por la consola del simulador, para facilitar su análisis (p.e. se podrían mostrar los valores de las ruedas, el premio y el crédito disponible). Hay que hacer comprobaciones adecuadas del correcto funcionamiento del diseño en todos los casos posibles.

Hay que tener presente también que:

- Se valorará la implementación de los niveles/premios adicionales a los mínimos indicados.
- Se valorará especialmente que el banco de pruebas sea completo y verifique los resultados esperados. Sería bueno facilitar el análisis de resultados mediante la utilización de la consola del simulador.



Parte 3: Utilización de herramientas avanzadas EDA-CAD (30%)

En la última parte de la práctica se trata de conocer ciertas herramientas avanzadas que los fabricantes de FPGAs ponen a nuestro alcance y con las cuales a menudo no hemos trabajado antes. Efectivamente, a menudo nos quedamos sólo con las herramientas de síntesis y simulación, pero incluso las versiones gratuitas del software ofrecido por Intel/Altera y Xilinx nos ofrecen mucho más. Vamos a ver si podemos sacarle un buen provecho.

En este apartado se pide:

a) Visualizar a nivel de puertas la implementación de uno de los módulos de nuestro diseño para comprobar si se ajusta a nuestras expectativas. Hacer lo mismo para una máquina de estados. ¿Podéis reconocer los estados de vuestro código VHDL y las transiciones entre ellos? Añadid una imagen de cada tipo a la memoria.

b) Utilizar las herramientas de análisis temporal (*TimeQuest* en el caso de Intel/Altera) para obtener la máxima frecuencia de funcionamiento de vuestro diseño. ¿Podéis encontrar información sobre el camino crítico de vuestro circuito. ¿Os parece razonable? ¿Qué otra información podemos obtener con la herramienta, y como podemos sacar el máximo provecho?

c) Utilizar las herramientas de análisis de consumo y disipación de potencia (*PowerPlay* en el caso de Intel/Altera) para obtener una estimación de la disipación térmica del circuito y su consumo. ¿Qué otra información podéis obtener? ¿Qué se debería hacer para poder mejorar las estimaciones?

d) **Se valorará también especialmente la participación en el foro** de la práctica por parte de cada alumno, para compartir la información que pueda encontrar o generar sobre **como configurar y utilizar estas herramientas avanzadas** de diseño. Por lo tanto, no dudéis en **participar, preguntar y contestar a los compañeros**.