PEC2: Diseño y simulación de circuitos digitales sobre dispositivos programables

Profesores responsables

Profesor responsable:

- Dr. Pere Tuset <peretuset@uoc.edu>

Profesores colaboradores:

- Dr. Francisco Vázquez <fvazquez@uoc.edu>

Presentación

Esta PAC se focaliza en diferentes aspectos de la microelectrónica actual, desde el proceso de diseño de un chip, hasta la implementación de un diseño sobre una FPGA (Field Programmable Gate Array), pasando por las características de los diversos tipos de ASICs (Application specific Integrated Circuits), aspectos económicos, dispositivos lógicos programables, etc. Es muy importante que se conozca el material de base facilitado en la asignatura, en concreto, la PAC contiene un conjunto de cuestiones y problemas relacionados con los contenidos de los Módulos 4 y 5.

Competencias

- Conocer las características generales y las herramientas involucradas en el proceso de diseño de un circuito integrado de aplicación específica (ASIC).
- Conocer los diferentes tipos de ASICs, con sus diferencias, ventajas e inconvenientes.
- Implementar funciones lógicas simples en dispositivos lógicos programables.
- Conocer las características de las FPGAs disponibles en el mercado y poder comparar entre ellas.
- Diseñar sobre FPGAs complejas, aplicando técnicas específicas, y verificar el correcto funcionamiento de la implementación resultante.

Objetivos

- Conocer los diferentes tipos de ASIC y sus características principales.
- Entender el proceso de diseño de un circuito integrado y las herramientas asociadas.
- Conocer las características principales de las FPGAs modernas.
- Aprender a implementar un diseño simple en VHDL, verificarlo y sintetizarlo sobre una FPGA.

Recursos

Los recursos que se recomienda utilizar para esta PAC son los siguientes:

- **Recursos** básicos:
  - **Módulo** 4. "Introducción a los circuitos integrados de aplicación específica", Jordi Riera Baburés, UOC
  - **Módulo** 5. "Dispositivos lógicos programables", Jordi Riera Baburés, UOC
  - "Simulación de sistemas digitales mediante lenguajes descriptores de hardware", Juan Antonio Martínez Carrascal, UOC
  - "VHDL Simulation: Test bench", diapositivas UOC
- **Recursos** complementarios:
  - Quartus® Prime Introduction: Using VHDL Designs , Intel Corporation - FPGA University Program, June 2018
  - Using the Intel® Quartus® Prime Standard Edition Software: An Introduction (80 ') https://www.intel.com/content/www/us/en/programmable/support/training/course/odsw1100.html

Criterios de valoración

- Razonar la respuesta en todos los ejercicios. Las respuestas sin justificación no recibirán puntuación.
- La valoración se indica en cada uno de los subapartados.

Formato y fecha de entrega

- Hay que entregar la solución en un archivo ZIP, que contenga la solución completa de la PEC en formato PDF utilizando 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 herramienta del Quartus, Project -> Archive Project).
- La solución de la PAC en PDF debe 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. Piense que si hay algún problema para reproducir los resultados de su diseño, esto será lo único que quedará para defender su trabajo.
- Se entregará a través de la aplicación de Entrega y registro de AC del apartado Evaluación de su aula.
- Para dudas y aclaraciones sobre el enunciado, diríjase al consultor responsable de su aula.
- La fecha límite de entrega es el **27 de Abril** (a las 23:59 horas).

Descripción de la actividad

Esta PAC está compuesta por diferentes tipos de actividades, en concreto:

- **Preguntas teóricas** (35%): Encontraréis una serie de preguntas, respecto a temas que se han de asimilar, y se trata de razonar por qué cada una de ellas es cierta o falsa. El objetivo es asimilar conceptos mediante el razonamiento de diferentes características de los ASICs, su proceso de diseño y en especial de los dispositivos lógicos programables o FPGAs. Hay que razonar siempre cada una de las opciones de respuesta; las respuestas no razonadas se calificarán con cero puntos.
- **Problemas prácticos** (55%): Se plantean diferentes actividades prácticas relacionadas con el diseño con VHDL y el entorno de desarrollo de Altera para FPGAs, que deberá trabajar a partir de conocimientos del material de la asignatura y / o otras fuentes de información.
- **Cuestión de investigación** (10%): Encontraréis una pregunta abierta, sin una solución cerrada y única, donde deberá hacer búsqueda de información, material o datos, con el fin de trabajar aspectos de la microelectrónica de hoy en día que van más allá de lo que hay en los apuntes.

Enunciado

Preguntas teóricas (35%)

Pregunta 1 (10%)

Determina si cada una de las siguientes afirmaciones son CIERTAS o FALSAS respecto a los ASICs y su proceso de diseño. **Hay que razonar siempre cada una de las opciones de respuesta**; las respuestas no razonadas adecuadamente se calificarán con cero puntos.

a) Para fabricar ASICs no siempre se utiliza la tecnología de fabricación más moderna disponible en cada momento, a pesar de ser la que ofrece mejores prestaciones, menor consumo de potencia y ocupar un área menor.
b) Los ASICs que más se utilizan a nivel industrial son los de tipo *full-custom*, puesto que son los más rápidos y los que tienen menor consumo energético.
c) A cualquier empresa, por pequeña que sea, le puede interesar desarrollar sus propias herramientas EDA-CAD.

Pregunta 2 (10%)

Determina si cada una de las siguientes afirmaciones son CIERTAS o FALSAS respecto a las FPGAs y los dispositivos lógicos programables en general. **Hay que razonar siempre cada una de las opciones de respuesta**; las respuestas no razonadas adecuadamente se calificarán con cero puntos.

a) Las FPGAs basadas en memoria RAM suelen utilizarse en el desarrollo de dispositivos para aplicaciones militares, aviónica y espacio.
b) Para capturar una señal digital de entrada asíncrona en una FPGA es recomendable utilizar un circuito simple formado por dos puertas XOR en cascada.
c) En las FPGA actuales es conveniente implementar las máquinas de estado con una codificación binaria porque este tipo de codificación permite una mayor frecuencia de funcionamiento de la máquina de estados.

Pregunta 3 (5%)

Dado el siguiente fragmento de código VHDL, indica cuál de las siguientes afirmaciones describe mejor la funcionalidad de éste:

A) Es un registro de desplazamiento de 256 bits, con entrada y salida serie, reset asíncrono y señal de habilitación.
B) Es un contador binario con reset asíncrono y señal de habilitación.
C) Es un registro de desplazamiento de N bits, con salida paralelo, reset asíncrono y señal de habilitación.
D) Todas las anteriores son falsas.

```
library ieee;
use ieee.std_logic_1164.all;

entity new_component is

   generic
   (
      NUM_STAGES : natural := 256
   );

   port
   (
      clk      : in std_logic;
      enable   : in std_logic;
      reset    : in std_logic;
      sr_in    : in std_logic;
      sr_out   : out std_logic_vector(NUM_STAGES-1 downto 0)
   );

end entity;

architecture rtl of new_component is

   type sr_length is array ((NUM_STAGES-1) downto 0) of std_logic;
   signal sr: sr_length;

begin

   process (clk, reset)
   begin
      if (reset = '1') then
         sr <= (others=>'0');
      elsif (rising_edge(clk)) then

         if (enable = '1') then
            sr(0) <= sr_in;
            sr((NUM_STAGES-1) downto 1) <= sr((NUM_STAGES-2) downto 0);
         end if;

      end if;
   end process;

   -- Capture the data before it is lost
   sr_out <= sr;

end rtl;
```

Pregunta 4 (5%)

Dado el siguiente fragmento de código VHDL, indica cuál de las siguientes afirmaciones describe mejor la funcionalidad de éste.

A) Es una máquina de Moore con 3 estados y reset asíncrono.
B) Es una máquina de Mealy con 3 estados y reset asíncrono.
C) Es un contador de 2 bits con señal de habilitación y reset asíncrono.
D) Todas las anteriores son falsas.

```
entity new_component is

   port
   (
      clk    : in std_logic;
      input  : in std_logic;
      reset  : in std_logic;
      output : out std_logic_vector(1 downto 0)
   );
end entity;

architecture rtl of new_component is
   type state_type is (s0, s1, s2);
   signal state : state_type;
begin
   process (clk, reset)
   begin

      if reset = '1' then
         state <= s0;
      elsif (rising_edge(clk)) then
         case state is
            when s0=>
               if input = '1' then
                  state <= s1;
               else
                  state <= s0;
               end if;
            when s1=>
               if input = '1' then
                  state <= s2;
               else
                  state <= s1;
               end if;
            when s2=>
               if input = '1' then
                  state <= s0;
               else
                  state <= s2;
               end if;
         end case;

      end if;
   end process;

   process (state, input)
   begin
         case state is
            when s0=>
               if input = '1' then
                  output <= "00";
               else
                  output <= "01";
               end if;
            when s1=>
               if input = '1' then
                  output <= "01";
               else
                  output <= "11";
               end if;
            when s2=>
               if input = '1' then
                  output <= "10";
               else
                  output <= "10";
               end if;
         end case;
   end process;
end rtl;
```

Pregunta 5 (5%)

Dado el siguiente fragmento de código VHDL, indica cuál de las siguientes afirmaciones describe mejor la funcionalidad de éste.

A) Es un detector de flancos de bajada en la señal de entrada y con reset asíncrono.
B) Es un detector de secuencia “101” en la señal de entrada y con reset asíncrono.
C) Es un biestable tipo D activo por flanco con reset asíncrono
D) Todas las anteriores son falsas.

```
process(clk, reset)
begin
   if(reset ='1') then
      data_out <= '0';
      in_delayed  <= '0';
   elsif(rising_edge(clk)) then
      data_out <= not in_delayed and data_in;
      in_delayed  <= data_in;
   end if;
end process;
```

Instalación de Software: Quartus Prime y ModelSim

Para realizar los ejercicios prácticos de la PEC2 y la PRAC2 debemos utilizar el software de Altera. A continuación, se detallan los pasos del proceso de descarga, instalación y configuración del software.

1) En primer lugar debemos crear una cuenta de usuario en el portal de Intel a través del siguiente enlace. A partir de ese momento, podremos entrar en el portal de Intel utilizando nuestra cuenta de usuario (Sign In).
   https://www.intel.com/content/www/us/en/homepage.html
2) A continuación, vamos a descargar el software Quartus Prime Lite Edition a través del siguiente enlace.
   https://fpgasoftware.intel.com/18.1/?edition=lite
   a) Antes de iniciar la descarga, debemos configurar los siguientes elementos:
      - **Select edition**: Lite
      - **Select release**: 18.1
      - **Operating system**: Windows o Linux
   b) En la pestaña **Combined Files**, iniciar la descarga del siguiente archivo:
      - En Windows: Quartus-lite-18.1.0.625-windows.tar
      - En Linux: Quartus-lite-18.1.0.625-linux.tar
   c) Una vez finalizada la descarga, ir al directorio sobre el que se ha realizado la descarga (Downloads) del archivo y descomprimirlo.
   d) Una vez descomprimido el fichero, inicia el proceso de instalación ejecutando:
      - A Windows: **setup.bat**
      - En Linux: **setup.sh**

      En Windows, selecciones el siguiente directorio para la instalación:
      C:\intelFPGA_lite\18.1

ATENCIÓN: Se instalará el **Quartus** Prime, **ModelSim-Altera Intel FPGA Starter Edition** (este no requiere licencia y es gratuito), y el **Cyclone IV device support** (es posible añadir otros dispositivos, pero no es imprescindible para la realización de los ejercicios). No se instalará la aplicación **ModelSim-Altera Intel FPGA** Edition, esto puede provocar problemas de licencia en las simulaciones.

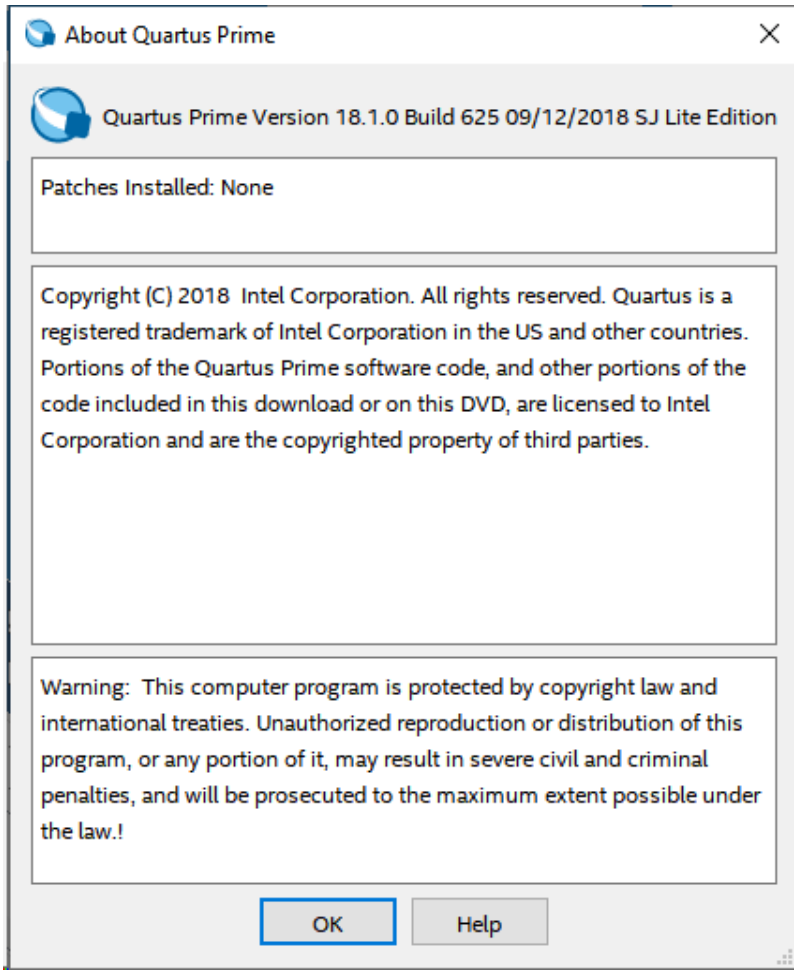
IMPORTANTE: Para evitar problemas, se recomienda no utilizar nombres de directorio con espacios en blanco ni acentos.

   e) Una vez finalizado el proceso de instalación, ejecutar la aplicación Quartus Prime haciendo clic sobre el icono del escritorio. En la aplicación Quartus Prime, configurar el path del ejecutable del ModelSim-Altera de la forma siguiente. Mediante el menú

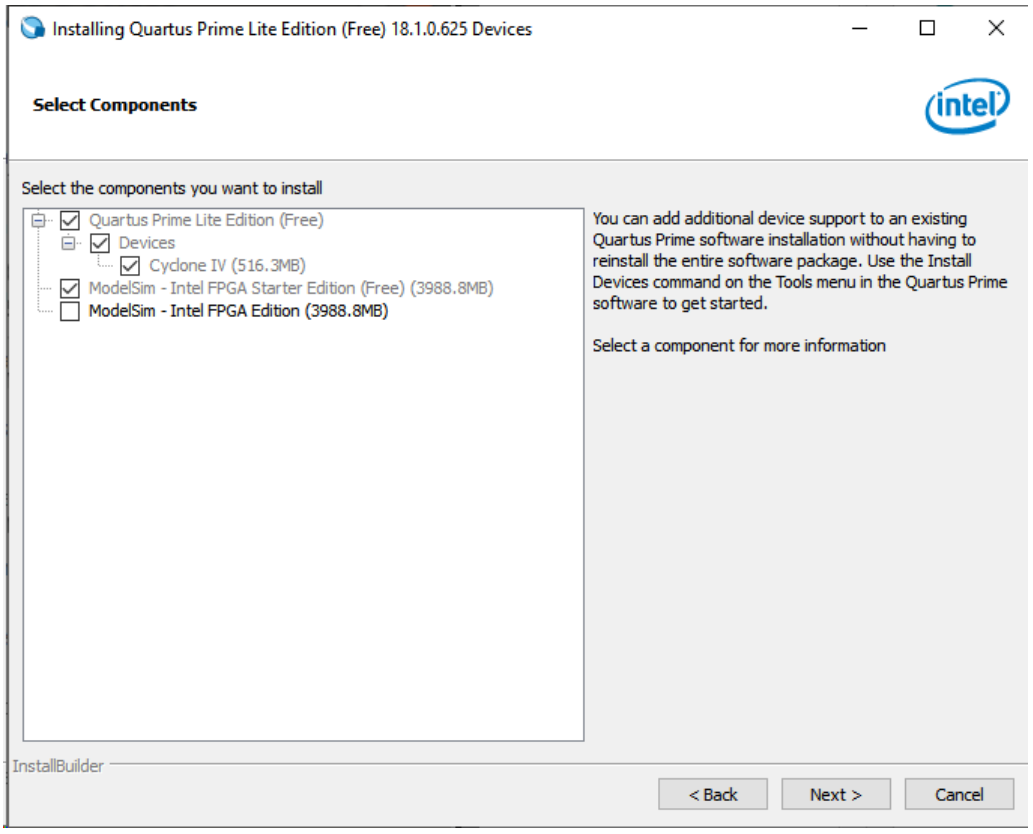
Tools>Options,aparecerá la ventana de configuración. Seleccionar *General* > *EDA Tool*Options, y en el apartado de ModelSim-Altera, introducir el path del ejecutable:

C:\intelFPGA_lite\18.1\modelsim_ase\win32aloem

3) Ejecutar el **Quartus Prime**, ir a *Help* → *About Quartus Prime* y comprobar versión instalada.



4) Ejecutar el **Device Installer** para ver los dispositivos y componentes que hemos descargado e instalado. Si utilizas Windows, puedes ejecutar el Device Installer a través del grupo de aplicaciones "Intel FPGA" que encontrarás en el menú de inicio de Windows. Se abrirá una aplicación donde tenemos que seleccionar el directorio al que hemos descargado nuestros dispositivos. A continuación se abrirá una ventana donde se pueden seleccionar nuevos componentes, y nos muestra también los que ya tenemos. En la siguiente captura de pantalla se puede comprobar que tenemos instalados el *«Modelsim-Intel FPGA Starter Edition"* y las FPGAs de la familia «Cyclone IV».



Problemas prácticos (55%)

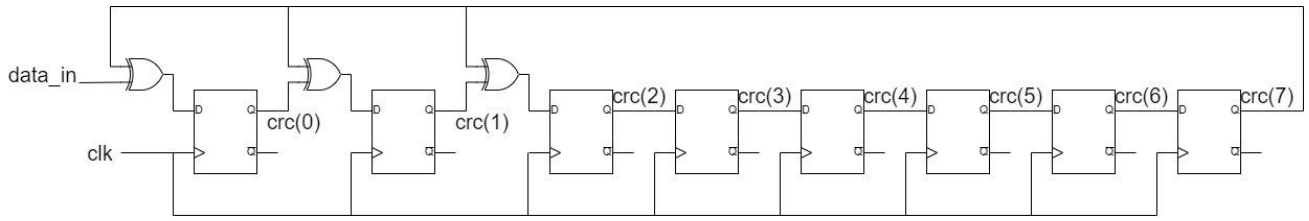
Problema 1 (15%)

Los códigos CRC (Cyclic Redundancy Check) se utilizan en los sistemas de comunicación digital para la detección de errores en la transmisión de datos. El dispositivo que envía el mensaje de datos calcula el valor del código CRC a partir de una división entre el mensaje (dividendo) y un polinomio generador (divisor). El CRC es el resto de la división. El CRC es una secuencia de bits de longitud fija (depende del tipo de polinomio generador) que se añade al final del mensaje durante el proceso de transmisión. El dispositivo que recibe el mensaje calcula el CRC a partir de los datos recibidos y lo compara con el CRC enviado por el dispositivo transmisor. Si el CRC calculado y el CRC recibido coinciden, entonces el mensaje de datos ha sido recibido con éxito. En caso contrario, el dispositivo receptor descarta el mensaje por haber detectado un error en el mensaje.

Existe una gran variedad de polinomios generadores estándar para ser utilizados en aplicaciones específicas. Un ejemplo de polinomio generador típicamente utilizado para el cálculo de CRC de 8 bits es el denominado CRC-8, que se representa analíticamente como:

$$P(x) = x^8 + x^2 + x^1 + x^0$$

Para calcular el CRC-8 de una secuencia de bits de forma eficiente se suele utilizar el circuito digital representado en la siguiente figura. Se trata de un registro de desplazamiento serie formado por 8 flip-flops D y 3 puertas XOR (exclusive OR). La secuencia de bits de entrada sobre la que debe calcularse el CRC entra a través de la señal “data_in”. Cada bit de la secuencia entra en el circuito en un flanco de subida de la señal de clock (“clk”). La principal ventaja de este circuito es que la secuencia de bits de entrada puede ser de longitud variable. Una vez haya entrado el último bit de la secuencia deben inyectarse 8 bits consecutivos a nivel ‘0’ en la señal “data_in”. Por tanto, si la secuencia de datos de entrada tiene una longitud de N bits, entonces el cálculo de CRC tarda N + 8 ciclos de clock. Antes de iniciar el cálculo del CRC de una nueva secuencia de bits es necesario hacer un reset (poner a ‘0’) de todos los flip-flops D del circuito.



El objetivo de este problema consiste en el diseño del código VHDL de un circuito de cálculo de CRC-8 para integrarlo en una FPGA. El siguiente listado (dividido en 2 partes) corresponde al código VHDL incompleto. Está formado por dos procesos: “crc_calculate_pr” y “counter_pr”. En el proceso “crc_calculate_pr” debe calcularse el valor del CRC-8 a partir de un diseño VHDL basado en el circuito anterior. El proceso “counter_pr” debe contar el número de ciclos de clock que se suceden desde que se inicia y hasta que finaliza el cálculo del CRC. Cuando finaliza el cálculo del CRC, el proceso “counter_pr” debe activar la señal “crc_ready”. Las señales de entrada y salida del circuito se describen a continuación:

- La señal “reset” es una entrada para hacer el reset del circuito.
- La señal “size_data” es una entrada de 16 bits para determinar la longitud en bits de la secuencia de datos de entrada.
- La secuencia de datos entra por la señal “data_in”.
- La señal “enable” es una entrada para habilitar el cálculo del CRC y el contaje del número de ciclos de clock. La señal “enable” debe mantenerse a nivel ‘1’ durante el número de ciclos de clock necesarios para realizar el cálculo del CRC (size_data + 8).
- La señal “crc_ready” es una salida para indicar el momento en que finaliza el cálculo del CRC.
- La señal “crc_out” es la salida de 8 bits con el valor del CRC obtenido.

Como se puede comprobar, la descripción VHDL de los procesos “crc_calculate_pr” y “counter_pr” está incompleta en los espacios donde hay comentarios que indican “Completar aquí”. En este problema se pide completar el diseño VHDL y verificar el funcionamiento mediante simulación.

Listado VHDL del fichero "crc_generate.vhd" (parte 1 de 2)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity crc_generate is
6   port (
7         clk          : in std_logic; -- clock
8         reset        : in std_logic; -- active high reset
9         enable       : in std_logic; -- enable CRC calculation
10        size_data    : in unsigned(15 downto 0);  -- size of input data stream in bits
11        data_in      : in std_logic; -- input data stream
12        crc_out      : out unsigned(7 downto 0); -- 8 bit CRC code
13        crc_ready    : out std_logic -- high when the calculation is done
14        );
15  end crc_generate;
16
17  architecture rtl of crc_generate is
18
19     signal count      : unsigned(15 downto 0) := (others => '0');
20     signal crc_reg    : unsigned(7 downto 0) := (others => '0');
21
```

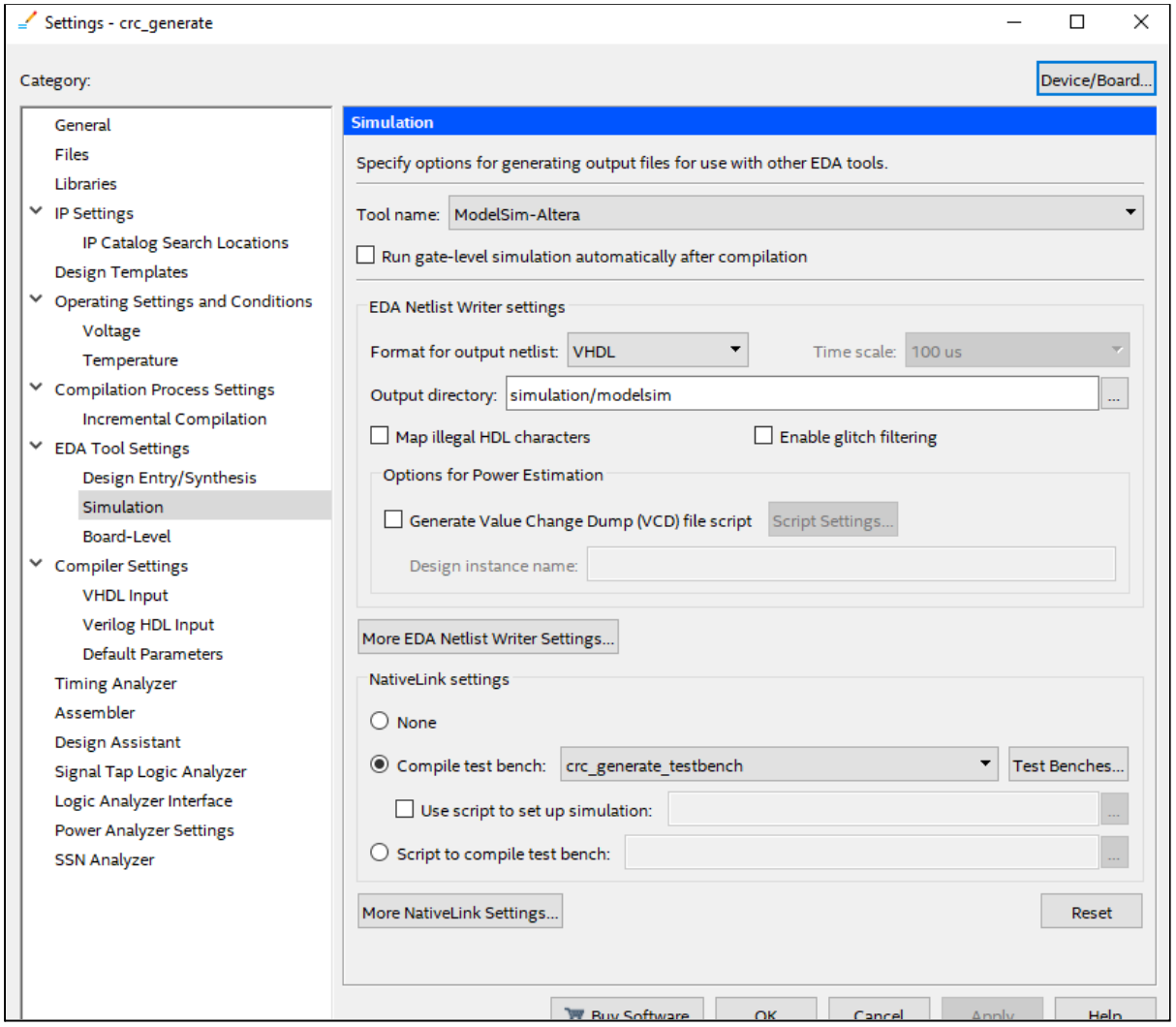
Listado VHDL del fichero "crc_generate.vhd" (parte 2 de 2)

```
23
24  crc_calculate_pr: process(clk, reset)
25  begin
26     if (reset = '1') then
27        crc_reg <= (others => '0');
28     elsif (rising_edge(clk)) then
29        -- Enable the calculation of CRC
30        if (enable = '1') then
31           --crc calculation
32           --------------------
33           -- Completar aquí
34           --------------------
35
36        else
37           crc_reg <= crc_reg;
38        end if;
39     end if;
40  end process crc_calculate_pr;
41
42  crc_out <= crc_reg;
43
44  -- Binary counter of number of bits
45  counter_pr: process(clk, reset)
46  begin
47     if (reset = '1') then
48        count <= (others => '0');
49        crc_ready <= '0';
50     elsif (rising_edge(clk)) then
51        -- Enable the count of bits during CRC calculation
52        if (enable = '1') then
53           -- count the number of bits
54           --------------------
55           -- Completar aquí
56           --------------------
57
58           --Check if CRC calculation ends
59           --------------------
60           -- Completar aquí
61           --------------------
62
63        end if;
64     end if;
65  end process counter_pr;
66
67  end rtl;
```

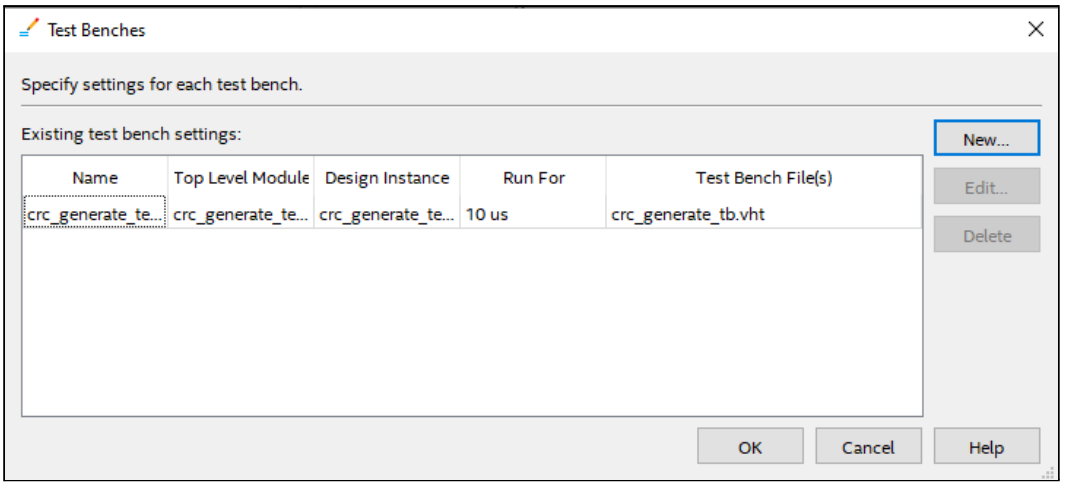
Junto con el enunciado de la PEC, se os ha proporcionado un archivo llamado "2020_2_PAC2_CRC.qar" que corresponde al proyecto a partir del cual se debe completar el diseño del código del "crc_generator.vhd". Con él se pide hacer lo siguiente:

1) Recuperar el proyecto a partir del archivo proporcionado (en Quartus Prime, ir a *Project -> Restore Archived Project*).
2) Completar el código que falta en la descripción del proceso “crc_calculate_pr” con los elementos lógicos necesarios para realizar el cálculo del CRC.
3) Completar el código que falta en la descripción del proceso “counter_pr” para contar el número de ciclos de clock (1 bit por ciclo de clock) necesarios para realizar el cálculo del CRC. La señal “crc_ready” debe activarse a nivel ‘1’ cuando finaliza el cálculo del CRC, es decir, cuando han transcurrido un total de (size_data + 8) ciclos de clock.

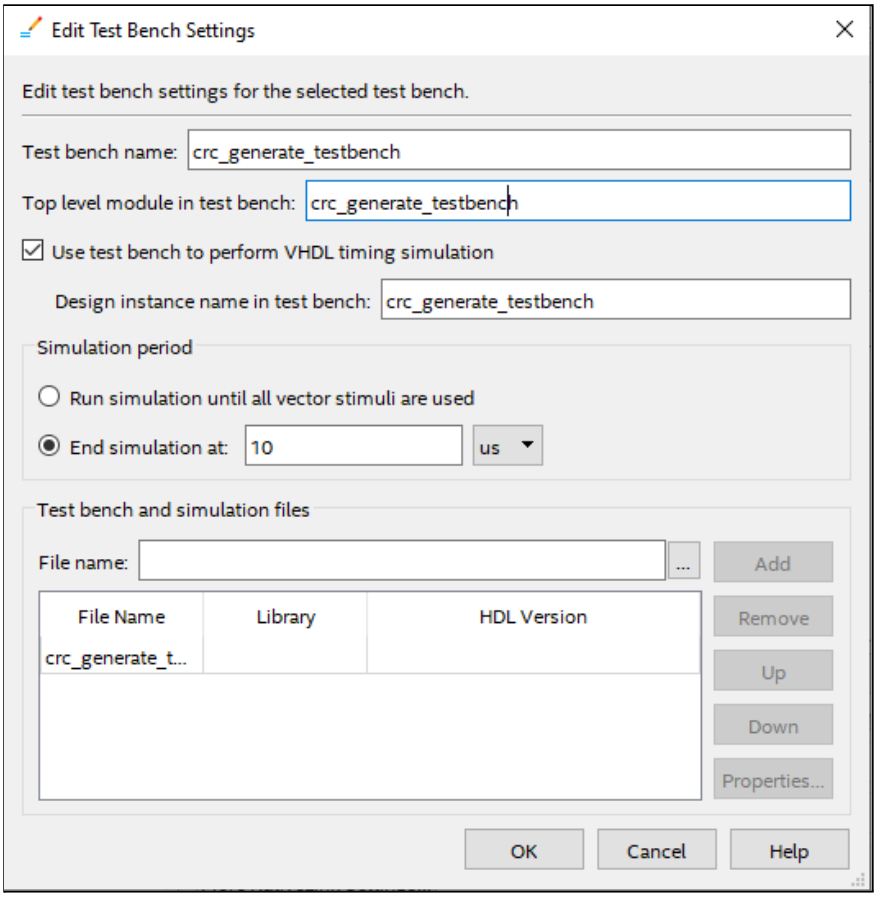
4) Compilar el diseño sobre una FPGA de Altera de la familia Cyclone IV E ejecutando *Processing → Start Compilation*. **Se deben mostrar los resultados de ocupación y explicar brevemente los elementos lógicos, número de pines, etc**.
5) El proyecto incluye un banco de pruebas (fichero "crc_generate_tb.vht") preparado para comprobar el funcionamiento del diseño mediante simulación con *Modelsim-Altera Starter Edition*. Podemos comprobar la configuración de la simulación desde la ventana *Tasks* en la izquierda, seleccionando en el desplegable *RTL Simulation* y entonces la tarea *RTL Simulation → Edit Settings*. Debería quedar algo similar a:



Y seleccionando el botón *Test Benches…*



A continuación, seleccionar "crc_generate_testbench" y clickar en *Edit* para editar la configuración del banco de pruebas para comprobar la configuración. Debería obtenerse lo siguiente.



Responde a las siguientes preguntas sobre el código VHDL del banco de pruebas "crc_generate_tb.vht": ¿cuál es la función realizada por el proceso "clk_process"? ¿qué funciones realiza el proceso "stimuli"?

6) Cerramos todas las ventanas y **lanzamos el ModelSim** (clickar con botón derecho en *RTL Simulation* → clickar en *Start*). Automáticamente, se realizará la simulación y se abrirá una ventana Wave con algunas señales. A continuación, añadiremos todas las señales del "crc_generator" en la ventana Wave. Para ello, en la ventana "sim - Default", seleccionar "uut" para que aparezcan todas las señales internas del "crc_generator" en la ventana "Objects". En la ventana "Objects", seleccionar las señales deseadas, clickar con botón derecho y seleccionar "Add Wave". Finalmente, lanzar de nuevo la simulación ejecutando los siguientes comandos en la consola (ventana "Transcript") de ModelSim:

```
> restart
> run 10 us
```

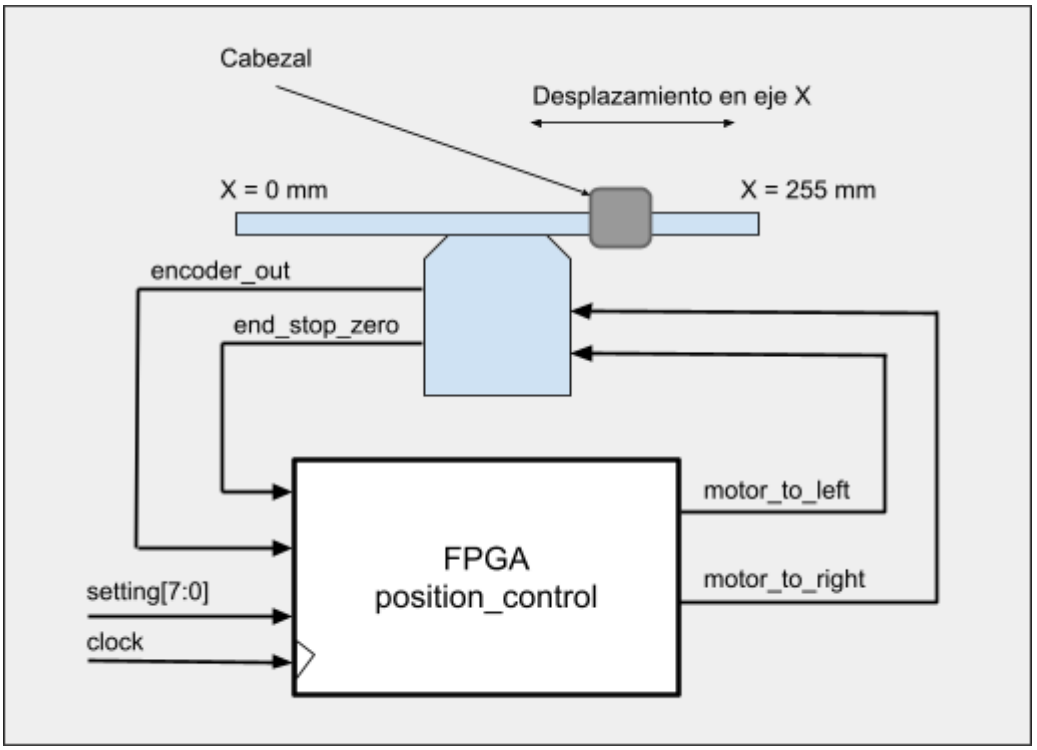
Se debe **explicar el resultado de la simulación** para el test case 1 incluido en el test-bench. Como puede observarse, la secuencia de datos de entrada es de 8 bits con valor 0xAB. Se puede comprobar el resultado del cálculo del CRC con la calculadora de CRC disponible en https://crccalc.com/ (seleccionar Hex como "input type").

7) Se pide **añadir el código que falta en el banco de pruebas** para verificar los test cases siguientes:
   - Test case 2: secuencia de datos de entrada de 16 bits con valor 0xAB00.
   - Test case 3: secuencia de datos de entrada de 16 bits con valor 0x71B8.

   Lanzar la simulación, comprobar el resultado del cálculo del CRC y explicar el resultado de la simulación para ambos test cases.

Problema 2 (20%)

El objetivo de este problema consiste en el diseño de un sistema para controlar la posición del cabezal de una impresora 3D en uno de los tres ejes (X). El sistema de control está formado por un **motor** para desplazar el cabezal en el eje X, un **encoder incremental** para medir el desplazamiento lineal en el eje X, dos **finales de carrera** mecánicos para evitar que el cabezal se desplace fuera del rango entre 0 y 255 mm, un **detector de posición 0 mm**, y un **dispositivo de control de posición** basado en una FPGA. La siguiente figura muestra un diagrama de bloques del sistema con las señales de entrada/salida del dispositivo de control de posición: “encoder_out”, “end_stop_zero”, “setting[7:0]”, “motor_to_right” y “motor_to_left”.



El **motor** permanece parado cuando las señales “motor_to_right” y “motor_to_left” están a nivel lógico “0”. El motor desplaza el cabezal hacia la izquierda cuando la señal “motor_to_left” está a nivel lógico “1” y la señal “motor_to_right” está a “0”. El motor desplaza el cabezal hacia la derecha cuando la señal “motor_to_left” está a nivel lógico “0” y la señal “motor_to_right” está a “1”.

Cuando el cabezal se desplaza hacia la izquierda, el valor de la coordenada X del cabezal disminuye hasta que llega al **final de carrera de 0 mm**. Si el cabezal llega al final de carrera de 0 mm, entonces el **detector de posición de 0 mm** activa la señal “end_stop_zero” a nivel lógico “1”. Cuando el cabezal se desplaza hacia la derecha, el valor de la coordenada X aumenta hasta que el cabezal llega al **final de carrera de 255 mm**.

El **encoder incremental** está formado por una regla codificada que está dividida en una serie de marcas separadas entre sí a una distancia de 1 mm. El encoder incremental genera un pulso (a nivel lógico "1") en la señal de salida "encoder_out" cada vez que el cabezal pasa por una de las marcas. Es decir, cuando el cabezal se desplaza 1 mm (a la derecha o izquierda) se produce un pulso en la señal "encoder_out". La señal "encoder_out" es una señal periódica cuadrada cuya frecuencia depende de la velocidad de movimiento del cabezal. Consideramos que la velocidad de movimiento del cabezal es constante, por tanto, el periodo de la señal "encoder_out" es constante e igual al **tiempo que tarda el motor en realizar un desplazamiento lineal de 1 mm**. Cuando el cabezal está parado, la señal "encoder_out" permanece fija a nivel lógico "0".

El **dispositivo de control de posición** basado en una FPGA se encarga de ajustar el valor de la coordenada X al valor introducido en la entrada digital "setting[7:0]" de 8 bits, que puede tomar valores entre 0 y 255. El dispositivo de control de posición incluye una **máquina de estados** y un **contador up/down** binario (de 8 bits) cuya salida es el valor la coordenada X actual, denominado "x_counter[7:0]".

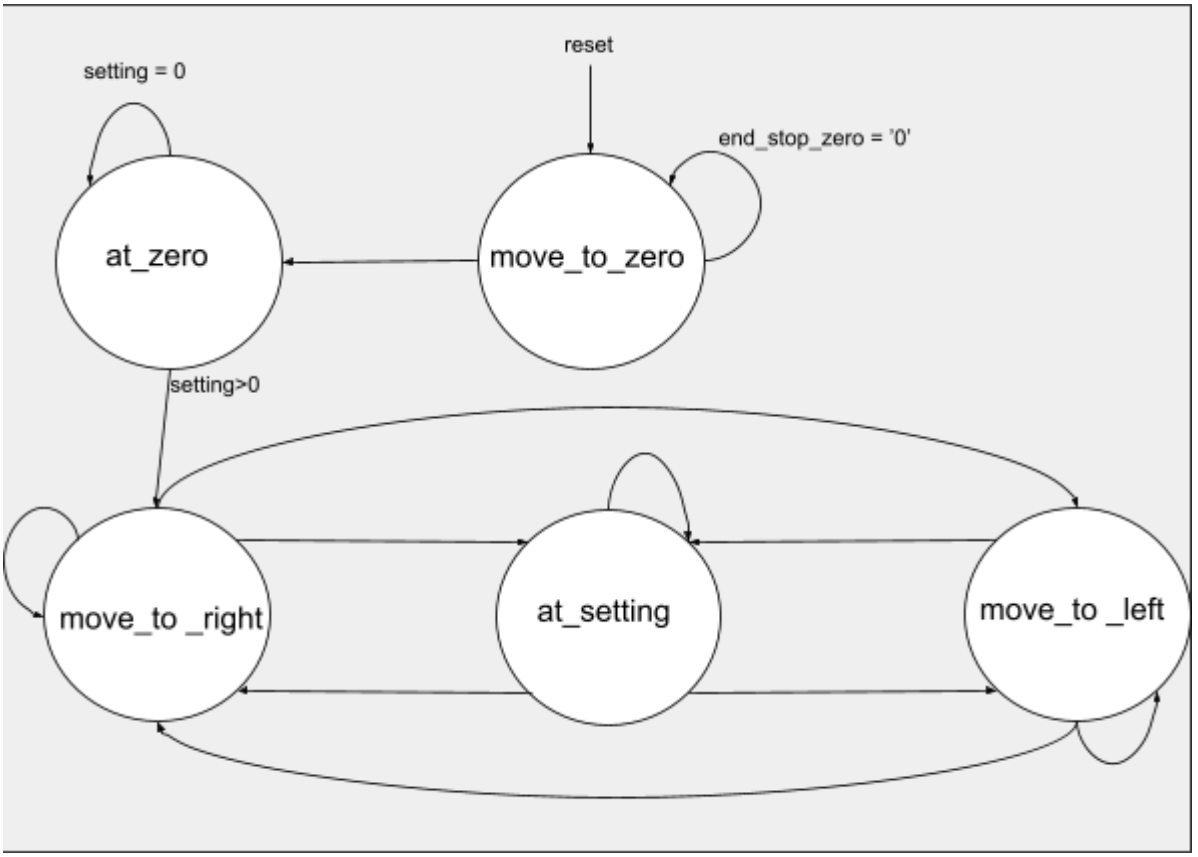
Cada vez que se introduce un nuevo valor en "setting[7:0]", la máquina de estados dará la orden al motor para que mueva el cabezal en el sentido adecuado, mediante sus señales de salida "motor_to_right" y "motor_to_left", según sea el el valor la coordenada X actual del cabezal. Se pueden dar los siguientes casos:

- Si el valor de "setting[7:0]" **es mayor que** "x_counter[7:0]", entonces **el motor debe mover el cabezal a la derecha** y la señal "x_counter[7:0]" se incrementa en 1 mm en cada flanco de subida de la señal "encoder_out" procedente del encoder.
- Si el valor de "setting[7:0]" **es menor que** "x_counter[7:0]", entonces **el motor debe mover el cabezal a la izquierda** y la señal "x_counter[7:0]" se decrementa en 1 mm en cada flanco de subida de la señal "encoder_out" procedente del encoder.
- Si el valor de "setting[7:0]" **es igual que** "x_counter[7:0]", entonces **el cabezal debe permanecer parado**.

Cuando se realiza un reset del dispositivo de control de posición o se pone en marcha, el valor de la coordenada X actual "x_counter[7:0]" conmuta a 0 mm. Al utilizarse un encoder incremental, y no un encoder absoluto, cada vez que se da esta situación debe realizarse una **maniobra de inicialización o ajuste de cero** que consiste en desplazar el cabezal hacia la izquierda hasta que el cabezal llega al final de carrera y se activa la salida "end_stop_zero" del detector de posición de X = 0 mm.

El funcionamiento de la máquina de estados se representa en el siguiente diagrama de estados (incompleto). Se trata de una **máquina de Moore**, que es un tipo de máquina de estados en la que el valor de las salidas sólo depende del estado actual, y no dependen del valor de las entradas. En las **máquinas de Mealy** las salidas dependen del estado actual y de las entradas. Como puede

observarse en el diagrama de estados, en algunas flechas de transición falta indicar la condición necesaria para conmutar de un estado a otro o para permanecer en el mismo estado.



En cada estado, las salidas de la máquina de estados deben hacer lo siguiente:

- En el estado “move_to_zero”:
  - El cabezal se desplaza a la izquierda.
  - La señal “x_counter[7:0]” se inicializa a 0 mm.
- En el estado “at_zero”:
  - El cabezal está parado.
  - La señal “x_counter[7:0]” se mantiene a 0 mm.
- En el estado “move_to_left”:
  - El cabezal se desplaza a la izquierda.
  - La señal “x_counter[7:0]” se decrementa en 1 mm en cada flanco de subida de “encoder_out”.
- En el estado “move_to_right”:
  - El cabezal se desplaza a la derecha.
  - La señal “x_counter[7:0]” se incrementa en 1 mm en cada flanco de subida de “encoder_out”.

- En el estado “at_setting”:
    - El cabezal está parado.
    - La señal “x_counter[7:0]” se mantiene constante.

El siguiente listado (dividido en 3 partes) corresponde al código VHDL del dispositivo de control de posición basado en FPGA. El proceso “rising_edge_detector_pr” detecta los flancos de subida en la señal “encoder_out”, generando un pulso de 1 ciclo de reloj cada vez que se produce un flanco de subida en “encoder_out”. El proceso “fsm_pr” describe las condiciones para cambiar de un estado a otro y el proceso “fsm_outputs” describe la asignación de valores a las salidas de la máquina. El proceso “counter_pr” describe el contador up/down binario de la coordenada X del cabezal.

Como se puede comprobar, la descripción VHDL de los procesos “fsm_pr”, “fsm_outputs” y “counter_pr” está incompleta en los espacios donde hay comentarios que indican “Completar aquí”. En este problema se pide completar el diseño VHDL y verificar el funcionamiento mediante simulación.

Listado VHDL del fichero "position_control.vhd" (parte 1 de 3)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity position_control is
6   port (
7       clk               : in  std_logic;      -- Clock signal
8       reset             : in  std_logic;      -- Reset signal
9       encoder_out       : in  std_logic;      -- Incremental encoder signal
10      end_stop_zero     : in  std_logic;      -- End Stop 0 signal
11      setting           : in  std_logic_vector(7 downto 0); -- Position setting
12      motor_to_left     : out std_logic;      -- Activate motor and move to left
13      motor_to_right    : out std_logic       -- Activate motor and move to right
14      );
15  end position_control;
16
17  architecture rtl of position_control is
18
19      type state_type is (move_to_zero, at_zero, move_to_left, move_to_right, at_setting);   -- states of fsm
20
21      signal state            : state_type;
22
23      signal x_counter        : unsigned(7 downto 0); -- Counter of current position
24      signal counter_en       : std_logic;            -- Enable counter
25      signal counter_up_ndown : std_logic;            -- 1 = count up (to right), 0 = count down (to left)
26      signal counter_rst      : std_logic;            -- Reset counter
27
28      signal in_rise          : std_logic;            -- Rising edge of signal "encoder_out"
29      signal in_delayed       : std_logic;            -- Signal "encoder_out" delayed 1 clock
30
31  begin
32
33  -- Detector of rising edges of signal "encoder_out"
34  rising_edge_detector_pr : process(clk, reset)
35  begin
36      if(reset ='1') then
37          in_rise  <= '0';
38          in_delayed  <= '0';
39      elsif(rising_edge(clk)) then
40          in_rise  <= not in_delayed and encoder_out;
41          in_delayed  <= encoder_out;
42      end if;
43  end process rising_edge_detector_pr;
44
```

Listado VHDL del fichero "position_control.vhd" (parte 2 de 3)

```
44
45  -- Finite states machine: conditions to switch between states
46  fsm_pr: process (clk, reset)
47  begin
48     if (reset ='1') then
49        --------------------
50        -- Completar aquí
51        --------------------
52
53     elsif (clk'event and clk = '1') then
54        case state is
55           when move_to_zero =>
56              --------------------
57              -- Completar aquí
58              --------------------
59           when at_zero =>
60              --------------------
61              -- Completar aquí
62              --------------------
63           when move_to_left =>
64              --------------------
65              -- Completar aquí
66              --------------------
67           when move_to_right =>
68              --------------------
69              -- Completar aquí
70              --------------------
71           when at_setting =>
72              --------------------
73              -- Completar aquí
74              --------------------
75           when others =>
76              state <= move_to_zero;
77
78        end case;
79     end if;
80  end process fsm_pr;
81
82  -- Finite states machine: outputs in each state
83  fsm_outputs: process (state)
84  begin
85     case state is
86        when move_to_zero =>
87              --------------------
88              -- Completar aquí
89              --------------------
90        when at_zero =>
91              --------------------
92              -- Completar aquí
93              --------------------
94        when move_to_left =>
95              --------------------
96              -- Completar aquí
97              --------------------
98        when move_to_right =>
99              --------------------
100             -- Completar aquí
101             --------------------
102       when at_setting =>      motor_to_left <= '0';
103             --------------------
104             -- Completar aquí
105             --------------------
106       when others =>          motor_to_left <= '0';
107                               motor_to_right <= '0';
108                               counter_en <= '0';
109                               counter_up_ndown <= '0';
110                               counter_rst <= '0';
111    end case;
112 end process fsm_outputs;
113
```

Listado VHDL del fichero "position_control.vhd" (parte 3 de 3)

```
114   -- Up/down binary counter of current position (1 count step = 1 mm)
115   counter_pr : process(clk, reset)
116   begin
117      if (reset = '1') then
118         x_counter <= (others => '0');
119      elsif (clk'event and clk = '1') then
120         if (counter_rst = '1') then
121            x_counter <= (others => '0');
122         elsif (counter_en = '1' and in_rise = '1') then
123
124               --------------------
125               -- Completar aquí
126               --------------------
127
128         end if;
129      end if;
130   end process counter_pr;
131
132   end rtl;
```

Junto con el enunciado de la PEC, se os ha proporcionado un archivo llamado "2020_2_PAC2_PositionControl.qar" que corresponde al proyecto a partir del cual se debe completar el diseño del código del "position_control.vhd". Con él se pide hacer lo siguiente:

1) Recuperar el proyecto a partir del archivo proporcionado (en Quartus Prime, ir a *Project -> Restore Archived Project*).
2) Completar el diagrama de estados con las condiciones para cambiar de estado según se detalla en las funcionalidades del sistema de control.
3) Completar el fichero "position_control.vhd" con el código que falta en la descripción de la máquina de estados en el proceso “fsm_pr”.
4) Completar el fichero "position_control.vhd" con el código que falta en la descripción de **las salidas de la máquina de estados** en el proceso “fsm_outputs”. Deben asignarse los valores adecuados a las salidas (motor_to_left, motor_to_right, counter_en, counter_up_ndown, counter_rst) en cada estado.
5) Completar el fichero "position_control.vhd" con el código que falta en la descripción del **contador up/down binario** en el proceso “counter_pr”. Deben asignarse los valores adecuados a la salida del contador (x_counter) para cada una de las condiciones.
6) Compilar el diseño sobre una FPGA de Altera de la familia Cyclone IV E ejecutando *Processing → Start Compilation*. Se deben mostrar los resultados de ocupación y explicar brevemente los elementos lógicos, número de pines, etc.
7) El proyecto incluye un banco de pruebas (fichero “position_control_tb.vht”) que debe completarse para verificar el funcionamiento del diseño del "position_control.vhd" mediante simulación RTL con *Modelsim-Altera Starter Edition*. **Se pide añadir el código que falta en el fichero “position_control_tb.vht”** para verificar los **4 test cases** siguientes:
    a) Maniobra de inicialización a 0 mm después del reset.
    b) Mover el cabezal a la posición 45 mm en la coordenada X.
    c) Mover el cabezal a la posición 100 mm.
    d) Mover el cabezal a la posición 10 mm.

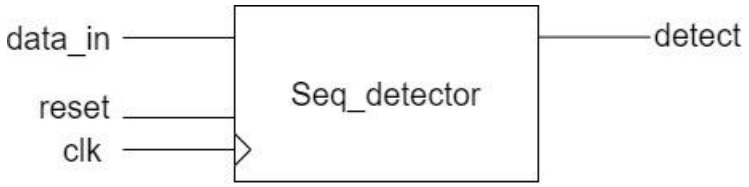
8) **Lanzar la simulación con ModelSim** (en Tasks, seleccionar *RTL Simulation* → clickar con botón derecho en *RTL Simulation* → clickar en *Start*). Si es necesario añadir más señales a la ventana Wave, en la ventana “sim - Default” debe seleccionarse “uut” para que aparezcan todas las señales internas del “position_control” en la ventana “Objects”. En la ventana “Objects”, seleccionar todas las señales deseadas, clickar con botón derecho y seleccionar “Add Wave”. Finalmente, lanzar de nuevo la simulación ejecutando los siguientes comandos en la consola (ventana “Transcript”) de ModelSim:

    > restart
    > run 25 us

ATENCIÓN: Se debe explicar el resultado de la simulación de cada test case.

Problema 3 (20%)

Los detectores de secuencia son circuitos digitales que suelen utilizarse en la sincronización de receptores en sistemas de comunicación, y en seguridad, por ejemplo, para el chequeo de contraseñas o palabras clave. Un detector de secuencia es un circuito secuencial síncrono que toma como entrada una secuencia de bits serie y activa una salida a nivel '1' cada vez que se detecta una secuencia de bits predefinida en la entrada. La siguiente figura muestra las entradas y salidas del detector de secuencia. Tiene una señal de "clock" y una señal de "reset". Cada bit de entrada por la señal "data_in" dura un ciclo de clock. La señal de salida "detected" se activa a nivel '1' durante un ciclo de clock al detectarse la secuencia.

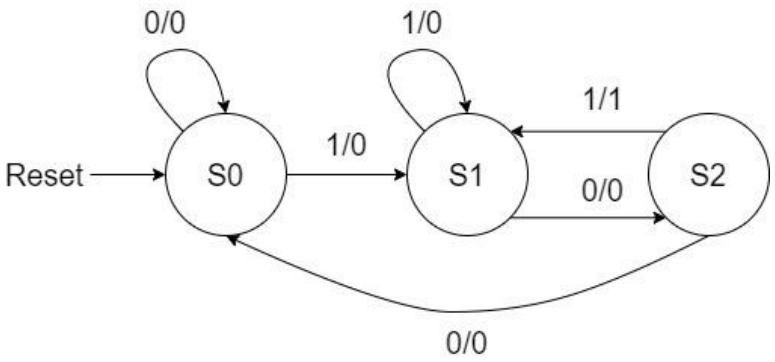


El objetivo de este problema consiste en el diseño VHDL de un detector de secuencia "11011". El detector de secuencia debe permitir el solapamiento de secuencias, es decir, que los últimos bits de una secuencia deben considerarse como válidos para la siguiente secuencia. A continuación, se muestra un ejemplo del funcionamiento que debe tener el circuito con una posible secuencia de bits en la entrada "data_in" y la activación de la señal "detected" cuando se detecta la secuencia predefinida. En azul se marcan las secuencias correctamente detectadas, en marrón se marca el inicio de una secuencia incompleta, y en verde se marca el solapamiento entre dos secuencias completas.

data_in = 11011000110100110110100
detected = 00001000000000000010010 0

El diseño del detector de secuencia debe basarse en una máquina de estados Mealy. Recuérdese que en la máquina Mealy las salidas dependen del estado actual y del valor de las entradas. La máquina debe tener 5 estados y debe establecer el estado inicial al activarse la señal de reset a nivel '1'. El funcionamiento de la señal de reset debe ser asíncrono.

En la siguiente figura se muestra un ejemplo de máquina de estados Mealy para la detección de la secuencia "101". La máquina del ejemplo tiene 3 estados, siendo S0 el estado inicial. En cada flecha de transición entre estados se indica una pareja de valores "X/Y", donde X es el valor de la entrada "data_in", e Y es el valor de la salida "detected". Fíjese que la salida "detected" se activa a '1' al pasar del estado S2 a S1, por tanto, el estado de destino es el S1 ya que el último 1 de la secuencia "101" puede convertirse en el primer 1 de otra posible secuencia detectada. En estos detectores, en caso de haber detectado parcialmente la secuencia y que el siguiente bit no sea lo que se espera, no necesariamente hay que volver al estado inicial ya que es posible que lo que se lleve detectado hasta ahora sea una subsecuencia más corta de la que se pretende detectar.



En el ejemplo anterior, el estado inicial es S0. La máquina permanece en S0 mientras la entrada sea '0'. Cuando la entrada es '1', la máquina conmuta al estado S1 (de momento se ha recibido un solo bit a '1'). La máquina permanece en S1 mientras la entrada sea '1'. Cuando la entrada es '0', la máquina conmuta al estado S2 (de momento se ha recibido una subsecuencia "10"). Finalmente, si la entrada es '1', la máquina conmuta a S1 (se ha recibido la secuencia completa "101"), pero si la entrada es '0', entonces la máquina vuelve al estado inicial S0.

Junto con el enunciado de la PEC, se os ha proporcionado un archivo llamado "2020_2_PAC2_SeqDetector.qar" que corresponde al proyecto a partir del cual se debe completar el diseño del código del detector de secuencia (seq_detector.vhd). Con él se pide hacer lo siguiente:

1) Diseñar el diagrama de estados de una máquina Mealy para el detector de secuencia "11011" indicando claramente el estado inicial y las condiciones de transición y salida de cada estado. Se recomienda empezar dibujando los estados que requiera y las transiciones que corresponden a la correcta detección de la secuencia objetivo.
2) Recuperar el proyecto a partir del archivo proporcionado (en Quartus Prime, ir a *Project -> Restore Archived Project*) y diseñar el código VHDL de la máquina de estados como circuito digital síncrono con las señales indicadas en el esquema anterior. Para ello, debe completarse el fichero "seq_detector.vhd" con el código que falta.
3) El proyecto incluye un banco de pruebas (fichero "seq_detector_tb.vht") que debe completarse para verificar el funcionamiento del diseño del "seq_detector.vhd" mediante simulación RTL con Modelsim-Altera Starter Edition. Se pide añadir el código que falta en el banco de pruebas para verificar que detecta correctamente la secuencia objetivo.
4) Lanzar la simulación con ModelSim (en Tasks, seleccionar *RTL Simulation* → clickar con botón derecho en *RTL Simulation* → clickar en *Start*) y explicar el resultado.

Cuestión de investigación (15%)

Comienza la respuesta a esta cuestión en una hoja aparte.

Cuestión: El tamaño y la potencia de las modernas FPGAs de los grandes fabricantes, como Altera, Xilinx y Microsemi, ofrecen la posibilidad de integrar un procesador en el interior de las FPGAs. Esto permite mover parte de las funcionalidades del hardware de la FPGA al software del procesador, especialmente los elementos de control y gestión del sistema, lo cual puede reducir los costes de fabricación, los tiempos de desarrollo y el tamaño de las placas, e incluso mejorar el rendimiento del sistema gracias a la reducción de los retardos de comunicación entre el procesador y los periféricos.

Se propone realizar un pequeño ejercicio de investigación para responder a las siguientes preguntas:

- ¿Qué es un procesador de tipo “hard-core”? ¿Y un procesador de tipo “soft-core”?
- ¿Cuáles son las ventajas y los inconvenientes de ambos tipos de procesadores?
- ¿Cuáles son los principales procesadores de tipo “soft-core” open-source?
- Selecciona un procesador de tipo “soft-core” open-source, que haya sido desarrollado para aplicaciones espaciales, y explica brevemente sus características básicas, por ejemplo, arquitectura, buses, número de bits, periféricos, etc.

Debes incluir en la respuesta las referencias originales en las que te has basado para hacer el ejercicio de investigación.